
What's New in Python

Release 3.13.0b3

A. M. Kuchling

julho 17, 2024

Python Software Foundation
Email: docs@python.org

Sumário

| | | |
|----------|--|-----------|
| 1 | Resumo – Destaques da versão | 3 |
| 2 | Novas funcionalidades | 4 |
| 2.1 | Um interpretador interativo melhor | 4 |
| 2.2 | Mensagens de erro melhoradas | 5 |
| 2.3 | Semântica de mutação definida para <code>locals()</code> | 6 |
| 2.4 | Coleta de lixo incremental | 6 |
| 2.5 | Suporte para plataformas móveis | 6 |
| 3 | Compilador JIT experimental | 7 |
| 4 | CPython com threads livres | 7 |
| 5 | Outras mudanças na linguagem | 8 |
| 6 | Novos módulos | 9 |
| 7 | Módulos melhorados | 10 |
| 7.1 | <code>argparse</code> | 10 |
| 7.2 | <code>array</code> | 10 |
| 7.3 | <code>ast</code> | 10 |
| 7.4 | <code>asyncio</code> | 10 |
| 7.5 | <code>base64</code> | 11 |
| 7.6 | <code>copy</code> | 11 |
| 7.7 | <code>dbm</code> | 11 |
| 7.8 | <code>dis</code> | 11 |
| 7.9 | <code>doctest</code> | 12 |
| 7.10 | <code>email</code> | 12 |
| 7.11 | <code>fractions</code> | 12 |
| 7.12 | <code>gc</code> | 12 |
| 7.13 | <code>glob</code> | 13 |
| 7.14 | <code>importlib</code> | 13 |
| 7.15 | <code>io</code> | 13 |
| 7.16 | <code>ipaddress</code> | 13 |
| 7.17 | <code>itertools</code> | 13 |
| 7.18 | <code>marshal</code> | 14 |
| 7.19 | <code>math</code> | 14 |
| 7.20 | <code>mimetypes</code> | 14 |
| 7.21 | <code>mmap</code> | 14 |

| | | |
|-----------|--|-----------|
| 7.22 | opcode | 14 |
| 7.23 | os | 14 |
| 7.24 | os.path | 15 |
| 7.25 | pathlib | 15 |
| 7.26 | pdb | 16 |
| 7.27 | queue | 16 |
| 7.28 | random | 16 |
| 7.29 | re | 16 |
| 7.30 | site | 16 |
| 7.31 | sqlite3 | 16 |
| 7.32 | statistics | 17 |
| 7.33 | subprocess | 17 |
| 7.34 | sys | 17 |
| 7.35 | tempfile | 17 |
| 7.36 | time | 17 |
| 7.37 | tkinter | 17 |
| 7.38 | traceback | 18 |
| 7.39 | types | 18 |
| 7.40 | typing | 18 |
| 7.41 | unicodedata | 18 |
| 7.42 | venv | 19 |
| 7.43 | warnings | 19 |
| 7.44 | xml.etree.ElementTree | 19 |
| 7.45 | zipimport | 19 |
| 8 | Otimizações | 19 |
| 9 | Módulos e APIs removidas | 19 |
| 9.1 | PEP 594: baterias mortas (e remoção de outros módulos) | 19 |
| 9.2 | configparser | 21 |
| 9.3 | importlib | 21 |
| 9.4 | locale | 21 |
| 9.5 | logging | 21 |
| 9.6 | pathlib | 21 |
| 9.7 | re | 21 |
| 9.8 | turtle | 21 |
| 9.9 | typing | 22 |
| 9.10 | unittest | 22 |
| 9.11 | urllib | 22 |
| 9.12 | webbrowser | 22 |
| 10 | Novas descontinuações | 23 |
| 10.1 | Remoção pendente no Python 3.14 | 25 |
| 10.2 | Remoção pendente no Python 3.15 | 26 |
| 10.3 | Remoção pendente no Python 3.16 | 27 |
| 10.4 | Remoção pendente em versões futuras | 27 |
| 11 | Mudanças no bytecode do CPython | 29 |
| 12 | Mudanças na API C | 29 |
| 12.1 | Novas funcionalidades | 29 |
| 13 | Mudanças na construção | 32 |
| 14 | Portando para o Python 3.13 | 32 |
| 14.1 | Mudanças na API Python | 33 |
| 14.2 | Mudanças na API C | 33 |
| 14.3 | APIs C removidas | 35 |
| 14.4 | APIs C descontinuadas | 36 |

| | |
|--|-----------|
| 14.5 Remoção pendente no Python 3.14 | 37 |
| 14.6 Remoção pendente no Python 3.15 | 38 |
| 14.7 Remoção pendente em versões futuras | 38 |
| 15 Mudanças em teste de regressão | 39 |
| Índice | 40 |

Editor

Thomas Wouters

Esse artigo explica os novos recursos no Python 3.13, comparado ao 3.12.

Para detalhes completos, veja o changelog.

Ver também:

PEP 719 – Agendamento de lançamento do Python 3.13

Nota: Os usuários de pré-lançamento devem estar cientes de que este documento está atualmente em forma de rascunho. Ele será atualizado substancialmente à medida que o Python 3.13 caminha para seu lançamento estável, portanto vale a pena conferir mesmo depois de ler as versões anteriores.

1 Resumo – Destaques da versão

Python 3.13 beta é o pré-lançamento da próxima versão da linguagem de programação Python, com uma mistura de mudanças na linguagem, na implementação e na biblioteca padrão. As maiores mudanças na implementação incluem um novo interpretador interativo e suporte experimental para eliminar a trava global do interpretador (**PEP 703**) e um compilador Just-In-Time (**PEP 744**). As alterações da biblioteca contêm a remoção de APIs e módulos descontinuados, bem como as melhorias usuais na facilidade de uso e correção.

Melhorias no interpretador:

- Um *interpretador interativo* bastante aprimorado e *mensagens de erro aprimoradas*.
- 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`.
- **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.
- **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.

Novos recursos de tipagem:

- **PEP 696:** Parâmetros de tipo (`typing.TypeVar`, `typing.ParamSpec` e `typing.TypeVarTuple`) agora oferecem suporte a padrões.
- **PEP 702:** Suporte para marcação de descontinuações no sistema de tipos usando o novo decorador `warnings.deprecated()`.
- **PEP 742:** `typing.TypeIs` foi adicionado, fornecendo um comportamento de restrição de tipo mais intuitivo.
- **PEP 705:** `typing.ReadOnly` foi adicionado, para marcar um item de `typing.TypedDict` como somente leitura para verificadores de tipo.

Threads livres:

- **PEP 703**: CPython 3.13 tem suporte experimental para ser executado com a trava global do interpretador, ou GIL, desabilitada quando compilado com `--disable-gil`. Veja [CPython com threads livres](#) para mais detalhes.

Suporte a plataforma:

- **PEP 730**: O iOS da Apple agora é uma plataforma com suporte oficial. O suporte oficial do Android (**PEP 738**) também está em andamento.

Módulos removidos:

- **PEP 594**: As 19 “baterias descarregadas” restantes foram removidas da biblioteca padrão: `aifc`, `audioop`, `cgi`, `cgitb`, `chunk`, `crypt`, `imghdr`, `mailcap`, `msilib`, `nis`, `ntplib`, `ossaudiodev`, `pipes`, `sndhdr`, `spwd`, `sunau`, `telnetlib`, `uu` e `xdrlib`.
- Também foram removidos os módulos `tkinter.tix` e `lib2to3`, e o programa `2to3`.

Mudanças no cronograma de lançamento:

- **PEP 602** (“Ciclo de lançamento anual para Python”) foi atualizado:
 - Python 3.9 até 3.12 tem um ano e meio de suporte total, seguido de 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 interpretador interativo melhor

Em sistemas do tipo Unix, como Linux ou macOS, vem como Windows, Python agora usa um novo console interativo. Quando o usuário inicia o REPL, a partir de um terminal interativo, o console interativo agora oferece suporte aos seguintes novos recursos:

- Prompts coloridos.
- Edição multilinha com preservação do histórico.
- 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).
- A capacidade de emitir comandos específicos do REPL como `help`, `exit` e `quit` sem a necessidade de usar parênteses de chamada após o nome do comando.

Se o novo console interativo não for desejado, ele pode ser desabilitado através da variável de ambiente `PYTHON_BASIC_REPL`.

O novo console requer `curses` em sistemas do tipo Unix.

