What's New in Python

Release 3.11.10

A. M. Kuchling

setembro 09, 2024

Python Software Foundation
Email: docs@python.org

Sumário

Editor
Pablo Galindo Salgado

Este artigo explica os novos recursos no Python 3.11, em comparação com 3.10. Python 3.11 foi lançado em 24 de outubro de 2022. Veja changelog para uma lista completa de mudanças.

1 Resumo – Destaques da versão

- O Python 3.11 é entre 10-60% mais rápido que o Python 3.10. Em média, medimos uma aceleração de 1,25x no conjunto de benchmarks padrão. Veja *CPython mais rápido* para detalhes.

Novos recursos de sintaxe:

- *PEP 654: Grupos de exceção e except**

Novos recursos embutidos:

- *PEP 678: Exceções podem ser enriquecidas com notas*

Novos módulos da biblioteca padrão:

- **PEP 680**: `tomllib` — Suporte para analisar TOML na Biblioteca Padrão

Melhorias no interpretador:

- *PEP 657: Localizações de erros refinadas em tracebacks*
- Nova opção de linha de comando `-P` e variável de ambiente `PYTHONSAFEPATH` para *desabilitar automaticamente a anteposição automática de caminhos potencialmente inseguros* para `sys.path`

Novos recursos de tipagem:

- *PEP 646: Genéricos variádicos*
- *PEP 655: Marcando itens TypedDict individuais como obrigatórios ou não obrigatórios*
- *PEP 673: Tipo Self*
- *PEP 675: Tipo de string literal arbitrário*
- *PEP 681: Transformações de classe de dados*

Descontinuações, remoções e restrições importantes:

- **PEP 594**: *Muitos módulos legados de biblioteca padrão foram descontinuados* e serão removidos no Python 3.13
- **PEP 624**: *APIs do codificador Py_UNICODE foram removidas*
- **PEP 670**: *Macros convertidos em funções estáticas em linha*

2 Novas funcionalidades

2.1 PEP 657: Localizações de erros refinadas em tracebacks

Ao imprimir tracebacks (situação da pilha de execução), o interpretador agora apontará para a expressão exata que causou o erro, em vez de apenas a linha. Por exemplo:

```
Traceback (most recent call last):
  File "distance.py", line 11, in <module>
    print(manhattan_distance(p1, p2))
          ^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "distance.py", line 6, in manhattan_distance
    return abs(point_1.x - point_2.x) + abs(point_1.y - point_2.y)
                           ^^^^^^^^^
AttributeError: 'NoneType' object has no attribute 'x'
```

Versões anteriores do interpretador apontavam apenas para a linha, tornando ambíguo qual objeto era `None`. Esses erros aprimorados também podem ser úteis ao lidar com objetos `dict` profundamente aninhados e várias chamadas de função:

```
Traceback (most recent call last):
  File "query.py", line 37, in <module>
    magic_arithmetic('foo')
  File "query.py", line 18, in magic_arithmetic
    return add_counts(x) / 25
           ^^^^^^^^^^^^^
  File "query.py", line 24, in add_counts
    return 25 + query_user(user1) + query_user(user2)
                ^^^^^^^^^^^^^^^^
  File "query.py", line 32, in query_user
    return 1 + query_count(db, response['a']['b']['c']['user'], retry=True)
                           ~~~~~~~~~~~~~~~~~~~~^^^^^
TypeError: 'NoneType' object is not subscriptable
```

Bem como expressões aritméticas complexas:

```
Traceback (most recent call last):
  File "calculation.py", line 54, in <module>
    result = (x / y / z) * (a / b / c)
             ~~~~~~^~~
ZeroDivisionError: division by zero
```

Além disso, as informações usadas pela funcionalidade de traceback aprimorado são disponibilizadas por meio de uma API geral, que pode ser usada para correlacionar instruções bytecode com a localização do código-fonte. Essas informações podem ser recuperadas usando:

- O método `codeobject.co_positions()` no Python.
- A função `PyCode_Addr2Location()` na API C.

Veja **PEP 657** para mais detalhes. (Contribuição de Pablo Galindo, Batuhan Taskaya e Ammar Askar em bpo-43950.)

Nota: Esse recurso requer o armazenamento de posições de coluna em codeobjects, o que pode resultar em um pequeno aumento no uso de memória do interpretador e no uso de disco para arquivos Python compilados. Para evitar armazenar as informações extras e desativar a exibição das informações extras de rastreamento, use a opção de linha de comando `-X no_debug_ranges` ou a variável de ambiente `PYTHONNODEBUGRANGES`.

2.2 PEP 654: Grupos de exceção e `except*`

PEP 654 introduz recursos de linguagem que permitem que um programa levante e manipule várias exceções não relacionadas simultaneamente. Os tipos internos `ExceptionGroup` e `BaseExceptionGroup` tornam possível agrupar exceções e criá-las juntas, e a nova sintaxe `except*` generaliza `except` para corresponder a subgrupos de grupos de exceção.

Veja **PEP 654** para mais detalhes.

(Contribuição de Irit Katriel em bpo-45292. PEP escrita por Irit Katriel, Yury Selivanov e Guido van Rossum.)

2.3 PEP 678: Exceções podem ser enriquecidas com notas

O método `add_note()` é adicionado a `BaseException`. Ele pode ser usado para enriquecer exceções com informações de contexto que não estão disponíveis no momento em que a exceção é gerada. As notas adicionadas aparecem no traceback padrão.

Veja **PEP 678** para mais detalhes.

(Contribuição de Irit Katriel em bpo-45607. PEP escrita por Zac Hatfield-Dodds.)

2.4 Melhorias no inicializador `py.exe` do Windows

A cópia do launcher incluída no Python 3.11 foi significativamente atualizada. Ele agora oferece suporte à sintaxe de company/tag conforme definido em **PEP 514** usando o argumento `-V:`*`<company>/<tag>`* em vez do limitado `-`*`<major>.<minor>`*. Isso permite iniciar outras distribuições além de `PythonCore`, aquela hospedada em python.org.

Ao usar seletores `-V:`, tanto company quanto tag podem ser omitidos, mas todas as instalações serão pesquisadas. Por exemplo, `-V:OtherPython/` selecionará a "melhor" tag registrada para `OtherPython`, enquanto `-V:3.11` ou `-V:/3.11` selecionará o "best" com a tag `3.11`.

Ao usar os argumentos legados `-`*`<major>`*, `-`*`<major>.<minor>`*, `-`*`<major>-<bitness>`* ou `-`*`<major>.<minor>-<bitness>`*, todo o comportamento existente deve ser preservado de versões anteriores e somente lançamentos de `PythonCore` será selecionado. No entanto, o sufixo `-64` agora implica "não 32 bits" (não necessariamente x86-64), pois existem várias plataformas de 64 bits suportadas. Tempos de execução de 32 bits são detectados verificando a tag do tempo de execução para um sufixo `-32`. Todas as versões do Python desde 3.5 incluíram isso em suas compilações de 32 bits.

3 Novos recursos relacionados a dicas de tipo

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

3.1 PEP 646: Genéricos variádicos

PEP 484 introduzido anteriormente `TypeVar`, permitindo a criação de genéricos parametrizados com um único tipo. **PEP 646** adiciona `TypeVarTuple`, permitindo a parametrização com um número *arbitrário* de tipos. Em outras palavras, uma `TypeVarTuple` é uma variável do tipo *variádico*, habilitando genéricos *variádicos*.

Isso permite uma ampla variedade de casos de uso. Em particular, ele permite que o tipo de estruturas do tipo array em bibliotecas de computação numérica, como NumPy e TensorFlow, sejam parametrizadas com o array *shape*. Os verificadores de tipo estáticos agora poderão capturar bugs relacionados a formas no código que usa essas bibliotecas.

Veja **PEP 646** para mais detalhes.

(Contribuição de Matthew Rahtz em bpo-43224, com contribuições de Serhiy Storchaka e Jelle Zijlstra. PEP escrita por Mark Mendoza, Matthew Rahtz, Pradeep Kumar Srinivasan e Vincent Siles.)

3.2 PEP 655: Marcando itens `TypedDict` individuais como obrigatórios ou não obrigatórios

`Required` e `NotRequired` fornecem uma maneira direta de marcar se itens individuais em um `TypedDict` devem estar presentes. Anteriormente, isso só era possível usando herança.

Todos os campos ainda são obrigatórios por padrão, a menos que o parâmetro *total* seja definido como `False`, caso em que todos os campos ainda não são obrigatórios por padrão. Por exemplo, o seguinte especifica um `TypedDict` com uma chave obrigatória e uma chave não obrigatória:

```
class Movie(TypedDict):
   title: str
   year: NotRequired[int]

m1: Movie = {"title": "Black Panther", "year": 2018}  # OK
m2: Movie = {"title": "Star Wars"}  # OK (year is not required)
m3: Movie = {"year": 2022}  # ERROR (missing required field title)
```

A seguinte definição é equivalente:

```
class Movie(TypedDict, total=False):
   title: Required[str]
   year: int
```

Veja **PEP 655** para mais detalhes.

(Contribuição de David Foster e Jelle Zijlstra em bpo-47087. PEP escrita por David Foster.)

3.3 PEP 673: Tipo `Self`

A nova anotação `Self` fornece uma maneira simples e intuitiva de anotar métodos que retornam uma instância de sua classe. Isso se comporta da mesma forma que a abordagem baseada em `TypeVar` **especificada na PEP 484**, mas é mais concisa e fácil de seguir.

Casos de uso comuns incluem construtores alternativos fornecidos como `classmethods` e métodos `__enter__()` que retornam `self`:

```
class MyLock:
    def __enter__(self) -> Self:
        self.lock()
        return self

    ...

class MyInt:
    @classmethod
    def fromhex(cls, s: str) -> Self:
        return cls(int(s, 16))

    ...
```

`Self` também pode ser usado para anotar parâmetros de métodos ou atributos do mesmo tipo que sua classe de inclusão.

Veja **PEP 673** para mais detalhes.

(Contribuição de James Hilton-Balfe em bpo-46534. PEP escrita por Pradeep Kumar Srinivasan e James Hilton-Balfe.)

3.4 PEP 675: Tipo de string literal arbitrário

A nova anotação `LiteralString` pode ser usada para indicar que um parâmetro de função pode ser de qualquer tipo de string literal. Isso permite que uma função aceite tipos de string literais arbitrários, bem como strings criadas a partir de outras strings literais. Os verificadores de tipo podem então impor que funções confidenciais, como aquelas que executam instruções SQL ou comandos de console, sejam chamadas apenas com argumentos estáticos, fornecendo proteção contra ataques de injeção.

Por exemplo, uma função de consulta SQL pode ser anotada da seguinte forma:

```
def run_query(sql: LiteralString) -> ...
    ...

def caller(
    arbitrary_string: str,
    query_string: LiteralString,
    table_name: LiteralString,
) -> None:
    run_query("SELECT * FROM students")       # ok
    run_query(query_string)                   # ok
    run_query("SELECT * FROM " + table_name)  # ok
    run_query(arbitrary_string)               # type checker error
    run_query(                                # type checker error
        f"SELECT * FROM students WHERE name = {arbitrary_string}"
    )
```

Veja **PEP 675** para mais detalhes.

(Contribuição de Jelle Zijlstra em bpo-47088. PEP escrita por Pradeep Kumar Srinivasan e Graham Bleaney.)

3.5 PEP 681: Transformações de classe de dados

`dataclass_transform` pode ser usado para decorar uma classe, metaclasse ou uma função que é um decorador. A presença de `@dataclass_transform()` informa a um verificador de tipo estático que o objeto decorado executa uma "mágica" de tempo de execução que transforma uma classe, dando a ela comportamentos semelhantes a `dataclass`.

Por exemplo:

```
# The create_model decorator is defined by a library.
@typing.dataclass_transform()
def create_model(cls: Type[T]) -> Type[T]:
    cls.__init__ = ...
    cls.__eq__ = ...
    cls.__ne__ = ...
    return cls

# The create_model decorator can now be used to create new model classes:
@create_model
class CustomerModel:
    id: int
    name: str

c = CustomerModel(id=327, name="Eric Idle")
```

Veja **PEP 681** para mais detalhes.

(Contribuição de Jelle Zijlstra em gh-91860. PEP escrita por Erik De Bonte e Eric Traut.)

3.6 PEP 563 pode não ser o futuro

PEP 563 Avaliação adiada de anotações (a `from __future__ import annotations` instrução future) que foi originalmente planejada para lançamento no Python 3.10 foi colocada em espera indefinidamente. Veja esta mensagem do Conselho Diretor para mais informações.

4 Outras mudanças na linguagem

- Expressões de desempacotamento com asterisco agora podem ser usadas em instruções `for`. (Veja bpo-46725 para mais detalhes.)
- Compreensões assíncronas agora são permitidas dentro de compreensões em funções assíncronas. As compreensões externas tornam-se implicitamente assíncronas neste caso. (Contribuição de Serhiy Storchaka em bpo-33346.)
- Uma exceção `TypeError` agora é levantada em vez de uma `AttributeError` em instruções `with` e `contextlib.ExitStack.enter_context()` para objetos que não oferecem suporte ao protocolo de gerenciador de contexto, e em instruções `async with` e `contextlib.AsyncExitStack.enter_async_context()` para objetos que não oferecem suporte ao protocolo de gerenciador de contexto assíncrono. (Contribuição de Serhiy Storchaka em bpo-12022 e bpo-44471.)
- Adicionado `object.__getstate__()`, que fornece a implementação padrão do método `__getstate__()`. Fazer `copy` e `pickle` de instâncias de subclasses de tipos embutidos `bytearray`, `set`, `frozenset`, `collections.OrderedDict`, `collections.deque`, `weakref.WeakSet` e `datetime.tzinfo` agora copia e serializa com pickle atributos de instância implementados como slots. Essa mudança tem um efeito colateral não intencional: ela atrapalha uma pequena minoria de projetos Python existentes que não esperavam que `object.__getstate__()` existisse. Consulte os comentários posteriores

em gh-70766 para discussões sobre quais soluções alternativas esse código pode precisar. (Contribuição de Serhiy Storchaka em bpo-26579.)

- Adicionada uma opção de linha de comando `-P` e uma variável de ambiente `PYTHONSAFEPATH`, que desativa o prefixo automático para `sys.path` do diretório do script ao executar um script ou o diretório atual ao usar `-c` e `-m`. Isso garante que apenas os módulos stdlib e instalados sejam escolhidos por `import`, e evita sombreamento involuntário ou mal-intencionado de módulos com aqueles em um diretório local (e normalmente gravável pelo usuário). (Contribuição de Victor Stinner em gh-57684.)

- Uma opção `"z"` foi adicionada ao formatspec que força zero negativo para positivo após o arredondamento para a precisão do formato. Veja **PEP 682** para mais detalhes. (Contribuição de John Belmonte em gh-90153.)

- Bytes não são mais aceitos em `sys.path`. O suporte quebrou em algum momento entre o Python 3.2 e o 3.6, sem que ninguém percebesse até o lançamento do Python 3.10.0. Além disso, trazer de volta o suporte seria problemático devido às interações entre `-b` e `sys.path_importer_cache` quando há uma mistura de chaves `str` e `bytes`. (Contribuição de Thomas Grainger em gh-91181.)

5 Outras mudanças na implementação do CPython

- Os métodos especiais `__complex__()` para `complex` e `__bytes__()` para `bytes` são implementados para oferecer suporte aos protocolos `typing.SupportsComplex` e `typing.SupportsBytes`. (Contribuição de Mark Dickinson e Donghee Na em bpo-24234.)

- `siphash13` foi adicionado como um novo algoritmo de hash interno. Ele tem propriedades de segurança semelhantes ao `siphash24`, mas é um pouco mais rápido para entradas longas. `str`, `bytes` e alguns outros tipos agora o usam como o algoritmo padrão para `hash()`. Arquivos .pyc baseados em hash da **PEP 552** agora usam `siphash13` também. (Contribuição de Inada Naoki em bpo-29410.)

- Quando uma exceção ativa é reativada por uma instrução `raise` sem parâmetros, o traceback anexado a essa exceção agora é sempre `sys.exc_info()[1].__traceback__`. Isso significa que as alterações feitas no traceback na cláusula `except` atual são refletidas na exceção levantada novamente. (Contribuição de Irit Katriel em bpo-45711.)

- A representação do estado do interpretador de exceções tratadas (também conhecidas como `exc_info` ou `_PyErr_StackItem`) agora tem apenas o campo `exc_value`; `exc_type` e `exc_traceback` foram removidos, pois podem ser derivados de `exc_value`. (Contribuição de Irit Katriel em bpo-45711.)

- Uma nova opção de linha de comando, `AppendPath`, foi adicionada para o instalador do Windows. Ela se comporta de maneira semelhante ao `PrependPath`, mas anexa os diretórios de instalação e scripts em vez de antecedê-los. (Contribuição de Bastian Neuburger em bpo-44934.)

- O campo `PyConfig.module_search_paths_set` agora deve ser definido como `1` para a inicialização usar `PyConfig.module_search_paths` para inicializar `sys.path`. Caso contrário, a inicialização irá recalcular o caminho e substituir quaisquer valores adicionados a `module_search_paths`.

- A saída da opção `--help` agora cabe em 50 linhas/80 colunas. Informações sobre as variáveis de ambiente do Python e a opção `-X` agora estão disponíveis usando os respectivos sinalizadores `--help-env` e `--help-xoptions`, e com o novo `--help-all`. (Contribuição de Éric Araujo em bpo-46142.)

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

6 Novos módulos

- `tomllib`: Para analisar TOML. Veja **PEP 680** para mais detalhes. (Contribuição de Taneli Hukkinen em bpo-40059.)
- `wsgiref.types`: Tipos específicos de **WSGI** para verificação de tipo estático. (Contribuição de Sebastian Rittau em bpo-42012.)

7 Módulos melhorados

7.1 asyncio

- Adicionada a classe `TaskGroup`, um assíncrono gerenciador de contexto contendo um grupo de tarefas que irá esperar por todas elas ao sair. Para código novo, é recomendado usar `create_task()` e `gather()` diretamente. (Contribuição de Yury Selivanov e outros em gh-90908.)
- Adicionado `timeout()`, um gerenciador de contexto assíncrono para definir um tempo limite em operações assíncronas. Para código novo, é recomendado usar `wait_for()` diretamente. (Contribuição de Andrew Svetlov em gh-90927.)
- Adicionada a classe `Runner`, que expõe o maquinário usado por `run()`. (Contribuição de Andrew Svetlov em gh-91218.)
- Adicionada a classe `Barrier` às primitivas de sincronização na biblioteca asyncio, e a exceção `BrokenBarrierError` relacionada. (Contribuição de Yves Duprat e Andrew Svetlov em gh-87518.)
- Adicionado o argumento nomeado *all_errors* para `asyncio.loop.create_connection()` para que vários erros de conexão possam ser levantados como `ExceptionGroup`.
- Adicionado o método `asyncio.StreamWriter.start_tls()` para atualizar conexões baseadas em fluxo existentes para TLS. (Contribuição de Ian Good em bpo-34975.)
- Adicionadas funções de soquete de datagrama *raw* ao laço de eventos: `sock_sendto()`, `sock_recvfrom()` e `sock_recvfrom_into()`. Eles têm implementações em `SelectorEventLoop` e `ProactorEventLoop`. (Contribuição de Alex Grönholm em bpo-46805.)
- Adicionados os métodos `cancelling()` e `uncancel()` a `Task`. Estes são destinados principalmente para uso interno, notadamente por `TaskGroup`.

7.2 contextlib

- Adicionado gerenciador de contexto não paralelo seguro `chdir()` para alterar o diretório de trabalho atual e depois restaurá-lo na saída. Wrapper simples em torno de `chdir()`. (Contribuição de Filipe Laíns em bpo-25625)

7.3 dataclasses

- Alterar a verificação de mutabilidade padrão do campo, permitindo apenas padrões que são hasheáveis em vez de qualquer objeto que não seja uma instância de `dict`, `list` ou `set`. (Contribuição de Eric V. Smith em bpo-44674.)

7.4 datetime

- Adicione `datetime.UTC`, um apelido de conveniência para `datetime.timezone.utc`. (Contribuição de Kabir Kwatra em gh-91973.)
- `datetime.date.fromisoformat()`, `datetime.time.fromisoformat()` e `datetime.datetime.fromisoformat()` agora podem ser usados para analisar a maioria dos formatos ISO 8601 (exceto aqueles que têm suporte a horas fracionárias e minutos). (Contribuição de Paul Ganssle em gh-80010.)

7.5 enum

- Renomeado `EnumMeta` para `EnumType` (`EnumMeta` mantido como um apelido).
- Adicionada `StrEnum`, com membros que podem ser usados como (e devem ser) strings.
- Adicionada `ReprEnum`, que apenas modifica o `__repr__()` dos membros enquanto retorna seus valores literais (ao invés de nomes) para `__str__()` e `__format__()` (usado por `str()`, `format()` e f-strings).
- Alterado `Enum.__format__()` (o padrão para `format()`, `str.format()` e f-strings) para sempre produz o mesmo resultado que `Enum.__str__()`: para enums herdados de `ReprEnum` será o valor do membro; para todos os outros enums, será o enum e o nome do membro (por exemplo, `Color.RED`).
- Adicionado um novo parâmetro de classe *boundary* para enums `Flag` e `FlagBoundary` com suas opções, para controlar como lidar com valores de sinalizadores fora do intervalo.
- Adicionado o decorador de enum `verify()` e o enum `EnumCheck` com suas opções, para verificar classes de enum em relação a várias restrições específicas.
- Adicionados os decoradores `member()` e `nonmember()`, para garantir que o objeto decorado seja/não seja convertido em um membro enum.
- Adicionado o decorador `property()`, que funciona como `property()` exceto para enums. Use isso em vez de `types.DynamicClassAttribute()`.
- Adicionado o decorador enum `global_enum()`, que ajusta `__repr__()` e `__str__()` para mostrar valores como membros de seu módulo em vez da classe enum. Por exemplo, `'re.ASCII'` para o membro `ASCII` de `re.RegexFlag` em vez de `'RegexFlag.ASCII'`.
- Aprimorada `Flag` para ter suporte a `len()`, iteração e `in`/`not in` em seus membros. Por exemplo, o seguinte agora funciona: `len(AFlag(3)) == 2 and list(AFlag(3)) == (AFlag.ONE, AFlag.TWO)`
- Alterada `Enum` e `Flag` para que os membros sejam agora definidos antes de `__init_subclass__()` ser chamado; `dir()` agora inclui métodos, etc., de tipos de dados mistos.
- Alterada `Flag` para considerar apenas valores primários (potência de dois) canônicos enquanto valores compostos (`3`, `6`, `10`, etc.) são considerados apelidos; bandeiras invertidas são coagidas ao seu equivalente positivo.

7.6 fcntl

- No FreeBSD, os sinalizadores `F_DUP2FD` e `F_DUP2FD_CLOEXEC` respectivamente são suportados, o primeiro é igual ao uso de `dup2` enquanto o último define o sinalizador `FD_CLOEXEC` adicionalmente.

7.7 fractions

- Suporte à inicialização no estilo **PEP 515** de `Fraction` da string. (Contribuição de Sergey B Kirpichev em bpo-44258.)
- `Fraction` agora implementa um método `__int__`, para que uma verificação de `isinstance(some_fraction, typing.SupportsInt)` seja aprovada. (Contribuição de Mark Dickinson em bpo-44547.)

7.8 functools

- `functools.singledispatch()` agora suporta `types.UnionType` e `typing.Union` como anotações para o argumento de dispatch.:

```
>>> from functools import singledispatch
>>> @singledispatch
... def fun(arg, verbose=False):
...     if verbose:
...         print("Let me just say,", end=" ")
...     print(arg)
...
>>> @fun.register
... def _(arg: int | float, verbose=False):
...     if verbose:
...         print("Strength in numbers, eh?", end=" ")
...     print(arg)
...
>>> from typing import Union
>>> @fun.register
... def _(arg: Union[list, set], verbose=False):
...     if verbose:
...         print("Enumerate this:")
...     for i, elem in enumerate(arg):
...         print(i, elem)
...
```

(Contribuição de Yurii Karabas na bpo-46014.)

7.9 gzip

- A função `gzip.compress()` agora é mais rápida quando usada com o argumento **mtime=0**, pois delega a compactação inteiramente para uma única operação `zlib.compress()`. Há um efeito colateral dessa mudança: o cabeçalho do arquivo gzip contém um byte de "OS" em seu cabeçalho. Tradicionalmente, isso sempre foi definido com um valor de 255 representando "desconhecido" pelo módulo `gzip`. Agora, ao usar `compress()` com **mtime=0**, ele pode ser definido com um valor diferente pela biblioteca zlib C subjacente à qual o Python estava vinculado. (Veja gh-112346 para detalhes sobre o efeito colateral.)