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 colore mensagens de erro ao exibir rastreamentos por padrão. Este recurso 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 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/random.py", line 1, in <module>
    import random; print(random.randint(5))
    ^^^^^^^^^^^^^
  File "/home/random.py", line 1, in <module>
    import random; print(random.randint(5))
    ^^^^^^^^^^^^^
AttributeError: module 'random' has no attribute 'randint' (consider renaming
→ '/home/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/numpy.py", line 1, in <module>
    import numpy as np; np.array([1,2,3])
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/numpy.py", line 1, in <module>
    import numpy as np; np.array([1,2,3])
    ^^^^^^^^^
AttributeError: module 'numpy' has no attribute 'array' (consider renaming '/'
→ home/numpy.py' if it has the same name as a third-party module you intended
→ to import)
```

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

- Quando um argumento nomeado incorreto é passado para uma função, a mensagem de erro agora sugere potencialmente o argumento nomeado correto. (Contribuição de Pablo Galindo Salgado e Shantanu Jain em [gh-107944](#).)

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

- As classes possuem um novo atributo `__static_attributes__`, preenchido pelo compilador, com uma tupla de nomes de atributos desta classe que são acessados através de `self.X` a partir de qualquer função em seu corpo. (Contribuição de Irit Katriel em [gh-115775](#).)

2.3 Semântica de mutação definida para `locals()`

Historicamente, o resultado esperado da mutação do valor de retorno de `locals()` foi deixado para implementação individual do Python definir.

Através de [PEP 667](#), Python 3.13 padroniza o comportamento histórico do CPython para a maioria dos escopos de execução de código, mas altera escopos otimizados (funções, geradores, corrotinas, compreensões e expressões geradoras) para retornar explicitamente instantâneos independentes das variáveis locais atualmente atribuídas, incluindo variáveis não locais referenciadas localmente capturadas em encerramentos.

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

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

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

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

2.4 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.

2.5 Suporte para plataformas móveis

- iOS agora é uma plataforma suportada por [PEP 11](#). `arm64-apple-ios` (dispositivos iPhone e iPad lançados após 2013) e `arm64-apple-ios-simulator` (simulador Xcode iOS rodando em hardware Apple Silicon) são agora plataformas de nível 3.

`x86_64-apple-ios-simulator` (simulador Xcode iOS rodando em hardware x86_64 mais antigo) não é uma plataforma suportada de nível 3, mas será suportada na base do melhor esforço.

Veja [PEP 730](#): para mais detalhes.

(Escrita e implementação da PEP foi uma contribuição de Russell Keith-Magee em [gh-114099](#).)

3 Compilador JIT experimental

Quando o CPython é configurado usando a opção `--enable-experimental-jit`, um compilador just-in-time é adicionado o que pode acelerar alguns programas Python.

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 um novo *Tier 2 IR* puramente interno, também conhecido como micro-ops (“uops”).
- O IR Tier 2 usa a mesma VM 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.

O sinalizador `--enable-experimental-jit` tem os seguintes valores opcionais:

- `no` (padrão) – Desativa todo o pipeline de Tier 2 e JIT.
- `yes` (padrão se o sinalizador estiver presente sem valor opcional) – 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`.

(No Windows, use `PCbuild/build.bat --experimental-jit` para habilitar o JIT ou `--experimental-jit=interpreter` para habilitar o interpretador Tier 2.)

Veja [PEP 744](#) para mais detalhes.

(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.)

4 CPython com threads livres

O CPython será executado com a trava global do interpretador (GIL) desabilitada quando configurado usando a opção `--disable-gil` no momento da construção. Este é um recurso experimental e, portanto, não é usado por padrão. Os usuários precisam construir seu próprio interpretador ou instalar uma das construções experimentais marcadas como *free-threaded* (threads livres). Veja [PEP 703](#) “Tornando a trava global do interpretador opcional no CPython” para mais detalhes.

A execução com threads livres permite a utilização total do poder de processamento disponível, executando threads em paralelo nos núcleos de CPU disponíveis. Embora nem todos os softwares se beneficiem disso automaticamente, os programas projetados com uso de threads em mente serão executados mais rapidamente em hardware com vários núcleos.

O trabalho ainda está em andamento: espere alguns bugs e um impacto substancial no desempenho de thread único.

A construção com threads livres ainda opcionalmente oferece suporte à execução com a GIL habilitada em tempo de execução usando a variável de ambiente `PYTHON_GIL` ou a opção de linha de comando `-X gil`.

Para verificar se o interpretador atual está configurado com `--disable-gil`, use `sysconfig.get_config_var("Py_GIL_DISABLED")`. Para verificar se a GIL está realmente desabilitada no processo em execução, a função `sys._is_gil_enabled()` pode ser usada.

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 24.1b1 ou mais recente é necessário para instalar pacotes com extensões C na construção com threads livres.

5 Outras mudanças na linguagem

- The `exec()` and `eval()` built-ins now accept their `globals` and `locals` namespace arguments as keywords. (Contributed by Raphael Gaschignard in [gh-105879](#))
- Permite que o argumento `count` de `str.replace()` seja um argumento nomeado. (Contribuição de Hugo van Kemenade em [gh-106487](#).)
- O compilador agora remove os recuos das docstrings. Isso reduzirá o tamanho do cache de bytecode (por exemplo, arquivo `.pyc`). Por exemplo, o tamanho do arquivo de cache para `sqlalchemy.orm.session` no SQLAlchemy 2.0 é reduzido em cerca de 5%. Esta mudança afetará ferramentas que usam docstrings, como `doctest`. (Contribuição de Inada Naoki em [gh-81283](#).)
- A função embutida `compile()` agora pode aceitar um novo sinalizador, `ast.PyCF_OPTIMIZED_AST`, que é semelhante a `ast.PyCF_ONLY_AST` exceto que o AST retornado é otimizado de acordo com o valor do argumento `optimize`. (Contribuição de Irit Katriel em [gh-108113](#).)
- `multiprocessing`, `concurrent.futures`, `compileall`: Substitui `os.cpu_count()` por `os.process_cpu_count()` para selecionar o número padrão de threads de trabalho e processos. Obtém a afinidade da CPU, se houver suporte. (Contribuição de Victor Stinner em [gh-109649](#).)
- `os.path.realpath()` agora resolve nomes de arquivos no estilo MS-DOS, mesmo que o arquivo não esteja acessível. (Contribuição de Moonsik Park em [gh-82367](#).)
- Corrigido um bug onde uma declaração `global` em um bloco `except` é rejeitada quando o `global` é usado no bloco `else`. (Contribuição de Irit Katriel em [gh-111123](#).)
- 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](#).)
- Adicionada uma nova variável de ambiente `PYTHON_FROZEN_MODULES`. Ela determina se os módulos congelados são ou não ignorados pelo mecanismo 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 nova variável de ambiente `PYTHON_HISTORY` pode ser usada para alterar a localização de um arquivo `.python_history`. (Contribuição de Levi Sabah, Zackery Spytz e Hugo van Kemenade em [gh-73965](#).)
- Adiciona a exceção `PythonFinalizationError`. Esta exceção derivada de `RuntimeError` é levantada quando uma operação é bloqueada durante a finalização do Python.

As seguintes funções agora levantam `PythonFinalizationError`, em vez de `RuntimeError`:

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

`- os.forkpty()`.

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

- 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](#).)
- 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](#).)

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

Nota: `ssl.VERIFY_X509_STRICT` pode rejeitar certificados pré-[RFC 5280](#) ou malformados que a implementação OpenSSL subjacente aceitaria de outra forma. Embora não seja recomendado desativar isso, você pode fazer isso usando:

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

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

- `configparser.ConfigParser` agora aceita seções sem nome antes das nomeadas se configurado para isso. (Contribuição de Pedro Sousa Lacerda em [gh-66449](#).)
- Escopo 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. (Contribuição de Jelle Zijlstra em [gh-109118](#) e [gh-118160](#).)
- As classes possuem um novo atributo `__firstlineno__`, preenchido pelo compilador, com o número da primeira linha da definição de classe. (Contribuição de Serhiy Storchaka em [gh-118465](#).)
- As instruções `from __future__ import ...` agora são apenas importações relativas normais se houver pontos antes do nome do módulo. (Contribuição de Jeremiah Gabriel Pascual em [gh-118216](#).)

6 Novos módulos

- Nenhum.

7 Módulos melhorados

7.1 argparse

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

7.2 array

- Adiciona o código do tipo `'w'` (`Py_UCS4`) que pode ser usado para strings Unicode. Ele pode ser usado no lugar do código do tipo `'u'`, que foi descontinuado. (Contribuição de Inada Naoki em [gh-80480](#).)
- Adiciona o método `clear()` para implementar `MutableSequence`. (Contribuição de Mike Zimin em [gh-114894](#).)

7.3 ast

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

Se um campo opcional em um nó AST não for incluído como argumento ao construir uma instância, o campo agora será definido como `None`. Da mesma forma, se um campo de lista for omitido, esse campo será agora definido como uma lista vazia, e se um campo `ast.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.)

Se outros argumentos forem omitidos, um `DeprecationWarning` será emitido. Isso causará uma exceção no Python 3.15. Da mesma forma, passar um argumento nomeado 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 atributo `ast.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 o `compile()` embutido. Isto torna possível obter um AST otimizado. (Contribuição de Irit Katriel em [gh-108113](#).)

7.4 asyncio

- `asyncio.loop.create_unix_server()` agora removerá automaticamente o soquete Unix quando o servidor for fechado. (Contribuição de Pierre Ossman em [gh-111246](#).)
- `asyncio.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 os métodos `asyncio.Server.close_clients()` e `asyncio.Server.abort_clients()` que permitem fechar de forma mais forçada um servidor `asyncio`. (Contribuição de Pierre Ossman em [gh-113538](#).)
- `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](#).)
- Quando `asyncio.TaskGroup.create_task()` é chamado em um `asyncio.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](#).)

- Comportamento aprimorado de `asyncio.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 `asyncio.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 (`asyncio.Task.cancelling()`).

Para lidar com alguns casos extremos, `asyncio.Task.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](#).)

- Adiciona `asyncio.Queue.shutdown()` (junto com `asyncio.QueueShutDown`) para encerramento da fila. (Contribuição de Laurie Opperman e Yves Duprat em [gh-104228](#).)
- Aceita uma tupla de separadores em `asyncio.StreamReader.readuntil()`, parando quando um deles for encontrado. (Contribuição de Bruce Merry em [gh-81322](#).)

7.5 base64

- Adiciona funções `base64.z85encode()` e `base64.z85decode()` que permitem codificar e decodificar dados Z85. Consulte a [especificação Z85](#) para obter mais informações. (Contribuição de Matan Perelman em [gh-75299](#).)

7.6 copy

- Adiciona a função `copy.replace()` que permite criar uma cópia modificada de um objeto, o que é especialmente útil para objetos imutáveis. Ele oferece suporte a tuplas nomeadas criadas com a função de fábrica `collections.namedtuple()`, instâncias `dataclass`, vários objetos `datetime`, objetos `Signature`, `Parameter`, objeto código, e quaisquer classes de usuário que definam o método `__replace__()`. (Contribuição de Serhiy Storchaka em [gh-108751](#).)

7.7 dbm

- Adiciona os métodos `dbm.gnu.gdbm.clear()` e `dbm.ndbm.ndbm.clear()` que removem todos os itens do banco de dados. (Contribuição de Donghee Na em [gh-107122](#).)
- Adiciona o novo backend `dbm.sqlite3` e torna-o o backend padrão `dbm`. (Contribuição de Raymond Hettinger e Erlend E. Aasland em [gh-100414](#).)

7.8 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 parâmetro `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](#).)

7.9 doctest

- A cor é adicionada à saída 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 `doctest.DocTestRunner.run()` agora conta o número de testes ignorados. Adiciona os atributos `doctest.DocTestRunner.skipped` e `doctest.TestResults.skipped`. (Contribuição de Victor Stinner em [gh-108794](#).)

7.10 email

- `email.utils.getaddresses()` e `email.utils.parseaddr()` agora retornam tuplas de dois elementos `(' ', '')` em mais situações onde endereços de e-mail inválidos são encontrados em vez de potencialmente imprecisos valores. Adiciona o parâmetro *strict* opcional a estas duas funções: use `strict=False` para obter o comportamento antigo, aceitar entradas malformadas. `getaddr(email.utils, 'supports_strict_parsing', False)` pode ser usado para verificar se o parâmetro *strict* está disponível. (Contribuição de Thomas Dwyer e Victor Stinner para [gh-102988](#) para melhorar a correção [CVE-2023-27043](#).)

7.11 fractions

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

7.12 gc

- O coletor de lixo cíclico agora é incremental, o que altera o significado dos resultados de `gc.get_threshold()` e `gc.set_threshold()`, bem como de `gc.get_count()` e `gc.get_stats()`.
 - `gc.get_threshold()` retorna uma tupla de três itens para compatibilidade com versões anteriores. 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.
 - `gc.set_threshold()` ignora quaisquer itens após o segundo.
 - `gc.get_count()` e `gc.get_stats()` retornam o mesmo formato de resultados de antes. 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.

7.13 glob

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

7.14 importlib

- As funções `importlib.resources` anteriormente descontinuado não estão mais descontinuadas:

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

Todos agora permitem um diretório (ou árvore) de recursos, usando múltiplos argumentos posicionais.

Para funções de leitura de texto, *encoding* e *errors* devem agora ser fornecidos como argumentos nomeados.

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](#).)

7.15 io

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

7.16 ipaddress

- Adiciona a propriedade `ipaddress.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`.

7.17 itertools

- Adicionada uma opção `strict` a `itertools.batched()`. Isso levanta `ValueError` se o lote final for menor que o tamanho do lote especificado. (Contribuição de Raymond Hettinger em [gh-113202](#).)

7.18 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](#).)

7.19 math

- Uma nova função `fma()` para operações fundidas de multiplicação e adição foi adicionada. Esta função 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](#).)

7.20 mimetypes

- Adiciona a função `guess_file_type()` que funciona com o caminho do arquivo. Passar o caminho do arquivo em vez da URL em `guess_type()` está suavemente descontinuado. (Contribuição de Serhiy Storchaka em [gh-66543](#).)

7.21 mmap

- A classe `mmap.mmap` agora tem um 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](#).)
- `mmap.mmap` agora tem um parâmetro `trackfd` no Unix; se for `False`, o descritor de arquivo especificado por `fileno` não será duplicado. (Contribuição de Zackery Spytz e Petr Viktorin em [gh-78502](#).)
- `mmap.mmap` agora está protegido 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](#).)

7.22 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](#).)
- Removidos `opcode.is_pseudo`, `opcode.MIN_PSEUDO_OPCODE` e `opcode.MAX_PSEUDO_OPCODE`, que foram adicionados em 3.12, nunca foram documentados ou expostos através de `dis`, e não foram planejados para serem usados externamente.

7.23 os

- Adiciona a função `os.process_cpu_count()` para obter o número de CPUs lógicas utilizáveis pelo thread de chamada do processo atual. (Contribuição de Victor Stinner em [gh-109649](#).)
- Adiciona uma interface de baixo nível para descritores de arquivo de notificação de temporizador do Linux via `os.timerfd_create()`, `os.timerfd_settime()`, `os.timerfd_gettime_ns()`, `os.timerfd_gettime()` e `os.timerfd_gettime_ns()`, `os.TFD_NONBLOCK`, `os.TFD_CLOEXEC`, `os.TFD_TIMER_ABSTIME` e `os.TFD_TIMER_CANCEL_ON_SET` (Contribuição de Masaru Tsuchiyama em [gh-108277](#).)
- `os.cpu_count()` e `os.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 o contêiner (código da aplicação). (Contribuição de Donghee Na em [gh-109595](#).)

- Adiciona suporte para `os.lchmod()` e o argumento *follow_symlinks* em `os.chmod()` no Windows. Observe que o valor padrão de *follow_symlinks* em `os.lchmod()` é `False` no Windows. (Contribuição de Serhiy Storchaka em [gh-59616](#).)
- Adiciona suporte para `os.fchmod()` e um descritor de arquivo em `os.chmod()` no Windows. (Contribuição de Serhiy Storchaka em [gh-113191](#).)
- `os.posix_spawn()` agora aceita `env=None`, o que faz com que o processo recém gerado use o ambiente de processo atual. (Contribuição de Jakub Kulik em [gh-113119](#).)
- `os.posix_spawn()` ganha um atributo `os.POSIX_SPAWN_CLOSEFROM` para uso em `file_actions=` em plataformas que oferecem suporte a `posix_spawn_file_actions_addclosefrom_np()`. (Contribuição de Jakub Kulik em [gh-113117](#).)
- `os.mkdir()` e `os.makedirs()` no Windows 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](#).)

7.24 os.path

- Adiciona `os.path.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, `os.path.isabs()` não considera mais caminhos que começam com exatamente uma (contra)barra como absolutos. (Contribuição de Barney Gale e Jon Foster em [gh-44626](#).)
- Adiciona suporte para argumentos nomeados *dir_fd* e *follow_symlinks* em `shutil.chown()`. (Contribuição de Berker Peksag e Tahia K em [gh-62308](#).)

7.25 pathlib

- Adiciona `pathlib.UnsupportedOperation`, que é levantado em vez de `NotImplementedError` quando uma operação de caminho não é suportada. (Contribuição de Barney Gale em [gh-89812](#).)
- Adiciona `pathlib.Path.from_uri()`, um novo construtor para criar um objeto `pathlib.Path` a partir de um URI 'file' (`file://`). (Contribuição de Barney Gale em [gh-107465](#).)
- Adiciona `pathlib.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 `pathlib.PurePath.parser` que armazena a implementação de `os.path` usado para análise e junção de caminhos de baixo nível: `posixpath` ou `ntpath`.
- Adiciona o argumento somente-nomeado *recurse_symlinks* a `pathlib.Path.glob()` e `rglob()`. (Contribuição de Barney Gale em [gh-77609](#).)
- Adiciona o argumento somente-nomeado *follow_symlinks* a `is_file()`, `is_dir()`, `owner()`, `group()`. (Contribuição de Barney Gale em [gh-105793](#) e Kamil Turek em [gh-107962](#).)
- Retorna arquivos e diretórios de `pathlib.Path.glob()` e `rglob()` quando recebe um padrão que termina com `"**"`. Nas versões anteriores, apenas os diretórios eram retornados. (Contribuição de Barney Gale em [gh-70303](#).)

7.26 pdb

- 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/instruções cujo prefixo é um comando pdb agora são identificadas e executadas corretamente. (Contribuição de Tian Gao em [gh-108464](#).)
- `sys.path[0]` não será mais substituído pelo diretório do script que está sendo depurado quando `sys.flags.safe_path` estiver definido (através da opção de linha de comando `-P` ou a variável de ambiente `PYTHONSAFEPATH`). (Contribuição de Tian Gao e Christian Walther em [gh-111762](#).)
- `zipapp` é suportado como alvo de depuração. (Contribuição de Tian Gao em [gh-118501](#).)
- `breakpoint()` e `pdb.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](#).)

7.27 queue

- Adiciona `queue.Queue.shutdown()` (junto com `queue.ShutDown`) para encerramento da fila. (Contribuição de Laurie Opperman e Yves Duprat em [gh-104750](#).)

7.28 random

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

7.29 re

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

7.30 site

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

7.31 sqlite3

- Uma `ResourceWarning` agora é emitida se um objeto `sqlite3.Connection` não for explicitamente fechado. (Contribuição de Erlend E. Aasland em [gh-105539](#).)
- Adiciona o parâmetro somente-nomeado `filter` a `sqlite3.Connection.iterdump()` para filtrar objetos de banco de dados para despejo. (Contribuição de Mariusz Felisiak em [gh-91602](#).)

7.32 statistics

- Adiciona `statistics.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. Também foi adicionado `statistics.kde_random()` para amostragem da função de densidade de probabilidade estimada. (Contribuição de Raymond Hettinger em [gh-115863](#).)

7.33 subprocess