7.10 hashlib

- `hashlib.blake2b()` e `hashlib.blake2s()` agora preferem libb2 em vez da cópia do Python. (Contribuição de Christian Heimes em bpo-47095.)
- O módulo interno `_sha3` com algoritmos SHA3 e SHAKE agora usa *tiny_sha3* em vez do *Keccak Code Package* para reduzir o código e o tamanho binário. O módulo `hashlib` prefere implementações otimizadas de SHA3 e SHAKE do OpenSSL. A alteração afeta apenas instalações sem suporte a OpenSSL. (Contribuição de Christian Heimes em bpo-47098.)
- Adiciona `hashlib.file_digest()`, uma função auxiliar para hash eficiente de arquivos ou objetos semelhantes a arquivos. (Contribuição de Christian Heimes em gh-89313.)

7.11 IDLE e idlelib

- Aplica realce de sintaxe em arquivos `.pyi`. (Contribuição de Alex Waygood e Terry Jan Reedy em bpo-45447.)
- Inclui prompts ao salvar o console com entradas e saídas. (Contribuição de Terry Jan Reedy em gh-95191.)

7.12 inspect

- Adiciona `getmembers_static()` para retornar todos os membros sem acionar a pesquisa dinâmica por meio do protocolo descritor. (Contribuição de Weipeng Hong em bpo-30533.)
- Adiciona `ismethodwrapper()` para verificar se o tipo de um objeto é um `MethodWrapperType`. (Contribuição de Hakan Çelik em bpo-29418.)
- Altera as funções relacionadas a quadros no módulo `inspect` para retornar novas instâncias de classe `FrameInfo` e `Traceback` (compatíveis com as anteriores interfaces do tipo tupla nomeada) que inclui as informações de posição estendidas **PEP 657** (número da linha final, coluna e coluna final). As funções afetadas são:
  - `inspect.getframeinfo()`
  - `inspect.getouterframes()`
  - `inspect.getinnerframes()`,
  - `inspect.stack()`
  - `inspect.trace()`

  (Contribuição por Pablo Galindo em gh-88116.)

7.13 locale

- Adiciona `locale.getencoding()` para obter a codificação local atual. É semelhante a `locale.getpreferredencoding(False)`, mas ignora o Modo UTF-8 do Python.

7.14 logging

- Adicionada `getLevelNamesMapping()` para retornar um mapeamento de nomes de nível de registro (por exemplo, `'CRITICAL'`) para os valores de seus levels correspondentes (por exemplo, `50`, por padrão). (Contribuição de Andrei Kulakovin em gh-88024.)
- Adicionado um método `createSocket()` a `SysLogHandler`, para corresponder a `SocketHandler.createSocket()` . Ele é chamado automaticamente durante a inicialização do manipulador e ao emitir um evento, se não houver nenhum soquete ativo. (Contribuição de Kirill Pinchuk em gh-88457.)

7.15 math

- Adiciona `math.exp2()`: retorna 2 elevado à potência de x. (Contribuição de Gideon Mitchell em bpo-45917.)
- Adiciona `math.cbrt()`: retorna a raiz cúbica de x. (Contribuição de Ajith Ramachandran em bpo-44357.)
- O comportamento de dois casos excepcionais `math.pow()` foi alterado, para consistência com a especificação IEEE 754. As operações `math.pow(0.0, -math.inf)` e `math.pow(-0.0, -math.inf)` agora retornam `inf`. Anteriormente eles levantavam `ValueError`. (Contribuição de Mark Dickinson em bpo-44339.)
- O valor `math.nan` agora está sempre disponível. (Contribuição de Victor Stinner em bpo-46917.)

7.16 operador

- Uma nova função `operator.call` foi adicionada, tal que `operator.call(obj, *args, **kwargs) == obj(*args, **kwargs)` (Contribuição de Antony Lee em bpo-44019.)

7.17 os

- No Windows, `os.urandom()` agora usa `BCryptGenRandom()`, em vez de `CryptGenRandom()` que foi descontinuado. (Contribuição de Dong-hee Na em bpo-44611.)
- As of 3.11.10, `os.mkdir()` and `os.makedirs()` on Windows now support passing a *mode* value of `0o700` to apply access control to the new directory. This implicitly affects `tempfile.mkdtemp()` and is a mitigation for CVE-2024-4030. Other values for *mode* continue to be ignored. (Contributed by Steve Dower in gh-118486.)

7.18 pathlib

- `glob()` e `rglob()` retornam apenas diretórios se *pattern* terminar com um separador de componentes de nome de caminho: `sep` ou `altsep`. (Contribuição de Eisuke Kawasima em bpo-22276 e bpo-33392.)

7.19 re

- Agrupamento atômico (`(?>...)`) e quantificadores possessivos (`*+`, `++`, `?+`, `{m,n}+`) agora são suportados em expressões regulares. (Contribuição de Jeffrey C. Jacobs e Serhiy Storchaka em bpo-433030.)

7.20 shutil

- Adicione o parâmetro opcional *dir_fd* em `shutil.rmtree()`. (Contribuição de Serhiy Storchaka em bpo--46245.)

7.21 socket

- Adiciona suporte ao soquete CAN para NetBSD. (Contribuição de Thomas Klausner em bpo-30512.)
- `create_connection()` tem a opção de levantar, em caso de falha na conexão, um `ExceptionGroup` contendo todos os erros ao invés de somente levantar o último erro. (Contribuição de Irit Katriel em bpo-29980.)

7.22 sqlite3

- Agora você pode desabilitar o autorizador passando `None` para `set_authorizer()`. (Contribuição de Erlend E. Aasland em bpo-44491.)
- O nome do agrupamento `create_collation()` agora pode conter qualquer caractere Unicode. Nomes de agrupamento com caracteres inválidos agora levantar `UnicodeEncodeError` em vez de `sqlite3.ProgrammingError`. (Contribuição de Erlend E. Aasland em bpo-44688.)
- As exceções `sqlite3` agora incluem o código de erro estendido SQLite como `sqlite_errorcode` e o nome do erro SQLite como `sqlite_errorname`. (Contribuição de Aviv Palivoda, Daniel Shahaf e Erlend E. Aasland em bpo-16379 e bpo-24139.)
- Adiciona `setlimit()` e `getlimit()` a `sqlite3.Connection` para definir e obter limites SQLite por base de conexão. (Contribuição de Erlend E. Aasland em bpo-45243.)
- `sqlite3` agora define `sqlite3.threadsafety` com base no modo de threading padrão com o qual a biblioteca SQLite subjacente foi compilada. (Contribuição de Erlend E. Aasland em bpo-45613.)
- O retorno de chamadas C `sqlite3` agora usam exceções que não podem ser levantadas se os rastreamentos de retorno de chamada estiverem habilitados. Os usuários agora podem registrar um `unraisable hook handler` para melhorar sua experiência de depuração. (Contribuição de Erlend E. Aasland em bpo-45828.)
- A busca por rollback não levanta mais `InterfaceError`. Em vez disso, deixamos para a biblioteca SQLite lidar com esses casos. (Contribuição de Erlend E. Aasland em bpo-44092.)
- Adiciona `serialize()` e `deserialize()` a `sqlite3.Connection` para serializar e desserializar bancos de dados. (Contribuição de Erlend E. Aasland em bpo-41930.)
- Adiciona `create_window_function()` a `sqlite3.Connection` para criar funções de janela agregadas. (Contribuição de Erlend E. Aasland em bpo-34916.)
- Adiciona `blobopen()` a `sqlite3.Connection`. `sqlite3.Blob` permite operações de E/S incrementais em blobs. (Contribuição de Aviv Palivoda e Erlend E. Aasland em bpo-24905.)

7.23 string

- Adiciona `get_identifiers()` e `is_valid()` a `string.Template`, que respectivamente retornam todos os marcadores de posição válidos e se quaisquer marcadores de posição inválidos estão presentes. (Contribuição de Ben Kehoe em gh-90465.)

7.24 sys

- `sys.exc_info()` agora deriva os campos `type` e `traceback` do `value` (a instância de exceção) de forma que quanto uma exceção é modificada enquanto está sendo tratada, são refletidos nos resultados de chamadas subsequentes para `exc_info()`. (Contribuição de Irit Katriel em bpo-45711.)
- Adiciona `sys.exception()` que retorna a instância de exceção ativa (equivalente a `sys.exc_info()[1]`). (Contribuição de Irit Katriel em bpo-46328.)
- Adiciona o sinalizador `sys.flags.safe_path`. (Contribuição de Victor Stinner em gh-57684.)

7.25 sysconfig

- Três novos esquemas de instalação (*posix_venv*, *nt_venv* e *venv*) foram adicionados e são usados quando o Python cria novos ambientes virtuais ou quando está sendo executado a partir de um ambiente virtual. Os dois primeiros esquemas (*posix_venv* e *nt_venv*) são específicos do SO para não Windows e Windows, o *venv* é essencialmente um apelido para um deles de acordo com o SO em que o Python é executado. Isso é útil para distribuidores downstream que modificam `sysconfig.get_preferred_scheme()`. O código de terceiros que cria novos ambientes virtuais deve usar o novo esquema de instalação *venv* para determinar os caminhos, assim como `venv`. (Contribuição de Miro Hrončok em bpo-45413.)

7.26 tempfile

- Objetos `SpooledTemporaryFile` agora implementam totalmente os métodos de `io.BufferedIOBase` ou `io.TextIOBase` (dependendo do modo de arquivo). Isso permite que eles funcionem corretamente com APIs que esperam objetos semelhantes a arquivos, como módulos de compactação. (Contribuição de Carey Metcalfe em gh-70363.)
- As of 3.11.10 on Windows, the default mode `0o700` used by `tempfile.mkdtemp()` now limits access to the new directory due to changes to `os.mkdir()`. This is a mitigation for CVE-2024-4030. (Contributed by Steve Dower in gh-118486.)

7.27 threading

- No Unix, se a função `sem_clockwait()` estiver disponível na biblioteca C (glibc 2.30 e mais recente), o método `threading.Lock.acquire()` agora usa o relógio monotônico (`time.CLOCK_MONOTONIC`) para o tempo limite, em vez de usar o relógio do sistema (`time.CLOCK_REALTIME`), para não ser afetado pelas alterações do relógio do sistema. (Contribuição de Victor Stinner em bpo-41710.)

7.28 time

- No Unix, `time.sleep()` agora usa a função `clock_nanosleep()` ou `nanosleep()`, se disponível, que tem uma resolução de 1 nanossegundo (10^{-9} segundos), em vez de usar `select()` que tem uma resolução de 1 microssegundo (10^{-6} segundos). (Contribuição de Benjamin Szőke e Victor Stinner em bpo-21302.)
- No Windows 8.1 e mais recente, `time.sleep()` agora usa um temporizador aguardável baseado em temporizadores de alta resolução que tem uma resolução de 100 nanossegundos (10^{-7} segundos). Anteriormente, tinha uma resolução de 1 milissegundo (10^{-3} segundos). (Contribuição de Benjamin Szőke, Donghee Na, Eryk Sun e Victor Stinner em bpo-21302 e bpo-45429.)

7.29 tkinter

- Adicionado o método `info_patchlevel()` que retorna a versão exata da biblioteca Tcl como uma tupla nomeada semelhante a `sys.version_info`. (Contribuição de Serhiy Storchaka em gh-91827.)

7.30 traceback

- Adiciona `traceback.StackSummary.format_frame_summary()` para permitir que os usuários substituam quais quadros aparecem no traceback e como eles são formatados. (Contribuição de Ammar Askar em bpo-44569.)
- Adiciona `traceback.TracebackException.print()`, que imprime a instância formatada `TracebackException` em um arquivo. (Contribuição de Irit Katriel em bpo-33809.)

7.31 typing

Para alterações principais, veja *Novos recursos relacionados a dicas de tipo*.