- O módulo `subprocess` agora usa a função `os.posix_spawn()` em mais situações. Notavelmente no caso padrão de `close_fds=True` em versões mais recentes de plataformas incluindo Linux, FreeBSD e Solaris onde a biblioteca C fornece `posix_spawn_file_actions_addclosefrom_np()`. No Linux, isso deve funcionar de maneira semelhante ao nosso código existente baseado em Linux `vfork()`. Um botão de controle privado `subprocess._USE_POSIX_SPAWN` pode ser definido como `False` se você precisar forçar `subprocess` a nunca usar `os.posix_spawn()`. Informe o motivo e os detalhes da plataforma no rastreador de problemas do CPython se você definir isso para que possamos melhorar nossa lógica de seleção de API para todos. (Contribuição de Jakub Kulik em [gh-113117](#).)

7.34 sys

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

7.35 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](#).)

7.36 time

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

7.37 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)`). Adiciona novo parâmetro opcional somente-nomeado `return_python_dict`: chamar `w.wm_attributes(return_python_dict=True)` retorna os atributos como um dict em vez de uma tupla. (Contribuição de Serhiy Storchaka em [gh-43457](#).)
- Adiciona um novo parâmetro opcional somente-nomeado `return_ints` no método `Text.count()`. Passar `return_ints=True` faz com que ele sempre retorne a contagem única como um número inteiro em vez de uma tupla de 1 elemento ou `None`. (Contribuição de Serhiy Storchaka em [gh-97928](#).)

- Adiciona 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 o método `copy_replace()` à classe `PhotoImage` para copiar uma região de uma imagem para outra imagem, possivelmente com ampliação e subamostragem de pixels. 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()` de `PhotoImage`. (Contribuição de Serhiy Storchaka em [gh-118271](#).)

7.38 traceback

- Adiciona o parâmetro `show_group` a `traceback.TracebackException.format_exception_only()` para formatar as exceções aninhadas de uma instância `BaseExceptionGroup`, recursivamente. (Contribuição de Irit Katriel em [gh-105292](#).)
- Adiciona o campo `exc_type_str` a `TracebackException`, que contém uma exibição de string do `exc_type`. Substitua o campo `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](#).)

7.39 types

- O construtor `SimpleNamespace` agora permite especificar valores iniciais de atributos como um argumento posicional que deve ser um mapeamento ou um iterável de pares chave-valor. (Contribuição de Serhiy Storchaka em [gh-108191](#).)

7.40 typing

- Adiciona `typing.get_protocol_members()` para retornar o conjunto de membros que definem um `typing.Protocol`. Adiciona `typing.is_protocol()` para verificar se uma classe é um `typing.Protocol`. (Contribuição de Jelle Zijlstra em [gh-104873](#).)
- Adiciona `typing.ReadOnly`, uma construção especial do `typing` para marcar um item de `typing.TypedDict` como somente leitura para verificadores de tipo. Veja [PEP 705](#) para mais detalhes.
- Adiciona `typing.NoDefault`, um objeto sinalizador usado para representar os padrões de alguns parâmetros no módulo `typing`. (Contribuição de Jelle Zijlstra em [gh-116126](#).)

7.41 unicodedata

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

7.42 venv

- Adiciona suporte para adicionar arquivos ignorados pelo gerenciamento de controle de fontes (SCM) ao 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 (`venv.EnvBuilder` e `venv.create()`) e desativado por padrão por meio da CLI (usando `--without-scm-ignore-files`). (Contribuição de Brett Cannon em [gh-108125](#).)

7.43 warnings

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

7.44 xml.etree.ElementTree

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

7.45 zipimport

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

8 Otimizações

- `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` usa `os.posix_spawn()` em mais situações, incluindo o padrão sendo `close_fds=True` em muitas plataformas modernas. 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](#).)
- 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 `importlib.metadata`, `threading`, `enum`, `functools` e `email.utils`. (Contribuição de Alex Waygood, Shantanu Jain, Adam Turner, Daniel Hollas e outros em [gh-109653](#).)

9 Módulos e APIs removidas

9.1 PEP 594: baterias mortas (e remoção de outros módulos)

- [PEP 594](#) removeu 19 módulos da biblioteca padrão, descontinuados no Python 3.11:
 - `aifc`. (Contribuição de Victor Stinner em [gh-104773](#).)
 - `audioop`. (Contribuição de Victor Stinner em [gh-104773](#).)
 - `chunk`. (Contribuição de Victor Stinner em [gh-104773](#).)
 - `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 projeto `multipart` do PyPI para POST e PUT.
- * `cgi.parse()` pode ser substituída chamando `urllib.parse.parse_qs()` diretamente na string de consulta desejada, exceto para a entrada `multipart/form-data`, que pode ser tratada conforme descrito 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 substituída pela funcionalidade do pacote `email` (por exemplo `email.message.EmailMessage` e `email.message.Message`) que implementa os mesmos RFCs MIME, ou com o projeto `multipart` do PyPI.

(Contribuição de Victor Stinner em [gh-104773](#).)

- Módulo `crypt` e sua extensão privada `_crypt`. O módulo `hashlib` é um substituto potencial para certos casos de uso. Caso contrário, os seguintes projetos PyPI podem ser usados:
 - * `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.

(Contribuição de Victor Stinner em [gh-104773](#).)

- `imghdr`: use os projetos `filetype`, `puremagic` ou `python-magic`. A função `puremagic.what()` pode ser usada para substituir a função `imghdr.what()` para todos os formatos de arquivo que eram suportados por `imghdr`. (Contribuição de Victor Stinner em [gh-104773](#).)
- `mailcap`. O módulo `mimetypes` fornece uma alternativa. (Contribuição de Victor Stinner em [gh-104773](#).)
- `msilib`. (Contribuição de Zachary Ware em [gh-104773](#).)
- `nis`. (Contribuição de Victor Stinner em [gh-104773](#).)
- `nntplib`: o projeto `nntplib` do PyPI pode ser usado em seu lugar. (Contribuição de Victor Stinner em [gh-104773](#).)
- `ossaudiodev`: use o projeto `pygame` para reprodução de áudio. (Contribuição de Victor Stinner em [gh-104780](#).)
- `pipes`: use o módulo `subprocess`. (Contribuição de Victor Stinner em [gh-104773](#).)
- `sndhdr`: use os projetos `filetype`, `puremagic` ou `python-magic`. (Contribuição de Victor Stinner em [gh-104773](#).)
- `spwd`: o projeto `python-pam` pode ser usado em seu lugar. (Contribuição de Victor Stinner em [gh-104773](#).)
- `sunau`. (Contribuição de Victor Stinner em [gh-104773](#).)
- `telnetlib`, use os projetos `telnetlib3` ou `Exscript`. (Contribuição de Victor Stinner em [gh-104773](#).)
- `uu`: o módulo `base64` é uma alternativa moderna. (Contribuição de Victor Stinner em [gh-104773](#).)

- `xdrllib`. (Contribuição de Victor Stinner em [gh-104773](#).)
- Remove o programa `2to3` e o módulo `lib2to3`, descontinuados no Python 3.11. (Contribuição de Victor Stinner em [gh-104780](#).)
- 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](#).)

9.2 configparser

- Remove a classe não documentada `configparser.LegacyInterpolation`, descontinuada na docs-tring desde Python 3.2 e com um aviso de descontinuação desde Python 3.11. (Contribuição de Hugo van Kemenade em [gh-104886](#).)

9.3 importlib

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

9.4 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](#).)

9.5 logging

- `logging`: Remove os métodos `Logger.warn()` e `LoggerAdapter.warn()` não documentados e não testados e a função `logging.warn()`. Descontinuados desde Python 3.3, eles eram apelidos para o método `logging.Logger.warning()`, método `logging.LoggerAdapter.warning()` e função `logging.warning()`. (Contribuição de Victor Stinner em [gh-105376](#).)

9.6 pathlib

- Remove o suporte para usar objetos `pathlib.Path` como gerenciadores de contexto. Esta funcionalidade foi descontinuada e tornou-se não operacional no Python 3.9.

9.7 re

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

9.8 turtle

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

9.9 typing

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

9.10 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`:

- `unittest.TestLoader.loadTestsFromModule()`
- `unittest.TestLoader.loadTestsFromTestCase()`
- `unittest.TestLoader.getTestCaseNames()`

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

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

9.11 urllib

- Remove os parâmetros `cafile`, `capath` e `cadefault` da função `urllib.request.urlopen()`, descontinuada no Python 3.6: passe o parâmetro `context` em vez disso. Use `ssl.SSLContext.load_cert_chain()` para carregar certificados específicos ou deixe `ssl.create_default_context()` selecionar os certificados CA confiáveis do sistema para você. (Contribuição de Victor Stinner em [gh-105382](#).)

9.12 webbrowser

- Remove a classe `webbrowser.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 `webbrowser.MacOSXOSAScript._name`. Use o atributo `webbrowser.MacOSXOSAScript.name` em vez disso. (Contribuição de Nikita Sobolev em [gh-105546](#).)

10 Novas descontinuações

- Removidos os descritores `classmethod` encadeados (introduzidos em [gh-63272](#)). Isto não pode mais ser usado para agrupar outros descritores como `property`. O design central desse recurso apresentava falhas e causou vários problemas posteriores. Para “passar” um `classmethod`, considere usar o atributo `__wrapped__` que foi adicionado no Python 3.10. (Contribuição de Raymond Hettinger em [gh-89519](#).)
- `array`: O código de formato `'u'` de `array`, descontinuado em documentos desde Python 3.3, emite `DeprecationWarning` desde 3.13 e será removido em Python 3.16. Use o código de formato `'w'`. (Contribuição de Hugo van Kemenade em [gh-80480](#).)
- `ctypes`: Descontinua funções não documentadas `ctypes.SetPointerType()` e `ctypes.ARRAY()`. Substitua `ctypes.ARRAY(item_type, size)` por `item_type * size`. (Contribuição de Victor Stinner em [gh-105733](#).)
- `decimal`: Descontinua o especificador de formato não padrão `"N"` para `decimal.Decimal`. Não foi documentado e apenas suportado na implementação C. (Contribuição de Serhiy Storchaka em [gh-89902](#).)
- `dis`: O separador `dis.HAVE_ARGUMENT` está descontinuado. Verifique a associação em `hasarg`. (Contribuição de Irit Katriel em [gh-109319](#).)
- `frame-objects`: Chamar `frame.clear()` em um quadro suspenso levanta `RuntimeError` (como sempre foi o caso para um quadro em execução). (Contribuição de Irit Katriel em [gh-79932](#).)