- Adiciona `typing.assert_never()` e `typing.Never`. `typing.assert_never()` é útil para pedir a um verificador de tipo para confirmar que uma linha de código não está acessível. Em tempo de execução, ele levanta uma exceção `AssertionError`. (Contribuição de Jelle Zijlstra em gh-90633.)
- Adiciona `typing.reveal_type()`. Isso é útil para perguntar a um verificador de tipo qual tipo ele inferiu para uma determinada expressão. Em tempo de execução imprime o tipo do valor recebido. (Contribuição de Jelle Zijlstra em gh-90572.)
- Adicione `typing.assert_type()`. Isso é útil para solicitar a um verificador de tipos que confirme se o tipo que ele inferiu para uma determinada expressão corresponde ao tipo fornecido. Em tempo de execução ele simplesmente retorna o valor recebido. (Contribuição de Jelle Zijlstra em gh-90638.)
- Os tipos `typing.TypedDict` agora podem ser genéricos. (Contribuição de Samodya Abeysiriwardane em gh-89026.)
- Os tipos `NamedTuple` agora podem ser genéricos. (Contribuição de Serhiy Storchaka em bpo-43923.)
- Permite criar subclasse de `typing.Any`. Isso é útil para evitar erros de verificador de tipo relacionados a classes altamente dinâmicas, como mocks. (Contribuição de Shantanu Jain em gh-91154.)
- O decorador `typing.final()` agora define o `__final__` atribuído no objeto decorado. (Contribuição de Jelle Zijlstra em gh-90500.)
- A função `typing.get_overloads()` pode ser usada para introspecção das sobrecargas de uma função. `typing.clear_overloads()` pode ser usado para limpar todas as sobrecargas registradas de uma função. (Contribuição de Jelle Zijlstra em gh-89263.)

- O método `__init__()` das subclasses `Protocol` agora é preservado. (Contribuição de Adrian Garcia Badarasco em gh-88970.)
- A representação de tipos de tupla vazias (`Tuple[()]`) está simplificada. Isso afeta a introspecção, por exemplo. `get_args(Tuple[()])` agora é avaliado como `()` em vez de `(),)`. (Contribuição de Serhiy Storchaka em gh-91137.)
- Afrouxa os requisitos de tempo de execução para anotações de tipo removendo a verificação chamável na função privada `typing._type_check`. (Contribuição de Gregory Beauregard em gh-90802.)
- `typing.get_type_hints()` agora tem suporte à avaliação de strings como referências diretas em apelidos genéricos da PEP 585. (Contribuição de Niklas Rosenstein em gh-85542.)
- `typing.get_type_hints()` não adiciona mais `Optional` aos parâmetros com `None` como padrão. (Contribuição de Nikita Sobolev em gh-90353.)
- `typing.get_type_hints()` agora tem suporte à avaliação de anotações `ClassVar` de strings simples. (Contribuição de Gregory Beauregard em gh-90711.)
- `typing.no_type_check()` não modifica mais classes e funções externas. Agora também marca corretamente os métodos de classe para não serem verificados quanto ao tipo. (Contribuição de Nikita Sobolev em gh-90729.)

7.32 unicodedata

- O banco de dados Unicode foi atualizado para a versão 14.0.0. (Contribuição de Benjamin Peterson em bpo-45190).

7.33 unittest

- Adicionados os métodos `enterContext()` e `enterClassContext()` da classe `TestCase`, método `enterAsyncContext()` da classe `IsolatedAsyncioTestCase` e a função `unittest.enterModuleContext()`. (Contribuição de Serhiy Storchaka em bpo-45046.)

7.34 venv

- Quando novos ambientes virtuais Python são criados, o esquema de instalação do sysconfig do *venv* é usado para determinar os caminhos dentro do ambiente. Quando o Python é executado em um ambiente virtual, o mesmo esquema de instalação é o padrão. Isso significa que os distribuidores downstream podem alterar o esquema de instalação padrão do sysconfig sem alterar o comportamento dos ambientes virtuais. O código de terceiros que também cria novos ambientes virtuais deve fazer o mesmo. (Contribuição de Miro Hrončok em bpo-45413.)

7.35 avisos

- `warnings.catch_warnings()` agora aceita argumentos para `warnings.simplefilter()`, fornecendo uma maneira mais concisa de ignorar localmente avisos ou convertê-los em erros. (Contribuição de Zac Hatfield-Dodds em bpo-47074.)

7.36 zipfile

- Adicionado suporte para especificar a codificação de nome de membro para leitura de metadados em um diretório `ZipFile` e cabeçalhos de arquivo. (Contribuição de Stephen J. Turnbull e Serhiy Storchaka em bpo-28080.)
- Adicionado `ZipFile.mkdir()` para criar novos diretórios dentro de arquivos ZIP. (Contribuição de Sam Ezeh em gh-49083.)
- Adicionado `stem`, `suffix` e `suffixes` a `zipfile.Path`. (Contribuição de Miguel Brito em gh-88261.)

8 Otimizações

Esta seção cobre otimizações específicas independentes do projeto *CPython mais rápido*, que é abordado em sua própria seção.

- O compilador agora otimiza a formatação simples de % no estilo printf em literais de string contendo apenas os códigos de formato `%s`, `%r` e `%a` e a torna tão rápida quanto uma expressão f-string correspondente. (Contribuição de Serhiy Storchaka em bpo-28307.)
- A divisão inteira (`//`) é melhor ajustada para otimização por compiladores. Agora é cerca de 20% mais rápido em x86-64 ao dividir um `int` por um valor menor que `2**30`. (Contribuição de Gregory P. Smith e Tim Peters em gh-90564.)
- `sum()` agora é quase 30% mais rápida para inteiros menores que `2**30`. (Contribuição de Stefan Behnel em gh-68264.)
- O redimensionamento de listas é simplificado para o caso comum, acelerando `list.append()` em ≈15% e compreensão de listas em até 20-30% (Contribuição de Dennis Sweeney em :gh: *91165*.)
- Dicionários não armazenam valores de hash quando todas as chaves são objetos Unicode, diminuindo o tamanho de `dict`. Por exemplo, `sys.getsizeof(dict.fromkeys("abcdefg"))` foi reduzido de 352 bytes para 272 bytes (23% menor) em plataformas de 64 bits. (Contribuição de Inada Naoki em bpo-46845.)
- Usar `asyncio.DatagramProtocol` agora é muito mais rápido ao transferir arquivos grandes por UDP, com velocidades mais de 100 vezes maiores para um arquivo de ≈60 MiB. (Contribuição de msoxzw em gh-91487.)
- As funções `comb()` e `perm()` de `math` são agora ≈10 vezes mais rápidas para argumentos grandes (com uma aceleração maior para *k* maiores). (Contribuição de Serhiy Storchaka em bpo-37295.)
- As funções `mean()`, `variance()` e `stdev()` do `statistics` agora consomem iteradores em uma passagem em vez de convertê-los em uma `list` primeiro. Isso é duas vezes mais rápido e pode economizar memória de forma substancial. (Contribuição de Raymond Hettinger em gh-90415.)
- `unicodedata.normalize()` agora normaliza strings de puro ASCII em tempo constante. (Contribuição de Dong-hee Na em bpo-44987.)

9 CPython mais rápido

O CPython 3.11 é em média 25% mais rápido que o CPython 3.10 medido com o conjunto de ferramentas de benchmarks pyperformance, quando compilado com GCC no Ubuntu Linux. Dependendo da sua carga de trabalho, a aceleração geral pode ser de 10 a 60%.

Este projeto se concentra em duas áreas principais em Python: *Inicialização mais rápida* e *Tempo de execução mais rápido*. Otimizações que não são abordadas neste projeto estão listadas em *Otimizações*.

9.1 Inicialização mais rápida

Importações congeladas/objetos de código estático

O Python armazena o bytecode no diretório __pycache__ para acelerar o carregamento do módulo.

Anteriormente na versão 3.10, a execução do módulo Python era assim:

```
Read __pycache__ -> Unmarshal -> Heap allocated code object -> Evaluate
```

No Python 3.11, os principais módulos essenciais para a inicialização do Python são "congelados". Isso significa que seus codeobjects (e bytecode) são alocados estaticamente pelo interpretador. Isso reduz as etapas no processo de execução do módulo para isso:

```
Statically allocated code object -> Evaluate
```

A inicialização do interpretador agora é 10-15% mais rápida no Python 3.11. Isso tem um grande impacto para programas de execução curta usando Python.

(Contribuição de Eric Snow, Guido van Rossum e Kumar Aditya em vários issues.)

9.2 Tempo de execução mais rápido

Quadros Python menos custosos e mais preguiçosos

Os quadros do Python, contendo informações de execução, são criados sempre que o Python chama uma função do Python. A seguir estão as novas otimizações de quadro:

- Simplificado o processo de criação do quadro.
- Alocação de memória evitada reutilizando generosamente o espaço do quadro na pilha C.
- Simplificada a estrutura interna do quadro para conter apenas as informações essenciais. Os quadros anteriormente continham informações extras de depuração e gerenciamento de memória.

Os antigos objetos quadros agora são criados apenas quando solicitados por depuradores ou por funções de introspecção do Python, como `sys._getframe()` e `inspect.currentframe()`. Para a maioria dos códigos de usuário, nenhum objeto quadro é criado. Como resultado, quase todas as chamadas de funções do Python aceleraram significativamente. Medimos uma aceleração de 3 a 7% no desempenho do py.

(Contribuição de Mark Shannon em bpo-44590.)

Chamadas de função Python em linha

Durante a chamada de uma função Python, o Python chamará uma função C de avaliação para interpretar o código dessa função. Isso efetivamente limita a recursão Python pura ao que é seguro para a pilha C.

No 3.11, quando o CPython detecta o código Python chamando outra função Python, ele configura um novo quadro e "pula" para o novo código dentro do novo quadro. Isso evita chamar a função de interpretação C completamente.

A maioria das chamadas de função do Python agora não consome espaço de pilha C, acelerando-as. Em funções recursivas simples como fibonacci ou fatorial, observamos um aumento de velocidade de 1,7x. Isso também significa que as funções recursivas podem recursar significativamente mais profundamente (se o usuário aumentar o limite de recursão com `sys.setrecursionlimit()`). Medimos uma melhoria de 1 a 3% no desempenho do py.

(Contribuição de Pablo Galindo e Mark Shannon em bpo-45256.)

PEP 659: Interpretador adaptativo especializado

PEP 659 é uma das partes principais do projeto Faster CPython. A ideia geral é que, embora o Python seja uma linguagem dinâmica, a maior parte do código possui regiões onde objetos e tipos raramente mudam. Este conceito é conhecido como *estabilidade de tipo*.

Em tempo de execução, o Python tentará procurar padrões comuns e estabilidade de tipo no código em execução. O Python substituirá a operação atual por uma mais especializada. Essa operação especializada usa caminhos rápidos disponíveis apenas para esses casos/tipos de uso, que geralmente superam suas contrapartes genéricas. Isso também traz outro conceito chamado *cache inline*, onde o Python armazena em cache os resultados de operações caras diretamente no bytecode.

O especializador também combinará certos pares de instruções comuns em uma superinstrução, reduzindo a sobrecarga durante a execução.

O Python só se especializará quando vir um código "quente" (executado várias vezes). Isso evita que o Python perca tempo com código de execução única. O Python também pode se especializar quando o código é muito dinâmico ou quando o uso muda. A especialização é tentada periodicamente e as tentativas de especialização não são muito caras, permitindo que a especialização se adapte às novas circunstâncias.

(PEP escrita por Mark Shannon, com ideias inspiradas por Stefan Brunthaler. Veja **PEP 659** para mais informações. Implementação por Mark Shannon e Brandt Bucher, com ajuda adicional de Irit Katriel e Dennis Sweeney.)

Operação	Forma	Especialização	Aceleração da operação	Contribuidor(es)

Operações binárias	`x + x`
	`x - x`
	`x * x`
		Adicionar, multiplicar e subtrair binários para tipos comuns como `int`, `float` e `str` usam caminhos rápidos personalizados para seus tipos subjacentes.	10%	Mark Shannon, Donghee Na, Brandt Bucher, Dennis Sweeney

Subscrição	`a[i]`	Subscrever tipos contêineres como `list`, `tuple` e `dict` indexam diretamente as estruturas de dados subjacentes. Subscrever `__getitem__()` personalizado também é inserido em linha de forma similar a *Chamadas de função Python em linha*.	10--25%	Irit Katriel, Mark Shannon
Armazenar com subscrição	`a[i] = z`	Similar à especialização de subscrição acima.	10--25%	Dennis Sweeney
Chamadas	`f(arg` `C(arg`	Chamadas para funções e tipos (C) embutidos comuns como `len()` e `str` chamam diretamente sua versão C subjacente. Isso evita passar pela convenção de chamada interna.	20%	Mark Shannon, Ken Jin
Carregar variável global	`print` `len`	O índice do objeto no espaço de nomes globais/embutidas é armazenado em cache. Carregar globais e embutidas requer zero pesquisas de espaço de nomes.	[1]	Mark Shannon
Carregar atributo	`o.attr`	Semelhante ao carregamento de variáveis globais. O índice do atributo dentro do espaço de nomes da classe/objeto é armazenado em cache. Na maioria dos casos, o carregamento de atributos exigirá zero pesquisas de espaço de nomes.	[2]	Mark Shannon
Carregar métodos para chamada	`o.meth(`	O endereço real do método é armazenado em cache. O carregamento de método agora não tem pesquisas de espaço de nomes – mesmo para classes com longas cadeias de herança.	10--20%	Ken Jin, Mark Shannon
Armazenar atributo	`o.attr = z`	Semelhante à otimização de carregamento de atributos.	2% no pyperformance	Mark Shannon
Desempacotar sequência	`*seq`	Especializado para contêineres comuns como `list` e `tuple`. Evita convenção de chamada interna.	8%	Brandt Bucher

[1] Uma otimização semelhante já existia desde o Python 3.8. O 3.11 é especializado em mais formas e reduz algumas despesas gerais.

[2] Uma otimização semelhante já existia desde o Python 3.10. 3.11 é especializado em mais formas. Além disso, todos os carregamentos de atributos devem ser acelerados por bpo-45947.

9.3 Diversos

- Os objetos agora requerem menos memória devido aos espaços de nomes de objetos criados lentamente. Seus dicionários de espaços de nomes agora também compartilham chaves mais livremente. (Contribuição de Mark Shannon em bpo-45340 e bpo-40116.)
- Exceções de "custo zero" estão implementadas, eliminando o custo das instruções `try` quando nenhuma exceção é gerada. (Contribuição de Mark Shannon em bpo-40222.)
- Uma representação mais concisa de exceções no interpretador reduziu o tempo necessário para capturar uma exceção em cerca de 10%. (Contribuição de Irit Katriel em bpo-45711.)
- O mecanismo de correspondência de expressão regular do `re` foi parcialmente refatorado e agora usa *gotos* computados (ou "código encadeado") em plataformas suportadas. Como resultado, o Python 3.11 executa os benchmarks de expressão regular do pyperformance até 10% mais rápido que o Python 3.10. (Contribuição de Brandt Bucher em gh-91404.)

9.4 FAQ

Como devo escrever meu código para utilizar esses aceleradores?

Escrever código pythônico que segue as melhores práticas comuns; você não precisa alterar seu código. O projeto Faster CPython otimiza para padrões de código comuns que observamos.

O CPython 3.11 usará mais memória?

Talvez não; não esperamos que o uso de memória exceda 20% a mais que 3.10. Isso é compensado por otimizações de memória para objetos quadro e dicionários de objetos, conforme mencionado acima.

Não vejo nenhuma aceleração em minha carga de trabalho. Por que?

Certos códigos não terão benefícios perceptíveis. Se seu código passa a maior parte do tempo em operações de E/S ou já faz a maior parte de sua computação em uma biblioteca de extensão C como NumPy, não haverá acelerações significativas. Atualmente, esse projeto é o que mais beneficia as cargas de trabalho Python puro.

Além disso, os valores de desempenho py são uma média geométrica. Mesmo dentro dos benchmarks de pyperformance, alguns benchmarks desaceleraram um pouco, enquanto outros aceleraram quase 2x!

Existe um compilador JIT?

Não. Ainda estamos explorando outras otimizações.

9.5 Sobre

O Faster CPython explora otimizações para o CPython. A equipe principal é financiada pela Microsoft para trabalhar nisso em tempo integral. Pablo Galindo Salgado também é financiado pela Bloomberg LP para trabalhar no projeto em tempo parcial. Finalmente, muitos colaboradores são voluntários da comunidade.

10 Alterações de bytecode do CPython

O bytecode agora contém entradas de cache inline, que assumem a forma das instruções recém-adicionadas `CACHE`. Muitos códigos de operações, ou *opcodes*, esperam ser seguidos por um número exato de caches e instruem o interpretador a ignorá-los em tempo de execução. Os caches preenchidos podem parecer instruções arbitrárias, portanto, muito cuidado deve ser tomado ao ler ou modificar o bytecode adaptativo bruto que contém dados acelerados.

10.1 Novos opcodes

- `ASYNC_GEN_WRAP`, `RETURN_GENERATOR` e `SEND`, usados em geradores e corrotinas.
- `COPY_FREE_VARS`, que evita precisar de código especial no lado do chamado para fechamentos.
- `JUMP_BACKWARD_NO_INTERRUPT`, para usasr em certos loops nos quais o tratamento de interrupções é indesejável.
- `MAKE_CELL`, para criar cell-objects.
- `CHECK_EG_MATCH` e `PREP_RERAISE_STAR`, para manipular *novos grupos de exceção e except** adicionados na **PEP 654**.
- `PUSH_EXC_INFO`, para uso em manipuladores de exceção.
- `RESUME`, um *no-op*, para rastreamento interno, depuração e verificações de otimização.

10.2 Opcodes substituídos

Opcode(s) substituído(s)	Novo(s) opcode(s)	Notas
`BINARY_*` `INPLACE_*`	`BINARY_OP`	Substituiu todos os opcodes numéricos binários/no local por um único opcode

`CALL_FUNCTION`
`CALL_FUNCTION_KW`
`CALL_METHOD`
	`CALL`
	`KW_NAMES`
	`PRECALL`
	`PUSH_NULL`
		Separa a mudança de argumento para métodos da manipulação de argumentos nomeados; permite uma melhor especialização dos atendimentos

`DUP_TOP`
`DUP_TOP_TWO`
`ROT_TWO`
`ROT_THREE`
`ROT_FOUR`
`ROT_N`
	`COPY`
	`SWAP`
		Instruções de manipulação de pilha

`JUMP_IF_NOT_EXC_MATCH`	`CHECK_EXC_MATCH`	Agora efetua verificações, mas não pula

`JUMP_ABSOLUTE`
`POP_JUMP_IF_FALSE`
`POP_JUMP_IF_TRUE`
	`JUMP_BACKWARD`
	`POP_JUMP_BACKWARD_IF_*`
	`POP_JUMP_FORWARD_IF_*`
		Veja[3]. Variantes `TRUE`, `FALSE`, `NONE` e `NOT_NONE` para cada direção

`SETUP_WITH` `SETUP_ASYNC_WITH`	`BEFORE_WITH`	Configuração de bloco `with`

10.3 Opcodes alterados/removidos

- Alterados `MATCH_CLASS` e `MATCH_KEYS` para não enviar mais um valor booleano adicional para indicar sucesso/falha. Em vez disso, `None` é enviado em caso de falha no lugar da tupla de valores extraídos.
- Alterados opcodes que funcionam com exceções para refleti-los agora sendo representados como um item na pilha em vez de três (consulte gh-89874).
- Removidos `COPY_DICT_WITHOUT_KEYS`, `GEN_START`, `POP_BLOCK`, `SETUP_FINALLY` e `YIELD_FROM`.

[3] Todos os opcodes de salto agora são relativos, incluindo os existentes `JUMP_IF_TRUE_OR_POP` e `JUMP_IF_FALSE_OR_POP`. O argumento agora é um deslocamento da instrução atual em vez de um local absoluto.

11 Descontinuados

Esta seção lista as APIs do Python que foram descontinuadas no Python 3.11.

As APIs C descontinuadas são *listadas separadamente*.

11.1 Linguagem/Embutidos

- O encadeamento de descritores `classmethod` (introduzido em bpo-19072) agora foi descontinuado. Ele não pode mais ser usado para agrupar outros descritores como `property`. O design principal desse recurso era falho e causava 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.)
- Escapes octais em strings e bytes literais com valores maiores que `0o377` (255 em decimal) agora produzem um `DeprecationWarning`. Em uma versão futura do Python, eles levantarão um `SyntaxWarning` e eventualmente um `SyntaxError`. (Contribuição de Serhiy Storchaka em gh-81548.)
- A delegação de `int()` para `__trunc__()` agora foi descontinuada. Chamar `int(a)` quando `type(a)` implementa `__trunc__()`, mas não `__int__()` ou `__index__()`, agora levanta uma exceção `DeprecationWarning`. (Contribuição de Zackery Spytz em bpo-44977.)

11.2 Módulos

- **PEP 594** levou à descontinuação dos seguintes módulos programados para remoção no Python 3.13:

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

  (Contribuição de Brett Cannon em bpo-47061 e Victor Stinner em gh-68966.)
- Os módulos `asynchat`, `asyncore` e `smtpd` foram descontinuados desde pelo menos o Python 3.6. A documentação e os avisos de descontinuação foram atualizados para observar que serão removidos no Python 3.12. (Contribuição de Hugo van Kemenade em bpo-47022.)
- O pacote `lib2to3` e a ferramenta 2to3 agora foram descontinuados e podem não ser capazes de analisar o Python 3.10 ou mais recente. Veja **PEP 617**, apresentando o novo analisador GASE, para detalhes. (Contribuição de Victor Stinner em bpo-40360.)
- Módulos não documentados `sre_compile`, `sre_constants` e `sre_parse` agora fora descontinuados. (Contribuição de Serhiy Storchaka em bpo-47152.)

11.3 Biblioteca Padrão

- Os itens a seguir foram descontinuados em `configparser` desde o Python 3.2. Seus avisos de descontinuação agora foram atualizados para observar que serão removidos no Python 3.12:
  - a classe `configparser.SafeConfigParser`
  - a propriedade `configparser.ParsingError.filename`
  - o método `configparser.RawConfigParser.readfp()`

  (Contribuição de Hugo van Kemenade em bpo-45173.)
- `configparser.LegacyInterpolation` foi descontinuada na docstring desde o Python 3.2, e não está listada na documentação do `configparser`. Agora ela emite um `DeprecationWarning` e será removida no Python 3.13. Use `configparser.BasicInterpolation` ou `configparser.ExtendedInterpolation`. (Contribuição de Hugo van Kemenade em bpo-46607.)
- O antigo conjunto de funções `importlib.resources` foi descontinuado em favor das substituições adicionadas no Python 3.9 e será removido em uma versão futura do Python, devido ao não suporte de recursos localizados nos subdiretórios do pacote:
  - `importlib.resources.contents()`
  - `importlib.resources.is_resource()`
  - `importlib.resources.open_binary()`
  - `importlib.resources.open_text()`
  - `importlib.resources.read_binary()`
  - `importlib.resources.read_text()`
  - `importlib.resources.path()`
- A função `locale.getdefaultlocale()` foi descontinuada e será removida no Python 3.15. Use as funções `locale.setlocale()`, `locale.getpreferredencoding(False)` e `locale.getlocale()`. (Contribuição de Victor Stinner em gh-90817.)
- A função `locale.resetlocale()` foi descontinuada e será removida no Python 3.13. Use `locale.setlocale(locale.LC_ALL, "")`. (Contribuição de Victor Stinner em gh-90817.)
- Regras mais rígidas agora serão aplicadas para referências numéricas de grupos e nomes de grupos em expressões regulares. Apenas sequências de dígitos ASCII serão aceitas como referência numérica, e o nome do grupo em padrões `bytes` e strings de substituição podem conter apenas letras ASCII, dígitos e sublinhados. Por enquanto, um aviso de descontinuação é levantado para a sintaxe que viola essas regras. (Contribuição de Serhiy Storchaka em gh-91760.)
- No módulo `re`, a função `re.template()` e os sinalizadores `re.TEMPLATE` e `re.T` correspondentes foram descontinuados, pois não foram documentados e careciam da um propósito óbvio. Eles serão removidos no Python 3.13. (Contribuição de Serhiy Storchaka e Miro Hrončok em gh-92728.)
- `turtle.settiltangle()` foi descontinuada desde o Python 3.1; ela agora emite um aviso de descontinuação e será removida no Python 3.13. Em vez dela, use `turtle.tiltangle()` (foi marcada anteriormente incorretamente como descontinuada e sua docstring agora está corrigida). (Contribuição de Hugo van Kemenade em bpo-45837.)
- `typing.Text`, que existe apenas para fornecer suporte de compatibilidade entre o código Python 2 e Python 3, foi descontinuada nesta versão. Sua remoção não está planejada, mas os usuários são encorajados a usar `str` sempre que possível. (Contribuição de Alex Waygood em gh-92332.)
- A sintaxe do argumento nomeado para a construção de tipos `typing.TypedDict` foi descontinuado neste versão. O suporte será removido no Python 3.13. (Contribuição de Jingchen Ye em gh-90224.)