- Módulos `getopt` e `optparse`: Eles agora estão suavemente descontinuado: o módulo `argparse` deve ser usado para novos projetos. Anteriormente, o módulo `optparse` já estava descontinuado, sua remoção não foi agendada e nenhum aviso foi emitido: portanto não há mudança na prática. (Contribuição de Victor Stinner em [gh-106535](#).)
- `gettext`: Emite aviso de descontinuação para números não inteiros em funções e métodos `gettext` que consideram formas plurais mesmo que a tradução não tenha sido encontrada. (Contribuição de Serhiy Storchaka em [gh-88434](#).)
- `glob`: As funções não documentadas `glob.glob0()` e `glob.glob1()` estão descontinuadas. Use `glob.glob()` e passe um diretório para seu argumento `root_dir`. (Contribuição de Barney Gale em [gh-117337](#).)
- `http.server`: `http.server.CGIHTTPRequestHandler` agora emite uma exceção `DeprecationWarning`, pois será removida na versão 3.15. Os servidores HTTP CGI baseados em processos estão em desuso há muito tempo. Este código estava desatualizado, sem manutenção e raramente usado. Tem um alto potencial para bugs de segurança e funcionalidade. Isso inclui a remoção do sinalizador `--cgi` da linha de comando `python -m http.server` na versão 3.15.
- `mimetypes`: Passar o caminho do arquivo em vez da URL em `guess_type()` está suavemente descontinuado. Use `guess_file_type()` em vez disso. (Contribuição de Serhiy Storchaka em [gh-66543](#).)
- `re`: Passar argumentos opcionais `maxsplit`, `count` e `flags` em funções de nível de módulo `re.split()`, `re.sub()` e `re.subn()` como argumentos posicionais agora é descontinuado. Nas versões futuras do Python, esses parâmetros serão somente-nomeados. (Contribuição de Serhiy Storchaka em [gh-56166](#).)
- `pathlib`: `pathlib.PurePath.is_reserved()` está descontinuado e agendado para remoção no Python 3.15. Use `os.path.isreserved()` para detectar caminhos reservados no Windows.
- `platform`: `java_ver()` está descontinuada e será removida na versão 3.15. Ela não foi testada em grande parte, tinha uma API confusa e só era útil para suporte a Jython. (Contribuição de Nikita Sobolev em [gh-116349](#).)
- `pydoc`: Descontinua a função não documentada `pydoc.ispackage()`. (Contribuição de Zackery Spytz em [gh-64020](#).)
- `sqlite3`: Passar mais de um argumento posicional para `sqlite3.connect()` e o construtor `sqlite3.Connection` está descontinuado. Os parâmetros restantes se tornarão somente-nomeados no Python 3.15.

Descontinua a passagem de nome, número de argumentos e o chamável como argumentos nomeados para as seguintes APIs de `sqlite3.Connection`:

- `create_function()`
- `create_aggregate()`

Descontinua a passagem do chamável de retorno de chamada como argumento nomeado para as seguintes APIs de `sqlite3.Connection`:

- `set_authorizer()`
- `set_progress_handler()`
- `set_trace_callback()`

Os parâmetros afetados se tornarão somente-posicionais no Python 3.15.

(Contribuição de Erlend E. Aasland em [gh-107948](#) e [gh-108278](#).)

- `sys`: função `sys._enablelegacywindowsfsencoding()`. Substitua-a pela variável de ambiente `PYTHONLEGACYWINDOWSFSENCODING`. (Contribuição de Inada Naoki em [gh-73427](#).)
- `tarfile`: O atributo `tarfile` não documentado e não utilizado de `tarfile.TarFile` está descontinuado e agendado para remoção no Python 3.16.
- `traceback`: O campo `exc_type` de `traceback.TracebackException` está descontinuado. Use `exc_type_str` em vez disso.
- `typing`:
 - Criar uma classe `typing.NamedTuple` usando argumentos nomeados para denotar os campos (`NT = NamedTuple("NT", x=int, y=int)`) está descontinuado e não será permitido no Python 3.15. Use a sintaxe baseada em classe ou a sintaxe funcional. (Contribuição de Alex Waygood em [gh-105566](#).)
 - Ao usar a sintaxe funcional para criar uma classe `typing.NamedTuple` ou uma classe `typing.TypedDict`, falhando ao passar um valor para o parâmetro 'fields' (`NT = NamedTuple("NT")` ou `TD = TypedDict("TD")`) está descontinuado. Passar `None` para o parâmetro 'fields' (`NT = NamedTuple("NT", None)` ou `TD = TypedDict("TD", None)`) também está descontinuado. Ambos não serão permitidos no Python 3.15. Para criar uma classe `NamedTuple` com 0 campos, use `class NT(NamedTuple):` ou `NT = NamedTuple("NT", [])`. Para criar uma classe `TypedDict` com 0 campos, use `class TD(TypedDict):` ou `TD = TypedDict("TD", {})`. (Contribuição de Alex Waygood em [gh-105566](#) e [gh-105570](#).)
 - `typing.no_type_check_decorator()` está descontinuado e agendado para remoção no Python 3.15. Depois de oito anos no módulo `typing`, ele ainda não foi suportado por nenhum dos principais verificadores de tipos. (Contribuição de Alex Waygood em [gh-106309](#).)
 - `typing.AnyStr` está descontinuado. No Python 3.16, ele será removido de `typing.__all__`, e uma exceção `DeprecationWarning` será emitida quando for importada ou acessada. Ele será removido totalmente no Python 3.18. Use a nova sintaxe de parâmetro de tipo. (Contribuição de Michael The em [gh-107116](#).)
- `user-defined-funcs`: A atribuição ao atributo `__code__` de uma função onde o tipo do novo objeto código não corresponde ao tipo da função está descontinuada. Os diferentes tipos são: função simples, gerador, gerador assíncrono e corrotina. (Contribuição de Irit Katriel em [gh-81137](#).)
- `wave`: Descontinua os métodos `getmark()`, `setmark()` e `getmarkers()` das classes `wave.Wave_read` e `wave.Wave_write`. Eles serão removidos no Python 3.15. (Contribuição de Victor Stinner em [gh-105096](#).)

10.1 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](#).)

- `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](#).)

10.2 Remoção pendente no Python 3.15

- `http.server.CGIHTTPRequestHandler` será removido junto com seu sinalizador relacionado `--cgi` para `python -m http.server`. Estava descontinuado e raramente usado. Não existe substituição direta. *Qualquer coisa* é melhor que CGI para fazer a interface de um servidor web com um manipulador de solicitações.
- `locale`: `locale.getdefaultlocale()` foi descontinuada no Python 3.11 e originalmente planejada para remoção no Python 3.13 ([gh-90817](#)), mas a remoção foi adiada para o Python 3.15. Use `locale.setlocale()`, `locale.getencoding()` e `locale.getlocale()` em vez disso. (Contribuição de Hugo van Kemenade em [gh-111187](#).)
- `pathlib`: `pathlib.PurePath.is_reserved()` está descontinuado e agendado para remoção no Python 3.15. Use `os.path.isreserved()` para detectar caminhos reservados no Windows.
- `platform`: `java_ver()` está descontinuada e será removida na versão 3.15. Ela não foi testada em grande parte, tinha uma API confusa e só era útil para suporte a Jython. (Contribuição de Nikita Sobolev em [gh-116349](#).)
- `threading`: Passar qualquer argumento para `threading.RLock()` agora está descontinuado. A versão C permite qualquer número de `args` e `kwargs`, mas eles são simplesmente ignorados. A versão Python não permite nenhum argumento. Todos os argumentos serão removidos de `threading.RLock()` no Python 3.15. (Contribuição de Nikita Sobolev em [gh-102029](#).)
- `typing.NamedTuple`:
 - A sintaxe de argumento nomeado não documentada para criar classes `NamedTuple` (`NT = NamedTuple("NT", x=int)`) está descontinuada e não será permitida em 3.15. Use a sintaxe baseada em classe ou a sintaxe funcional.
 - Ao usar a sintaxe funcional para criar uma classe `NamedTuple`, falhar ao passar um valor para o parâmetro *fields* (`NT = NamedTuple("NT")`) está descontinuada. Passar `None` para o parâmetro *fields* (`NT = NamedTuple("NT", None)`) também está descontinuada. Ambos não serão permitidos no Python 3.15. Para criar uma classe `NamedTuple` com 0 campos, use `class NT(NamedTuple):` ou `NT = NamedTuple("NT", [])`.
- `typing.TypedDict`: Ao usar a sintaxe funcional para criar uma classe `TypedDict`, falhar ao passar um valor para o parâmetro *fields* (`TD = TypedDict("TD")`) está descontinuada. Passar `None` para o parâmetro *fields* (`TD = TypedDict("TD", None)`) também está descontinuada. Ambos não serão permitidos no Python 3.15. Para criar uma classe `TypedDict` com 0 campos, use `class TD(TypedDict):` ou `TD = TypedDict("TD", {})`.
- `wave`: Descontinua os métodos `getmark()`, `setmark()` e `getmarkers()` das classes `wave`, `Wave_read` e `Wave_write`. Eles serão removidos no Python 3.15. (Contribuição de Victor Stinner em [gh-105096](#).)

10.3 Remoção pendente no Python 3.16

- `array.array`: tipo `'u'` (`wchar_t`): use o tipo `'w'` (`Py_UCS4`).

10.4 Remoção pendente em versões futuras

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

- `argparse`: O aninhamento de grupos de argumentos e o aninhamento de grupos mutuamente exclusivos estão descontinuados.
- `builtins`:
 - `~bool`, inversão bit a bit em booleanos.
 - `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__()`.
- `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_notab`: 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 `EntryPoints`.
 - `None` implícito nos valores de retorno.
- `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()`.

11 Mudanças no bytecode do CPython

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

12 Mudanças na API C

12.1 Novas funcionalidades

- 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](#).)
- O parâmetro `keywords` de `PyArg_ParseTupleAndKeywords()` e `PyArg_VaParseTupleAndKeywords()` agora tem tipo `char *const*` em C e `const char *const*` em C++, em vez de `char**`. Torna essas funções compatíveis com argumentos do tipo `const char *const*`, `const char**` ou `char *const*` em C++ e `char *const*` em C sem uma conversão de tipo explícita. Isso pode ser substituído pela macro `PY_CXX_CONST`. (Contribuição de Serhiy Storchaka em [gh-65210](#).)
- Adiciona `PyImport_AddModuleRef()`: semelhante a `PyImport_AddModule()`, mas retorna uma referência forte em vez de uma referência emprestada. (Contribuição de Victor Stinner em [gh-105922](#).)
- Adiciona a função `PyWeakref_GetRef()`: semelhante a `PyWeakref_GetObject()`, mas retorna uma referência forte, ou `NULL` se o referente não mais existir. (Contribuição de Victor Stinner em [gh-105927](#).)
- Adiciona `PyObject_GetOptionalAttr()` e `PyObject_GetOptionalAttrString()`, variantes de `PyObject_GetAttr()` e `PyObject_GetAttrString()` que não levantam `AttributeError` se o atributo não for encontrado. Essas variantes são mais convenientes e rápidas se o atributo ausente não for tratado como uma falha. (Contribuição de Serhiy Storchaka em [gh-106521](#).)
- Adiciona `PyMapping_GetOptionalItem()` e `PyMapping_GetOptionalItemString()`, variantes de `PyObject_GetItem()` e `PyMapping_GetItemString()` que não levantam `KeyError` se a chave não for encontrada. Essas variantes são mais convenientes e rápidas se a chave ausente não for tratada como uma falha. (Contribuição de Serhiy Storchaka em [gh-106307](#).)