- `webbrowser.MacOSX` foi descontinuada e será removida no Python 3.13. Ela não foi testada, não foi documentada e não é usada pelo próprio `webbrowser`. (Contribuição de Dong-hee Na em bpo-42255.)
- O comportamento de retornar um valor de um dos métodos de teste de `TestCase` e `IsolatedAsyncioTestCase` (diferente do valor padrão `None`) foi descontinuado nesta versão.
- Foram descontinuadas as seguintes funções `unittest` não formalmente documentadas, agendadas para remoção no Python 3.13:
  - `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 Erlend E. Aasland em bpo-5846.)
- `unittest.TestProgram.usageExit()` foi marcado como descontinuado, para ser removido em 3.13. (Contribuição de Carlos Damázio em gh-67048.)

12 Pendente remoção no Python 3.12

As seguintes APIs do Python foram descontinuadas em versões anteriores do Python e serão removidas no Python 3.12.

As APIs C com remoção pendente são *listadas separadamente*.

- O módulo `asynchat`
- O módulo `asyncore`
- Todo o pacote distutils
- The `imp` module
- O espaço de nomes `typing.io`
- O espaço de nomes `typing.re`
- `cgi.log()`
- `importlib.find_loader()`
- `importlib.abc.Loader.module_repr()`
- `importlib.abc.MetaPathFinder.find_module()`
- `importlib.abc.PathEntryFinder.find_loader()`
- `importlib.abc.PathEntryFinder.find_module()`
- `importlib.machinery.BuiltinImporter.find_module()`
- `importlib.machinery.BuiltinLoader.module_repr()`
- `importlib.machinery.FileFinder.find_loader()`
- `importlib.machinery.FileFinder.find_module()`
- `importlib.machinery.FrozenImporter.find_module()`

- `importlib.machinery.FrozenLoader.module_repr()`
- `importlib.machinery.PathFinder.find_module()`
- `importlib.machinery.WindowsRegistryFinder.find_module()`
- `importlib.util.module_for_loader()`
- `importlib.util.set_loader_wrapper()`
- `importlib.util.set_package_wrapper()`
- `pkgutil.ImpImporter`
- `pkgutil.ImpLoader`
- `pathlib.Path.link_to()`
- `sqlite3.enable_shared_cache()`
- `sqlite3.OptimizedUnicode()`
- Variável de ambiente `PYTHONTHREADDEBUG`
- Os seguintes aliases foram descontinuados no `unittest`:

Apelido descontinuado	Método	Descontinuado em
`failUnless`	`assertTrue()`	3.1
`failIf`	`assertFalse()`	3.1
`failUnlessEqual`	`assertEqual()`	3.1
`failIfEqual`	`assertNotEqual()`	3.1
`failUnlessAlmostEqual`	`assertAlmostEqual()`	3.1
`failIfAlmostEqual`	`assertNotAlmostEqual()`	3.1
`failUnlessRaises`	`assertRaises()`	3.1
`assert_`	`assertTrue()`	3.2
`assertEquals`	`assertEqual()`	3.2
`assertNotEquals`	`assertNotEqual()`	3.2
`assertAlmostEquals`	`assertAlmostEqual()`	3.2
`assertNotAlmostEquals`	`assertNotAlmostEqual()`	3.2
`assertRegexpMatches`	`assertRegex()`	3.2
`assertRaisesRegexp`	`assertRaisesRegex()`	3.2
`assertNotRegexpMatches`	`assertNotRegex()`	3.5

13 Removidos

Esta seção lista as APIs do Python que foram removidas no Python 3.11.

As APIs C removidas são *listadas separadamente*.

- Removido o decorador `@asyncio.coroutine()` permitindo que corrotinas baseadas em gerador legado sejam compatíveis com o código `async` / `await`. A função foi descontinuada desde o Python 3.8 e a remoção foi inicialmente agendada para o Python 3.10. Use `async def` em vez disso. (Contribuição de Illia Volochii em bpo-43216.)
- Removida `asyncio.coroutines.CoroWrapper`, que era usada para encapsular objetos de corrotina baseados em gerador legado no modo de depuração. (Contribuição de Illia Volochii em bpo-43216.)

- Devido a preocupações significativas de segurança, o parâmetro *reuse_address* de `asyncio.loop.create_datagram_endpoint()`, desabilitado no Python 3.9, agora foi totalmente removido. Isto é devido ao comportamento da opção de socket `SO_REUSEDDR` no UDP. (Contribuição de Hugo van Kemenade em bpo-45129.)

- Removido o módulo `binhex`, descontinuado no Python 3.9. Também foram removidas as funções `binascii` relacionadas e igualmente descontinuadas:

  - `binascii.a2b_hqx()`
  - `binascii.b2a_hqx()`
  - `binascii.rlecode_hqx()`
  - `binascii.rldecode_hqx()`

  A função `binascii.crc_hqx()` continua disponível.

  (Contribuição de Victor Stinner em bpo-45085.)

- Removido o comando `bdist_msi` do `distutils` que foi descontinuado no Python 3.9. Em vez disso, use `bdist_wheel` (pacotes wheel). (Contribuição de Hugo van Kemenade em bpo-45124.)

- Removidos os métodos `__getitem__()` das classes `xml.dom.pulldom.DOMEventStream`, `wsgiref.util.FileWrapper` e `fileinput.FileInput`, descontinuados desde o Python 3.9 . (Contribuição de Hugo van Kemenade em bpo-45132.)

- Removidas as funções descontinuadas `lgettext()`, `ldgettext()`, `lngettext()` e `ldngettext()` de `gettext`. Também removidas a função `bind_textdomain_codeset()`, os métodos `NullTranslations.output_charset()` e `NullTranslations.set_output_charset()`, e o parâmetro *codeset* de `translation()` e `install()`, já que eles são usados apenas para as funções `l*gettext()`. (Contribuição de Dong-hee Na e Serhiy Storchaka em bpo-44235.)

- Removido do módulo `inspect`:

  - A função `getargspec()`, descontinuada desde o Python 3.0; use `inspect.signature()` ou `inspect.getfullargspec()`.
  - A função `formatargspec()`, descontinuada desde o Python 3.5; use a função `inspect.signature()` ou o objeto `inspect.Signature` diretamente.
  - Os métodos não documentados `Signature.from_builtin()` e `Signature.from_function()`, descontinuados desde o Python 3.5; use o método `Signature.from_callable()`.

  (Contribuição de Hugo van Kemenade em bpo-45320.)

- Removido o método `__class_getitem__()` de `pathlib.PurePath`, pois não era usado e foi adicionado por engano em versões anteriores. (Contribuição de Nikita Sobolev em bpo-46483.)

- Removida a classe `MailmanProxy` do módulo `smtpd`, pois ela não pode ser usada sem o pacote `mailman` externo. (Contribuição de Dong-hee Na em bpo-35800.)

- Removido o método descontinuado `split()` de `_tkinter.TkappType`. (Contribuição de Erlend E. Aasland em bpo-38371.)

- Removido o suporte a pacote de espaço de nomes da descoberta de `unittest`. Foi introduzido no Python 3.4, mas está quebrado desde o Python 3.7. (Contribuição de Inada Naoki em bpo-23882.)

- Removido o método privado não documentado `float.__set_format__()`, anteriormente conhecido como `float.__setformat__()` no Python 3.7. Sua docstring dizia: "Você provavelmente não quer usar esta função. Ela existe principalmente para ser usada na conjunto de testes do Python." (Contribuição de Victor Stinner em bpo-46852.)

- O sinalizador de configuração `--experimental-isolated-subinterpreters` e a macro `EXPERIMENTAL_ISOLATED_SUBINTERPRETERS` correspondente foram removidos.
- Pynche – O Pythonically Natural Color and Hue Editor – foi removido de `Tools/scripts` e está sendo desenvolvido independentemente da árvore fonte do Python.

14 Portando para Python 3.11

Esta seção lista as alterações descritas anteriormente e outras correções de bugs na API do Python que podem exigir alterações em seu código Python.

As notas de portabilidade para a API C são *listadas separadamente*.

- `open()`, `io.open()`, `codecs.open()` e `fileinput.FileInput` não aceitam mais `'U'` ("nova linha universal") em o modo de arquivo. No Python 3, o modo "nova linha universal" é usado por padrão sempre que um arquivo é aberto no modo de texto, e o sinalizador `'U'` foi descontinuado desde o Python 3.3. O parâmetro de nova linha para essas funções controla como as novas linhas universais funcionam. (Contribuição de Victor Stinner em bpo-37330.)
- As posições do nó da `ast.AST` agora são validadas quando fornecidas para `compile()` e outras funções relacionadas. Se posições inválidas forem detectadas, uma exceção `ValueError` será levantado. (Contribuição de Pablo Galindo em gh-93351)
- Proibido passar executores que não sejam `concurrent.futures.ThreadPoolExecutor` para `asyncio.loop.set_default_executor()` após uma descontinuação no Python 3.8. (Contribuição de Illia Volochii em bpo-43234.)
- `calendar`: As classes `calendar.LocaleTextCalendar` e `calendar.LocaleHTMLCalendar` agora usam `locale.getlocale()`, em vez de usar `locale.getdefaultlocale()`, se nenhuma localidade é especificada. (Contribuição de Victor Stinner em bpo-46659.)
- O módulo `pdb` agora lê o arquivo de configuração `.pdbrc` com a codificação `'UTF-8'`. (Contribuição de Srinivas Reddy Thatiparty (□□□□□□□□□□□ □□□□□□ □□□□□□□□□□) em bpo-41137.)
- O parâmetro *population* de `random.sample()` deve ser uma sequência, e a conversão automática de `sets` para `lists` não é mais suportada. Além disso, se o tamanho da amostra for maior que o tamanho da população, uma exceção `ValueError` é levantada. (Contribuição de Raymond Hettinger em bpo-40465.)
- O parâmetro opcional *random* de `random.shuffle()` foi removido. Anteriormente, era uma função aleatória arbitrária a ser usada para o shuffle; agora, `random.random()` (seu padrão anterior) sempre será usado.
- Em re-syntax do `re`, sinalizadores globais inline (por exemplo, `(?i)`) agora só podem ser usados no início de expressões regulares. Se uso em outro lugar foi descontinuado no Python 3.6. (Contribuição de Serhiy Storchaka em bpo-47066.)
- No módulo `re`, vários bugs de longa data foram corrigidos que, em casos raros, poderiam fazer com que os grupos de captura obtivessem o resultado errado. Portanto, isso pode alterar a saída capturada nesses casos. (Contribuição de Ma Lin em bpo-35859.)

15 Alterações de compilação

- CPython agora tem **suporte a Tier 3** da **PEP 11** para compilação cruzada para as plataformas WebAssembly Emscripten (`wasm32-unknown-emscripten`, ou seja, Python no navegador) e WebAssembly System Interface (WASI) (`wasm32-unknown-wasi`). O esforço é inspirado em trabalhos anteriores como Pyodide. Essas plataformas fornecem um subconjunto limitado de APIs POSIX; Os recursos e módulos das bibliotecas padrão do Python relacionados a redes, processos, encadeamento, sinais, mmap e usuários/grupos não estão disponíveis ou não funcionam. (Emscripten contribuído por Christian Heimes e Ethan Smith em gh-84461 e WASI contribuído por Christian Heimes em gh-90473; plataformas promovidas em gh-95085)
- A construção do CPython agora requer:
  - Um compilador C11. Recursos C11 opcionais não são necessários. (Contribuição de Victor Stinner em bpo-46656, bpo-45440 e bpo-46640.)
  - Suporte para números de ponto flutuante IEEE 754. (Contribuição de Victor Stinner em bpo-46917.)
- A macro `Py_NO_NAN` foi removida. Como o CPython agora requer pontos flutuantes de IEEE 754, os valores NaN estão sempre disponíveis. (Contribuição de Victor Stinner em bpo-46656.)
- O pacote `tkinter` agora requer Tcl/Tk versão 8.5.12 ou mais recente. (Contribuição de Serhiy Storchaka em bpo-46996.)
- Dependências de construção, sinalizadores de compilador e sinalizadores de vinculador para a maioria dos módulos de extensão stdlib agora são detectados pelo **`configure`**. Os sinalizadores, ou *flags*, de libffi, libnsl, libsqlite3, zlib, bzip2, liblzma, libcrypt, Tcl/Tk e uuid são detectados por pkg-config (Quando disponível). `tkinter` agora requer um comando pkg-config para detectar configurações de desenvolvimento para cabeçalhos e bibliotecas Tcl/Tk. (Contribuição de Christian Heimes e Erlend Egeberg Aasland em bpo-45847, bpo-45747 e bpo-45763.)
- libpython não está mais vinculado a libcrypt. (Contribuição de Mike Gilbert em bpo-45433.)
- O CPython agora pode ser construído com a opção ThinLTO passando `thin` para `--with-lto`, ou seja, `--with-lto=thin`. (Contribuição de Donghee Na e Brett Holman em bpo-44340.)
- Freelists for object structs can now be disabled. A new **`configure`** option `--without-freelists` can be used to disable all freelists except empty tuple singleton. (Contributed by Christian Heimes in bpo-45522.)
- `Modules/Setup` e `Modules/makesetup` foram melhorados e amarrados. Módulos de extensão agora podem ser construídos através do `makesetup`. Todos, exceto alguns módulos de teste, podem ser vinculados estaticamente em um binário principal ou biblioteca. (Contribuição de Brett Cannon e Christian Heimes em bpo-45548, bpo-45570, bpo-45571 e bpo-43974.)