- 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()`.

Novas funções retornam não apenas 1 para verdadeiro e 0 para falso, mas também -1 para erro.

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

- 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](#).)
- Adiciona a função `PyModule_Add()`: semelhante a `PyModule_AddObjectRef()` e `PyModule_AddObject()`, mas sempre rouba uma referência ao valor. (Contribuição de Serhiy Storchaka em [gh-86493](#).)
- Adiciona funções `PyDict_GetItemRef()` e `PyDict_GetItemStringRef()`: semelhantes a `PyDict_GetItemWithError()`, mas retornando uma referência forte em vez de uma referência emprestada. Além disso, essas funções retornam -1 em caso de erro e, portanto, a verificação de `PyErr_Occurred()` não é necessária. (Contribuição de Victor Stinner em [gh-106004](#).)
- Adicionada `PyDict_SetDefaultRef()`, que é semelhante a `PyDict_SetDefault()`, mas retorna uma referência forte em vez de uma referência emprestada. Esta função retorna -1 em caso de erro, 0 na inserção e 1 se a chave já estivesse presente no dicionário. (Contribuição de Sam Gross em [gh-112066](#).)
- Adiciona a função `PyDict_ContainsString()` como `PyDict_Contains()`, mas `key` é especificada como uma string de bytes `const char*` codificada em UTF-8, em vez de um `PyObject*`. (Contribuição de Victor Stinner em [gh-108314](#).)
- Adicionada a função `PyList_GetItemRef()`: semelhante a `PyList_GetItem()`, mas retorna uma referência forte em vez de uma referência emprestada.
- Adiciona a função `Py_IsFinalizing()`: verifica se o interpretador Python principal está sendo desligado. (Contribuição de Victor Stinner em [gh-108014](#).)
- Adiciona a função `PyLong_AsInt()`: semelhante a `PyLong_AsLong()`, mas armazena o resultado em um `int C` em vez de um `long C`. Anteriormente, era conhecida como função privada `_PyLong_AsInt()` (com um prefixo de sublinhado). (Contribuição de Victor Stinner em [gh-108014](#).)
- Python construído com `configure --with-trace-refs` (referências de rastreamento) agora oferece suporte à API Limitada. (Contribuição de Victor Stinner em [gh-108634](#).)
- Adiciona funções `PyObject_VisitManagedDict()` e `PyObject_ClearManagedDict()` que devem ser chamadas pelas funções `traverse` e `clear` de um tipo usando o sinalizador `Py_TPFLAGS_MANAGED_DICT`. O projeto [pythoncapi-compat](#) pode ser usado para obter essas funções no Python 3.11 e 3.12. (Contribuição de Victor Stinner em [gh-107073](#).)
- Adiciona as funções `PyUnicode_EqualToUTF8AndSize()` e `PyUnicode_EqualToUTF8()`: comparam o objeto `Unicode` com uma string `const char*` codificada em UTF-8 e retornam `true` (1) se forem iguais, ou `false` (0) caso contrário. Essas funções não levantam exceções. (Contribuição de Serhiy Storchaka em [gh-110289](#).)
- Adiciona a função `PyThreadState_GetUnchecked()`: semelhante a `PyThreadState_Get()`, mas não encerra forçadamente o processo com um erro fatal se for `NULL`. O chamador é responsável por verificar se o resultado é `NULL`. Anteriormente, a função era privada e conhecida como `_PyThreadState_UncheckedGet()`. (Contribuição de Victor Stinner em [gh-108867](#).)
- Adiciona a função `PySys_AuditTuple()`: semelhante a `PySys_Audit()`, mas passa argumentos de evento como um objeto `tuple` do Python. (Contribuição de Victor Stinner em [gh-85283](#).)
- `PyArg_ParseTupleAndKeywords()` agora oferece suporte a nomes de parâmetros nomeados não-ASCII. (Contribuição de Serhiy Storchaka em [gh-110815](#).)

- Adiciona `PyMem_RawMalloc()`, `PyMem_RawCalloc()`, `PyMem_RawRealloc()` e `PyMem_RawFree()` à API C limitada (versão 3.13). (Contribuição de Victor Stinner em [gh-85283](#).)
- Adiciona as funções `PySys_Audit()` e `PySys_AuditTuple()` à API C limitada. (Contribuição de Victor Stinner em [gh-85283](#).)
- Adiciona a função `PyErr_FormatUnraisable()`: semelhante a `PyErr_WriteUnraisable()`, mas permite personalizar a mensagem de aviso. (Contribuição de Serhiy Storchaka em [gh-108082](#).)
- Adiciona as funções `PyList_Extend()` e `PyList_Clear()`: semelhantes aos métodos `list.extend()` e `list.clear()` do Python. (Contribuição de Victor Stinner em [gh-111138](#).)
- Adiciona funções `PyDict_Pop()` e `PyDict_PopString()`: removem uma chave de um dicionário e, opcionalmente, retornam o valor removido. Isto é semelhante a `dict.pop()`, mas sem o valor padrão e sem levantar `KeyError` se a chave estiver ausente. (Contribuição de Stefan Behnel e Victor Stinner em [gh-111262](#).)
- Adiciona a função `Py_HashPointer()` para fazer hash de um ponteiro. (Contribuição de Victor Stinner em [gh-111545](#).)
- Adiciona a função `PyObject_GenericHash()` que implementa a função hashing padrão de um objeto Python. (Contribuição de Serhiy Storchaka em [gh-113024](#).)
- Adiciona a API C `PyTime`:
 - Tipo `PyTime_t`.
 - Constantes `PyTime_MIN` e `PyTime_MAX`.
 - Adiciona as funções:
 - * `PyTime_AsSecondsDouble()`.
 - * `PyTime_Monotonic()`.
 - * `PyTime_MonotonicRaw()`.
 - * `PyTime_PerfCounter()`.
 - * `PyTime_PerfCounterRaw()`.
 - * `PyTime_Time()`.
 - * `PyTime_TimeRaw()`.
 (Contribuição de Victor Stinner e Petr Viktorin em [gh-110850](#).)
- Adiciona as funções `PyLong_AsNativeBytes()`, `PyLong_FromNativeBytes()` e `PyLong_FromUnsignedNativeBytes()` para simplificar a conversão entre tipos inteiros nativos e objetos `int` do Python. (Contribuição de Steve Dower em [gh-111140](#).)
- Adiciona a função `PyType_GetFullyQualifiedName()` para obter o nome totalmente qualificado do tipo. Equivalente a `f"{type.__module__}.{type.__qualname__}"`, ou `type.__qualname__` se `type.__module__` não for uma string ou for igual a `"builtins"`. (Contribuição de Victor Stinner em [gh-111696](#).)
- Adiciona a função `PyType_GetModuleName()` para obter o nome do módulo do tipo. Equivalente a obter o atributo `type.__module__`. (Contribuição de Eric Snow e Victor Stinner em [gh-111696](#).)
- Adiciona suporte para os formatos `%T`, `%#T`, `%N` e `%#N` para `PyUnicode_FromFormat()`: formata o nome totalmente qualificado de um tipo de objeto e de um tipo: chame `PyType_GetModuleName()`. Veja [PEP 737](#) para mais informações. (Contribuição de Victor Stinner em [gh-111696](#).)
- Adiciona as funções `Py_GetConstant()` e `Py_GetConstantBorrowed()` para obter constantes. Por exemplo, `Py_GetConstant(Py_CONSTANT_ZERO)` retorna uma referência forte para a constante zero. (Contribuição de Victor Stinner em [gh-115754](#).)
- Adiciona `PyType_GetModuleByDef()` à API C limitada (contribuição de Victor Stinner em [gh-116936](#).)

- Adiciona duas novas funções à API C, `PyRefTracer_SetTracer()` e `PyRefTracer_GetTracer()`, que permitem rastrear a criação e destruição de objetos da mesma forma que o módulo `tracemalloc`. (Contribuição de Pablo Galindo em [gh-93502](#).)
- Adiciona `PyEval_GetFrameBuiltins()`, `PyEval_GetFrameGlobals()` e `PyEval_GetFrameLocals()` à API C. Essas substituições para `PyEval_GetBuiltins()`, `PyEval_GetGlobals()` e `PyEval_GetLocals()` retornam referências fortes em vez de referências emprestadas. (Adicionado como parte de [PEP 667](#).)
- Adiciona a API `PyMutex`, um mutex leve que ocupa um único byte. A função `PyMutex_Lock()` irá liberar a GIL (se atualmente mantida) se a operação precisar de bloqueio. (Contribuição de Sam Gross em [gh-108724](#).)

13 Mudanças na construção

- A opção `--with-system-libmpdec` do `configure` agora tem como padrão `yes`. A cópia empacotada de `libmpdecimal` será removida no Python 3.15.
- `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](#).)
- Python construído com `configure --with-trace-refs` (referências de rastreamento) tem compatibilidade de ABI com a construção de lançamento do Python e a construção de depuração. (Contribuição de Victor Stinner em [gh-108634](#).)
- Construir CPython agora requer um compilador com suporte para a biblioteca atômica C11, funções atômicas embutidas do GCC ou intrínsecos interligados MSVC.
- As extensões `Cerrno`, `fcntl`, `grp`, `md5`, `pwd`, `resource`, `termios`, `winsound`, `_ctypes_test`, `_multiprocessing.posixshmem`, `_scproxy`, `_stat`, `_statistics`, `_testconsole`, `_testimportmultiple` e `_uuid` são agora construídas com a API C limitada. (Contribuição de Victor Stinner em [gh-85283](#).)
- `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](#).)
- Python agora inclui a biblioteca `mimalloc`. Está licenciada sob a licença do MIT; veja licença do `mimalloc`. O `mimalloc` incluído tem alterações personalizadas, veja [gh-113141](#) para detalhes. (Contribuição de Dino Viehland em [gh-109914](#).)
- 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`.

14 Portando para o Python 3.13

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

14.1 Mudanças na API Python

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

Qualquer biblioteca ou aplicação que forneça um módulo `_thread` personalizado deve obrigatoriamente fornecer `_is_main_interpreter()`, da mesma forma que outros atributos “privados” do módulo. (Veja [gh-112826](#).)

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

14.2 Mudanças na API C

- `Python.h` não inclui mais o cabeçalho padrão `<ieeefp.h>`. Ele foi incluído para a função `finite()` que agora é fornecida pelo cabeçalho `<math.h>`. Agora deve ser incluído explicitamente, se necessário. Remova também a macro `HAVE_IEEEFP_H`. (Contribuição de Victor Stinner em [gh-108765](#).)
- `Python.h` não inclui mais estes arquivos de cabeçalho padrão: `<time.h>`, `<sys/select.h>` e `<sys/time.h>`. Se necessário, deverão agora ser incluídos explicitamente. Por exemplo, `<time.h>` fornece as funções `clock()` e `gmtime()`, `<sys/select.h>` fornece o `select()`, e `<sys/time.h>` fornece as funções `futimes()`, `gettimeofday()` e `setitimer()`. (Contribuição de Victor Stinner em [gh-108765](#).)
- No Windows, `Python.h` não inclui mais o arquivo de cabeçalho padrão `<stddef.h>`. Se necessário, deverá agora ser incluído explicitamente. Por exemplo, ele fornece a função `offsetof()` e os tipos `size_t` e `ptrdiff_t`. Incluindo `<stddef.h>` explicitamente já era necessário para todas as outras plataformas, a macro `HAVE_STDDEF_H` só é definida no Windows. (Contribuição de Victor Stinner em [gh-108765](#).)

- Se a macro `Py_LIMITED_API` estiver definida, as macros `Py_BUILD_CORE`, `Py_BUILD_CORE_BUILTIN` e `Py_BUILD_CORE_MODULE` agora são indefinidas por `<Python.h>`. (Contribuição de Victor Stinner em [gh-85283](#).)
- As macros antigas da lixeira `Py_TRASHCAN_SAFE_BEGIN` e `Py_TRASHCAN_SAFE_END` foram removidas. Elas devem ser substituídas pelas novas macros `Py_TRASHCAN_BEGIN` e `Py_TRASHCAN_END`.

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