Nota: Use as variáveis de ambiente `TCLTK_CFLAGS` e `TCLTK_LIBS` para especificar manualmente a localização dos cabeçalhos e bibliotecas Tcl/Tk. As opções **`configure`** `--with-tcltk-includes` e `--with-tcltk-libs` foram removidas.

No RHEL 7 e CentOS 7 os pacotes de desenvolvimento não fornecem `tcl.pc` e `tk.pc`; use `TCLTK_LIBS="-ltk8.5 -ltkstub8.5 -ltcl8.5"`. O diretório `Misc/rhel7` contém arquivos `.pc` e instruções sobre como construir Python com RHEL 7's e CentOS 7's Tcl/Tk e OpenSSL.

- O CPython agora usará dígitos de 30 bits por padrão para a implementação de `int` do Python. Anteriormente, o padrão era usar dígitos de 30 bits em plataformas com `SIZEOF_VOID_P >= 8` e, caso contrário, dígitos de 15 bits. Ainda é possível solicitar explicitamente o uso de dígitos de 15 bits através da opção `--enable-big-digits` para o script de configuração ou (para Windows) a variável `PYLONG_BITS_IN_DIGIT` em `PC/pyconfig.h`, mas esta opção pode ser removida em algum momento no futuro. (Contribuição de Mark Dickinson em bpo-45569.)

16 Alterações na API C

16.1 Novas funcionalidades

- Adiciona uma nova função `PyType_GetName()` para obter o nome curto do tipo. (Contribuição de Hai Shi em bpo-42035.)
- Adiciona uma nova função `PyType_GetQualName()` para obter o nome qualificado do tipo. (Contribuição de Hai Shi em bpo-42035.)
- Adiciona novas funções `PyThreadState_EnterTracing()` e `PyThreadState_LeaveTracing()` à API C limitada para suspender e retomar rastreamento e criação de perfil. (Contribuição de Victor Stinner em bpo-43760.)
- Adicionada a constante `Py_Version` que possui o mesmo valor que `PY_VERSION_HEX`. (Contribuição de Gabriele N. Tornetta em bpo-43931.)
- `Py_buffer` e APIs agora fazem parte da API limitada e da ABI estável:
  - `PyObject_CheckBuffer()`
  - `PyObject_GetBuffer()`
  - `PyBuffer_GetPointer()`
  - `PyBuffer_SizeFromFormat()`
  - `PyBuffer_ToContiguous()`
  - `PyBuffer_FromContiguous()`
  - `PyObject_CopyData()`
  - `PyBuffer_IsContiguous()`
  - `PyBuffer_FillContiguousStrides()`
  - `PyBuffer_FillInfo()`
  - `PyBuffer_Release()`
  - `PyMemoryView_FromBuffer()`
  - Slotes de tipo `bf_getbuffer` e `bf_releasebuffer`

  (Contribuição de Christian Heimes em bpo-45459.)
- Adicionada a função `PyType_GetModuleByDef()`, utilizada para obter o módulo no qual um método foi definido, nos casos em que esta informação não está disponível diretamente (via `PyCMethod`). (Contribuição de Petr Viktorin em bpo-46613.)
- Adiciona novas funções para compactar e desempacotar double C (serializar e desserializar): `PyFloat_Pack2()`, `PyFloat_Pack4()`, `PyFloat_Pack8()`, `PyFloat_Unpack2()`, `PyFloat_Unpack4()` e `PyFloat_Unpack8()`. (Contribuição de Victor Stinner em bpo-46906.)
- Adiciona novas funções para obter atributos do objeto quadro: `PyFrame_GetBuiltins()`, `PyFrame_GetGenerator()`, `PyFrame_GetGlobals()`, `PyFrame_GetLasti()`.
- Adicionadas duas novas funções para obter e definir a instância de exceção ativa: `PyErr_GetHandledException()` e `PyErr_SetHandledException()`. Estas são alternativas para `PyErr_SetExcInfo()` e `PyErr_GetExcInfo()` que funcionam com a representação de tuplas de 3 elementos legadas de exceções. (Contribuição de Irit Katriel em bpo-46343.)
- Adicionada o membro `PyConfig.safe_path`. (Contribuição de Victor Stinner em gh-57684.)

16.2 Portando para Python 3.11

- Algumas macros foram convertidas em funções inline estáticas para evitar armadilhas de macro. A alteração deve ser transparente para os usuários, pois as funções de substituição converterão seus argumentos para os tipos esperados para evitar avisos do compilador devido a verificações de tipo estático. No entanto, quando a API C limitada é definida como >=3.11, essas conversões não são feitas e os chamadores precisarão converter argumentos para seus tipos esperados. Veja **PEP 670** para mais detalhes. (Contribuição de Victor Stinner e Erlend E. Aasland em gh-89653.)

- `PyErr_SetExcInfo()` não usa mais os argumentos `type` e `traceback`, o interpretador agora deriva esses valores da instância de exceção (o argumento `value`). A função ainda rouba referências de todos os três argumentos. (Contribuição de Irit Katriel em bpo-45711.)

- `PyErr_GetExcInfo()` agora deriva os campos `type` e `traceback` do resultado da instância de exceção (o campo `value`). (Contribuição de Irit Katriel em bpo-45711.)

- `_frozen` tem um novo campo `is_package` para indicar se o módulo frozen é ou não um pacote. Anteriormente, um valor negativo no campo `size` era o indicador. Agora apenas valores não negativos serão usados para `size`. (Contribuição de Kumar Aditya em bpo-46608.)

- `_PyFrameEvalFunction()` agora usa `_PyInterpreterFrame*` como seu segundo parâmetro, em vez de `PyFrameObject*`. Veja **PEP 523** para mais detalhes de como usar este tipo de ponteiro de função.

- `PyCode_New()` e `PyCode_NewWithPosOnlyArgs()` agora recebem um argumento adicional `exception_table`. O uso dessas funções deve ser evitado, se possível. Para obter um objeto código personalizado: crie um objeto código usando o compilador e obtenha uma versão modificada com o método `replace`.

- `PyCodeObject` não possui mais os campos `co_code`, `co_varnames`, `co_cellvars` e `co_freevars`. Em vez disso, use `PyCode_GetCode()`, `PyCode_GetVarnames()`, `PyCode_GetCellvars()` e `PyCode_GetFreevars()` respectivamente para acessá-los por meio da API C. (Contribuição de Brandt Bucher em bpo-46841 e Ken Jin em gh-92154 e gh-94936.)

- As antigas macros trashcan (`Py_TRASHCAN_SAFE_BEGIN`/`Py_TRASHCAN_SAFE_END`) foram descontinuadas. Elas devem ser substituídas pelas novas macros `Py_TRASHCAN_BEGIN` e `Py_TRASHCAN_END`.

  Uma função tp_dealloc que possui 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.

Para oferecer suporte a versões mais antigas do Python na mesma base de código, você pode definir as seguintes macros e usá-las em todo o código (crédito: elas foram copiadas da base de código do `mypy`):

```
#if PY_VERSION_HEX >= 0x03080000
#  define CPy_TRASHCAN_BEGIN(op, dealloc) Py_TRASHCAN_BEGIN(op, dealloc)
#  define CPy_TRASHCAN_END(op) Py_TRASHCAN_END
#else
#  define CPy_TRASHCAN_BEGIN(op, dealloc) Py_TRASHCAN_SAFE_BEGIN(op)
#  define CPy_TRASHCAN_END(op) Py_TRASHCAN_SAFE_END(op)
#endif
```

- A função `PyType_Ready()` agora levanta um erro se um tipo é definido com o sinalizador `Py_TPFLAGS_HAVE_GC` definido, mas não tem função transversal (`PyTypeObject.tp_traverse`). (Contribuição de Victor Stinner em bpo-44263.)

- Tipos de heap com o sinalizador `Py_TPFLAGS_IMMUTABLETYPE` agora podem herdar o protocolo vectorcall da **PEP 590**. Anteriormente, isso só era possível para tipos estáticos. (Contribuição de Erlend E. Aasland em bpo-43908)

- Uma vez que `Py_TYPE()` é alterada para uma função estática inline, `Py_TYPE(obj) = new_type` deve ser substituído por `Py_SET_TYPE(obj, new_type)`: consulte a função `Py_SET_TYPE()` (disponível desde Python 3.9). Para compatibilidade com versões anteriores, esta macro pode ser usada:

```
#if PY_VERSION_HEX < 0x030900A4 && !defined(Py_SET_TYPE)
static inline void _Py_SET_TYPE(PyObject *ob, PyTypeObject *type)
{ ob->ob_type = type; }
#define Py_SET_TYPE(ob, type) _Py_SET_TYPE((PyObject*)(ob), type)
#endif
```

  (Contribuição de Victor Stinner em bpo-39573.)

- Uma vez que `Py_SIZE()` é alterada para uma função estática inline, `Py_SIZE(obj) = new_size` deve ser substituído por `Py_SET_SIZE(obj, new_size)`: veja a função `Py_SET_SIZE()` (disponível desde Python 3.9). Para compatibilidade com versões anteriores, esta macro pode ser usada:

```
#if PY_VERSION_HEX < 0x030900A4 && !defined(Py_SET_SIZE)
static inline void _Py_SET_SIZE(PyVarObject *ob, Py_ssize_t size)
{ ob->ob_size = size; }
#define Py_SET_SIZE(ob, size) _Py_SET_SIZE((PyVarObject*)(ob), size)
#endif
```

  (Contribuição de Victor Stinner em bpo-39573.)

- `<Python.h>` não inclui mais os arquivos de cabeçalho `<stdlib.h>`, `<stdio.h>`, `<errno.h>` e `<string.h>` quando a macro `Py_LIMITED_API` é definida como `0x030b0000` (Python 3.11) ou superior. As extensões C devem incluir explicitamente os arquivos de cabeçalho após `#include <Python.h>`. (Contribuição de Victor Stinner em bpo-45434.)

- Os arquivos API não limitados `cellobject.h`, `classobject.h`, `code.h`, `context.h`, `funcobject.h`, `genobject.h` e `longintrepr.h` foram movidos para o diretório `Include/cpython`. Além disso, o arquivo de cabeçalho `eval.h` foi removido. Esses arquivos não devem ser incluídos diretamente, pois já estão incluídos aos arquivos de inclusão do `Python.h`. Se eles foram incluídos diretamente, considere incluir `Python.h` em seu lugar. (Contribuição de Victor Stinner em bpo-35134.)

- A macro `PyUnicode_CHECK_INTERNED()` foi excluída da API C limitada. Nunca foi utilizável lá, porque usava estruturas internas que não estão disponíveis na API C limitada. (Contribuição de Victor Stinner em bpo--46007.)

- As seguintes funções e tipo de quadro agora estão disponíveis diretamente com `#include <Python.h>`, não é mais necessário adicionar `#include <frameobject.h>`:

  - `PyFrame_Check()`
  - `PyFrame_GetBack()`
  - `PyFrame_GetBuiltins()`
  - `PyFrame_GetGenerator()`
  - `PyFrame_GetGlobals()`
  - `PyFrame_GetLasti()`
  - `PyFrame_GetLocals()`
  - `PyFrame_Type`

  (Contribuição de Victor Stinner em gh-93937.)