```
static void
mytype_dealloc(mytype *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
mytype_dealloc(mytype *p)
{
    PyObject_GC_UnTrack(p);
    Py_TRASHCAN_BEGIN(p, mytype_dealloc)
    ...
    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](#).)

- Funções `PyDict_GetItem()`, `PyDict_GetItemString()`, `PyMapping_HasKey()`, `PyMapping_HasKeyString()`, `PyObject_HasAttr()`, `PyObject_HasAttrString()`, e `PySys_GetObject()`, que apagam todos os erros que ocorrem ao chamá-los, agora se reportam usando `sys.unraisablehook()`. Você pode substituí-las por outras funções conforme recomendado na documentação. (Contribuição de Serhiy Storchaka em [gh-106672](#).)
- `PyCode_GetFirstFree()` é uma API instável agora e foi renomeada para `PyUnstable_Code_GetFirstFree()`. (Contribuição de Bogdan Romanyuk em [gh-115781](#).)
- Os efeitos da mutação do dicionário retornado de `PyEval_GetLocals()` em um escopo otimizado foram alterados. Novas entradas de dict adicionadas desta forma agora *apenas* ficarão visíveis para chamadas `PyEval_GetLocals()` subsequentes nesse quadro, como `PyFrame_GetLocals()`, `locals()` e `FrameType.f_locals` não acessa mais o mesmo dicionário armazenado em cache subjacente. As alterações feitas nas entradas para nomes de variáveis reais e nomes adicionados através das interfaces proxy de “write-through” serão substituídas em chamadas subsequentes para `PyEval_GetLocals()` nesse quadro. A atualização de código recomendada depende de como a função estava sendo usada, portanto, consulte o aviso de descontinuação na função para obter detalhes. (Alteração como parte da [PEP 667](#).)
- Chamar `PyFrame_GetLocals()` em um escopo otimizado agora retorna um proxy de “write-through” em vez de uma captura que é atualizada em horários mal especificados. Se uma captura for desejada, ela deve ser criada explicitamente (por exemplo, com `PyDict_Copy()`) ou chamando a nova API `PyEval_GetFrameLocals()`. (Alteração como parte da [PEP 667](#).)
- `PyFrame_FastToLocals()` e `PyFrame_FastToLocalsWithError()` não têm mais nenhum efeito. Chamar essas funções tem sido redundante desde o Python 3.11, quando `PyFrame_GetLocals()` foi introduzido pela primeira vez. (Alteração como parte da [PEP 667](#).)
- `PyFrame_LocalsToFast()` não tem mais efeito. Chamar esta função é redundante agora que `PyFrame_GetLocals()` retorna um proxy de “write-through” para escopos otimizados. (Alteração como parte da [PEP 667](#).)

14.3 APIs C removidas

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

- Remova funções descontinuadas no Python 3.9:

- `PyEval_CallObject()`, `PyEval_CallObjectWithKeywords()`: use `PyObject_CallNoArgs()` ou `PyObject_Call()` em vez disso. Aviso: os argumentos posicionais de `PyObject_Call()` devem ser tuple e não podem ser NULL, os argumentos nomeados devem ser dict ou NULL, enquanto funções removidas verificavam o tipo de argumentos e aceitavam argumentos posicionais e nomeados NULL. Para substituir `PyEval_CallObjectWithKeywords(func, NULL, kwargs)` por `PyObject_Call()`, passe uma tupla vazia como argumentos posicionais usando `PyTuple_New(0)`.
- `PyEval_CallFunction()`: use `PyObject_CallFunction()`.
- `PyEval_CallMethod()`: use `PyObject_CallMethod()`.
- `PyCFunction_Call()`: use `PyObject_Call()`.

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

- Remova protocolos de buffer antigos descontinuados no Python 3.0. Use `bufferobjects` em vez disso.

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

```
Py_buffer view;
if (PyObject_GetBuffer(obj, &view, PyBUF_SIMPLE) < 0) {
    return NULL;
}
// Use `view.buf` and `view.len` to read from the buffer.
// You may need to cast buf as `(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;
}
// Use `view.buf` and `view.len` to write to the buffer.
PyBuffer_Release(&view);
```

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

- Remova as seguintes funções antigas da configuração de inicialização do Python, descontinuadas no Python 3.11:

- `PySys_AddWarnOptionUnicode()`: use `PyConfig.warnoptions`.
- `PySys_AddWarnOption()`: use `PyConfig.warnoptions`.
- `PySys_AddXOption()`: use `PyConfig.xoptions`.
- `PySys_HasWarnOptions()`: use `PyConfig.xoptions`.
- `PySys_SetPath()`: defina `PyConfig.module_search_paths`.
- `Py_SetPath()`: defina `PyConfig.module_search_paths`.

- `Py_SetStandardStreamEncoding()`: defina `PyConfig.stdio_encoding` e talvez também defina `PyConfig.legacy_windows_stdio` (no Windows).
- `_Py_SetProgramFullPath()`: defina `PyConfig.executable`.

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

- Remove a função `PyEval_ThreadsInitialized()`, descontinuada no Python 3.9. Desde Python 3.7, `Py_Initialize()` sempre cria a GIL: chamar `PyEval_InitThreads()` não faz nada e `PyEval_ThreadsInitialized()` sempre retorna diferente de zero. (Contribuição de Victor Stinner em [gh-105182](#).)
- Remove as funções `PyEval_AcquireLock()` e `PyEval_ReleaseLock()`, descontinuadas no Python 3.2. Eles não atualizavam o estado atual da thread. Elas podem ser substituídas por:
 - `PyEval_SaveThread()` e `PyEval_RestoreThread()`;
 - `PyEval_AcquireThread()` e `PyEval_RestoreThread()` de baixo nível;
 - ou `PyGILState_Ensure()` e `PyGILState_Release()`.

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

- Remove a função privada `_PyObject_FastCall()`: use `PyObject_Vectorcall()`, a qual está disponível desde Python 3.8 (**PEP 590**). (Contribuição de Victor Stinner em [gh-106023](#).)
- Remove o arquivo de cabeçalho `cpython/pytime.h`: ele continha apenas funções privadas. (Contribuição de Victor Stinner em [gh-106316](#).)
- Remove o apelido `_PyInterpreterState_Get()` de `PyInterpreterState_Get()` que era mantido para compatibilidade com versões anteriores do Python 3.8. O projeto [pythoncapi-compat](#) pode ser usado para obter `PyInterpreterState_Get()` no Python 3.8 e versões anteriores. (Contribuição de Victor Stinner em [gh-106320](#).)
- A função `PyModule_AddObject()` agora está suavemente descontinuado: funções `PyModule_Add()` ou `PyModule_AddObjectRef()` devem ser usadas em seu lugar. (Contribuição de Serhiy Storchaka em [gh-86493](#).)

14.4 APIs C descontinuadas

- Descontinua os antigos tipos `Py_UNICODE` e `PY_UNICODE_TYPE`: use diretamente o tipo `wchar_t`. Desde o Python 3.3, `Py_UNICODE` e `PY_UNICODE_TYPE` são apenas apelidos para `wchar_t`. (Contribuição de Victor Stinner em [gh-105156](#).)
- Descontinua antigas funções de inicialização do Python
 - `PySys_ResetWarnOptions()`: apague `sys.warnoptions` e `warnings.filters`.
 - `Py_GetExecPrefix()`: obtenha `sys.exec_prefix`.
 - `Py_GetPath()`: obtenha `sys.path`.
 - `Py_GetPrefix()`: obtenha `sys.prefix`.
 - `Py_GetProgramFullPath()`: obtenha `sys.executable`.
 - `Py_GetProgramName()`: obtenha `sys.executable`.
 - `Py_GetPythonHome()`: obtenha `PyConfig.home` ou a variável de ambiente `PYTHONHOME`.

Funções agendadas para remoção no Python 3.15. (Contribuição de Victor Stinner em [gh-105145](#).)

- Descontinua a função `PyImport_ImportModuleNoBlock()` que é apenas um apelido para `PyImport_ImportModule()` desde o Python 3.3. Programado para remoção no Python 3.15. (Contribuição de Victor Stinner em [gh-105396](#).)

- Descontinua as funções `PyWeakref_GetObject()` e `PyWeakref_GET_OBJECT()`, que retornam uma referência emprestada: use a nova função `PyWeakref_GetRef()` em vez disso, ela retorna uma referência forte. O projeto [pythoncapi-compat](#) pode ser usado para obter `PyWeakref_GetRef()` no Python 3.12 e versões anteriores. (Contribuição de Victor Stinner em [gh-105927](#).)
- Descontinua as funções `PyEval_GetBuiltins()`, `PyEval_GetGlobals()` e `PyEval_GetLocals()`, que retornam uma referência emprestada. Consulte os avisos de descontinuação em cada função para ver as substituições recomendadas. (Suavemente descontinuadas como parte da [PEP 667](#).)

14.5 Remoção pendente no Python 3.14

- A criação de tipos imutáveis (`Py_TPFLAGS_IMMUTABLETYPE`) com bases mutáveis usando a API C.
- Funções para configurar a inicialização do Python, descontinuadas no Python 3.11:

- `PySys_SetArgvEx()`: defina `PyConfig.argv`.
- `PySys_SetArgv()`: defina `PyConfig.argv`.
- `Py_SetProgramName()`: defina `PyConfig.program_name`.
- `Py_SetPythonHome()`: defina `PyConfig.home`.

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

- Variáveis de configuração globais

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

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

14.6 Remoção pendente no Python 3.15

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

14.7 Remoção pendente em versões futuras

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

- `Py_TPFLAGS_HAVE_FINALIZE`: desnecessário desde o Python 3.8.
- `PyErr_Fetch()`: use `PyErr_GetRaisedException()`.
- `PyErr_NormalizeException()`: use `PyErr_GetRaisedException()`.
- `PyErr_Restore()`: use `PyErr_SetRaisedException()`.
- `PyModule_GetFilename()`: use `PyModule_GetFilenameObject()`.
- `PyOS_AfterFork()`: use `PyOS_AfterFork_Child()`.
- `PySlice_GetIndicesEx()`.
- `PyUnicode_AsDecodedObject()`.
- `PyUnicode_AsDecodedUnicode()`.
- `PyUnicode_AsEncodedObject()`.
- `PyUnicode_AsEncodedUnicode()`.
- `PyUnicode_READY()`: desnecessário desde o Python 3.12.
- `_PyErr_ChainExceptions()`.
- O membro `PyBytesObject.ob_shash`: chame `PyObject_Hash()`.
- O membro `PyDictObject.ma_version_tag`.
- API TLS:
 - `PyThread_create_key()`: use `PyThread_tss_alloc()`.
 - `PyThread_delete_key()`: use `PyThread_tss_free()`.

- `PyThread_set_key_value()`: use `PyThread_tss_set()`.
 - `PyThread_get_key_value()`: use `PyThread_tss_get()`.
 - `PyThread_delete_key_value()`: use `PyThread_tss_delete()`.
 - `PyThread_ReInitTLS()`: não mais necessário.
- Remove a constante `PY_TIMEOUT_MAX` não documentada da API C limitada. (Contribuição de Victor Stinner em [gh-110014](#).)

15 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 Estendidas Python

- PEP 11, [6](#), [32](#)
- PEP 587, [36](#)
- PEP 590, [36](#)
- PEP 594, [19](#)
- PEP 602, [4](#)
- PEP 626, [26](#)
- PEP 667, [3](#), [6](#), [3234](#), [37](#)
- PEP 696, [3](#)
- PEP 702, [3](#), [19](#)
- PEP 703, [3](#), [4](#), [7](#)
- PEP 705, [3](#), [18](#)
- PEP 709, [33](#)
- PEP 719, [3](#)
- PEP 730, [4](#), [6](#)
- PEP 737, [31](#)
- PEP 738, [4](#)
- PEP 742, [3](#)
- PEP 744, [3](#), [7](#)

- PYTHON_BASIC_REPL, [4](#)
- PYTHON_COLORS, [3](#), [5](#), [12](#)
- PYTHON_CPU_COUNT, [14](#)
- PYTHON_FROZEN_MODULES, [8](#)
- PYTHON_GIL, [7](#), [8](#)
- PYTHON_HISTORY, [8](#)
- PYTHON_PERF_JIT_SUPPORT, [8](#)
- PYTHONHOME, [36](#), [38](#)
- PYTHONLEGACYWINDOWSFSENCODING, [24](#)
- PYTHONSAFEPATH, [16](#)

R

RFC

- RFC 5280, [9](#)

V

variável de ambiente

- PYTHON_BASIC_REPL, [4](#)
- PYTHON_COLORS, [3](#), [5](#), [12](#)
- PYTHON_CPU_COUNT, [14](#)
- PYTHON_FROZEN_MODULES, [8](#)
- PYTHON_GIL, [7](#), [8](#)
- PYTHON_HISTORY, [8](#)
- PYTHON_PERF_JIT_SUPPORT, [8](#)
- PYTHONHOME, [36](#), [38](#)
- PYTHONLEGACYWINDOWSFSENCODING, [24](#)
- PYTHONSAFEPATH, [16](#)