- Os membros da estrutura `PyFrameObject` foram removidos da API C pública.

  Embora a documentação observe que os campos `PyFrameObject` estão sujeitos a alterações a qualquer momento, eles permaneceram estáveis por um longo tempo e foram usados em várias extensões populares.

  No Python 3.11, a estrutura do quadro foi reorganizada para permitir otimizações de desempenho. Alguns campos foram totalmente removidos, pois eram detalhes da antiga implementação.

  Campos do `PyFrameObject`:

  - `f_back`: use `PyFrame_GetBack()`.
  - `f_blockstack`: removido.
  - `f_builtins`: use `PyFrame_GetBuiltins()`.
  - `f_code`: use `PyFrame_GetCode()`.
  - `f_gen`: use `PyFrame_GetGenerator()`.
  - `f_globals`: use `PyFrame_GetGlobals()`.
  - `f_iblock`: removido.
  - `f_lasti`: use `PyFrame_GetLasti()`. Código usando `f_lasti` com `PyCode_Addr2Line()` deve usar `PyFrame_GetLineNumber()`, pois deve ser mais rápido.
  - `f_lineno`: use `PyFrame_GetLineNumber()`
  - `f_locals`: use `PyFrame_GetLocals()`.
  - `f_stackdepth`: removido.
  - `f_state`: nenhuma API pública (renomeado para `f_frame.f_state`).
  - `f_trace`: nenhuma API pública.
  - `f_trace_lines`: use `PyObject_GetAttrString((PyObject*)frame, "f_trace_lines")`.
  - `f_trace_opcodes`: use `PyObject_GetAttrString((PyObject*)frame, "f_trace_opcodes")`.
  - `f_localsplus`: nenhuma API pública (renomeado para `f_frame.localsplus`).
  - `f_valuestack`: removido.

  O objeto quadro do Python agora é criado de forma "preguiçosa" (*lazy*). Um efeito colateral é que o membro `f_back` não deve ser acessado diretamente, já que seu valor agora também é calculado de forma "preguiçosa". A função `PyFrame_GetBack()` deve ser chamada em seu lugar.

Depuradores que acessam `f_locals` diretamente *devem* chamar `PyFrame_GetLocals()`. Eles não precisam mais chamar `PyFrame_FastToLocalsWithError()` ou `PyFrame_LocalsToFast()`. Na verdade, eles não devem chamar essas funções. A atualização necessária do quadro agora é gerenciada pela máquina virtual.

Código definindo `PyFrame_GetCode()` no Python 3.8 e anteriores:

```
#if PY_VERSION_HEX < 0x030900B1
static inline PyCodeObject* PyFrame_GetCode(PyFrameObject *frame)
{
    Py_INCREF(frame->f_code);
    return frame->f_code;
}
#endif
```

Código definindo `PyFrame_GetBack()` no Python 3.8 e anteriores:

```
#if PY_VERSION_HEX < 0x030900B1
static inline PyFrameObject* PyFrame_GetBack(PyFrameObject *frame)
{
    Py_XINCREF(frame->f_back);
    return frame->f_back;
}
#endif
```

Ou use o projeto pythoncapi_compat para obter essas duas funções em versões mais antigas do Python.

- Mudanças nos membros da estrutura `PyThreadState`:
  - `frame`: removido, use `PyThreadState_GetFrame()` (função adicionada ao Python 3.9 por bpo-40429). Aviso: a função retorna um referência forte, precisa chamar `Py_XDECREF()`.
  - `tracing`: alterado, use `PyThreadState_EnterTracing()` e `PyThreadState_LeaveTracing()` (funções adicionadas ao Python 3.11 por bpo-43760).
  - `recursion_depth`: removido, use `(tstate->recursion_limit - tstate->recursion_remaining)`.
  - `stackcheck_counter`: removido.

Código definindo `PyThreadState_GetFrame()` no Python 3.8 e anteriores:

```
#if PY_VERSION_HEX < 0x030900B1
static inline PyFrameObject* PyThreadState_GetFrame(PyThreadState *tstate)
{
    Py_XINCREF(tstate->frame);
    return tstate->frame;
}
#endif
```

Código definindo `PyThreadState_EnterTracing()` e `PyThreadState_LeaveTracing()` no Python 3.10 e anteriores:

```
#if PY_VERSION_HEX < 0x030B00A2
static inline void PyThreadState_EnterTracing(PyThreadState *tstate)
{
    tstate->tracing++;
#if PY_VERSION_HEX >= 0x030A00A1
    tstate->cframe->use_tracing = 0;
#else
```

(continua na próxima página)

(continuação da página anterior)

```
    tstate->use_tracing = 0;
#endif
}

static inline void PyThreadState_LeaveTracing(PyThreadState *tstate)
{
    int use_tracing = (tstate->c_tracefunc != NULL || tstate->c_profilefunc !=␣
↪NULL);
    tstate->tracing--;
#if PY_VERSION_HEX >= 0x030A00A1
    tstate->cframe->use_tracing = use_tracing;
#else
    tstate->use_tracing = use_tracing;
#endif
}
#endif
```

Ou use o projeto pythoncapi-compat para obter essas funções em funções antigas do Python.

- Os distribuidores são encorajados a compilar o Python com a biblioteca Blake2 otimizada libb2.
- O campo `PyConfig.module_search_paths_set` agora deve ser definido como 1 para inicialização para usar `PyConfig.module_search_paths` para inicializar `sys.path`. Caso contrário, a inicialização recalculará o caminho e substituirá quaisquer valores adicionados a `module_search_paths`.
- `PyConfig_Read()` não calcula mais o caminho de pesquisa inicial e não preencherá nenhum valor em `PyConfig.module_search_paths`. Para calcular caminhos padrão e modificá-los, termine a inicialização e use `PySys_GetObject()` para recuperar `sys.path` como um objeto lista do Python e modifique-o diretamente.

16.3 Descontinuados

- Descontinua as seguintes funções para configurar a inicialização do Python:
  - `PySys_AddWarnOptionUnicode()`
  - `PySys_AddWarnOption()`
  - `PySys_AddXOption()`
  - `PySys_HasWarnOptions()`
  - `PySys_SetArgvEx()`
  - `PySys_SetArgv()`
  - `PySys_SetPath()`
  - `Py_SetPath()`
  - `Py_SetProgramName()`
  - `Py_SetPythonHome()`
  - `Py_SetStandardStreamEncoding()`
  - `_Py_SetProgramFullPath()`

  Em vez disso, use a nova API `PyConfig` da Configuração de Inicialização do Python (**PEP 587**). (Contribuição de Victor Stinner em gh-88279.)

- Descontinua o membro `ob_shash` do `PyBytesObject`. Use `PyObject_Hash()` em vez disso. (Contribuição de Inada Naoki em bpo-46864.)

16.4 Pendente remoção no Python 3.12

As seguintes APIs C foram descontinuadas em versões anteriores do Python e serão removidas no Python 3.12.

- `PyUnicode_AS_DATA()`
- `PyUnicode_AS_UNICODE()`
- `PyUnicode_AsUnicodeAndSize()`
- `PyUnicode_AsUnicode()`
- `PyUnicode_FromUnicode()`
- `PyUnicode_GET_DATA_SIZE()`
- `PyUnicode_GET_SIZE()`
- `PyUnicode_GetSize()`
- `PyUnicode_IS_COMPACT()`
- `PyUnicode_IS_READY()`
- `PyUnicode_READY()`
- `PyUnicode_WSTR_LENGTH()`
- `_PyUnicode_AsUnicode()`
- `PyUnicode_WCHAR_KIND`
- `PyUnicodeObject`
- `PyUnicode_InternImmortal()`

16.5 Removidos

- `PyFrame_BlockSetup()` e `PyFrame_BlockPop()` foram removidas. (Contribuição de Mark Shannon em bpo-40222.)
- Remove as seguintes macros matemáticas usando a variável `errno`:
  - `Py_ADJUST_ERANGE1()`
  - `Py_ADJUST_ERANGE2()`
  - `Py_OVERFLOWED()`
  - `Py_SET_ERANGE_IF_OVERFLOW()`
  - `Py_SET_ERRNO_ON_MATH_ERROR()`

  (Contribuição de Victor Stinner em bpo-45412.)
- Remove as macros `Py_UNICODE_COPY()` e `Py_UNICODE_FILL()`, descontinuadas desde o Python 3.3. Use `PyUnicode_CopyCharacters()` ou `memcpy()` (string `wchar_t*`) e as funções `PyUnicode_Fill()`. (Contribuição de Victor Stinner em bpo-41123.)
- Remove o arquivo de cabeçalho `pystrhex.h`. Ele contém apenas funções privadas. As extensões C devem incluir apenas o arquivo principal `<Python.h>`. (Contribuição de Victor Stinner em bpo-45434.)

- Remove a macro `Py_FORCE_DOUBLE()`. Era usada pela macro `Py_IS_INFINITY()`. (Contribuição de Victor Stinner em bpo-45440.)
- Os seguintes itens não estão mais disponíveis quando `Py_LIMITED_API` é definida:
  - `PyMarshal_WriteLongToFile()`
  - `PyMarshal_WriteObjectToFile()`
  - `PyMarshal_ReadObjectFromString()`
  - `PyMarshal_WriteObjectToString()`
  - a macro `Py_MARSHAL_VERSION`

  Eles não fazem parte da API limitada.

  (Contribuição de Victor Stinner em bpo-45474.)
- Exclui `PyWeakref_GET_OBJECT()` da API C limitada. Nunca funcionou porque a estrutura `PyWeakReference` é opaca na API C limitada. (Contribuição de Victor Stinner em bpo-35134.)
- Remove a macro `PyHeapType_GET_MEMBERS()`. Estava exposta na API C pública por engano, deve ser usada apenas pelo Python internamente. Em vez disso, use o membro `PyTypeObject.tp_members`. (Contribuição de Victor Stinner em bpo-40170.)
- Remove a macro `HAVE_PY_SET_53BIT_PRECISION` (movida para a API C interna). (Contribuição de Victor Stinner em bpo-45412.)
- Remove as APIs de codificação `Py_UNICODE`, pois foram descontinuadas desde o Python 3.3, são pouco usadas e ineficientes em relação às alternativas recomendadas.

  As funções removidas são:
  - `PyUnicode_Encode()`
  - `PyUnicode_EncodeASCII()`
  - `PyUnicode_EncodeLatin1()`
  - `PyUnicode_EncodeUTF7()`
  - `PyUnicode_EncodeUTF8()`
  - `PyUnicode_EncodeUTF16()`
  - `PyUnicode_EncodeUTF32()`
  - `PyUnicode_EncodeUnicodeEscape()`
  - `PyUnicode_EncodeRawUnicodeEscape()`
  - `PyUnicode_EncodeCharmap()`
  - `PyUnicode_TranslateCharmap()`
  - `PyUnicode_EncodeDecimal()`
  - `PyUnicode_TransformDecimalToASCII()`

  Veja a **PEP 624** para detalhes e **orientação sobre migração**. (Contribuição de Inada Naoki em bpo-44029.)

17 Notable Changes in 3.11.4

17.1 tarfile

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

18 Notable Changes in 3.11.5

18.1 OpenSSL

- Construções do Windows e instaladores do macOS em python.org agora usam OpenSSL 3.0.

19 Notable changes in 3.11.10

19.1 ipaddress

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

19.2 email

- Cabeçalhos com novas linhas embutidas agora são colocados entre aspas na saída.

  O `generator` agora se recusará a serializar (escrever) cabeçalhos que são dobrados ou delimitados incorretamente, de modo que eles seriam analisados como vários cabeçalhos ou unidos a dados adjacentes. Se você precisar desativar esse recurso de segurança, defina `verify_generated_headers`. (Contribuição de Bas Bloemsaat e Petr Viktorin em gh-121650.)

- `email.utils.getaddresses()` and `email.utils.parseaddr()` now return `('', '')` 2-tuples in more situations where invalid email addresses are encountered, instead of potentially inaccurate values. An optional *strict* parameter was added to these two functions: use `strict=False` to get the old behavior, accepting malformed inputs. `getattr(email.utils, 'supports_strict_parsing', False)` can be used to check if the *strict* paramater is available. (Contributed by Thomas Dwyer and Victor Stinner for gh-102988 to improve the CVE-2023-27043 fix.)

Índice

P

V