What's New in Python

Release 3.8.18

A. M. Kuchling

fevereiro 08, 2024

Python Software Foundation
Email: docs@python.org

Sumário

Editor Raymond Hettinger

Este artigo explica os novos recursos no Python 3.8, em comparação com 3.7. Para detalhes completos, veja o changelog.

Python 3.8 foi lançado em 14 de outubro de 2019.

1 Resumo – Destaques da versão

2 Novas funcionalidades

2.1 Expressões de atribuição

Existe uma nova sintaxe `:=` que atribui valores a variáveis como parte de uma expressão maior. É carinhosamente conhecido como "o operador morsa" ("the walrus operator", em inglês) devido à sua semelhança com os olhos e presas de uma morsa.

Neste exemplo, a expressão de atribuição ajuda a evitar a chamada de `len()` duas vezes:

```
if (n := len(a)) > 10:
    print(f"List is too long ({n} elements, expected <= 10)")
```

Um benefício semelhante surge durante a correspondência de expressões regulares, onde os objetos de correspondência são necessários duas vezes, uma vez para testar se ocorreu uma correspondência e outra para extrair um subgrupo:

```
discount = 0.0
if (mo := re.search(r'(\d+)% discount', advertisement)):
    discount = float(mo.group(1)) / 100.0
```

O operador também é útil com loops while que calculam um valor para testar a finalização do loop e, em seguida, precisam desse mesmo valor novamente no corpo do loop:

```
# Loop over fixed length blocks
while (block := f.read(256)) != '':
    process(block)
```

Outro caso de uso motivador surge em compreensões de lista em que um valor calculado em uma condição de filtragem também é necessário no corpo da expressão:

```
[clean_name.title() for name in names
 if (clean_name := normalize('NFC', name)) in allowed_names]
```

Tente limitar o uso do operador da morsa para limpar os casos que reduzem a complexidade e melhoram a legibilidade.

Consulte **PEP 572** para uma descrição completa.

(Contribuído por Emily Morehouse em bpo-35224.)

2.2 Parâmetros apenas posicionais

Existe uma nova sintaxe de parâmetro de função `/` para indicar que alguns parâmetros de função devem ser especificados posicionalmente e não podem ser usados como argumentos nomeados. Esta é a mesma notação mostrada por `help()` para funções C anotadas com a ferramenta Argument Clinic de Larry Hastings.

No exemplo a seguir, os parâmetros *a* e *b* são apenas posicionais, enquanto *c* ou *d* podem ser posicionais ou nomeados e *e* ou *f* são obrigatoriamente nomeados:

```
def f(a, b, /, c, d, *, e, f):
    print(a, b, c, d, e, f)
```

A seguir, uma chamada válida:

```
f(10, 20, 30, d=40, e=50, f=60)
```

Porém, essas chamadas são inválidas:

```
f(10, b=20, c=30, d=40, e=50, f=60)   # b cannot be a keyword argument
f(10, 20, 30, 40, 50, f=60)           # e must be a keyword argument
```

Um caso de uso para essa notação é que ele permite que funções puras do Python emulem completamente os comportamentos das funções codificadas em C existentes. Por exemplo, a função embutida `divmod()` não aceita argumentos nomeados:

```
def divmod(a, b, /):
    "Emulate the built in divmod() function"
    return (a // b, a % b)
```

Outro caso de uso é impedir argumentos nomeados quando o nome do parâmetro não for útil. Por exemplo, a função embutida `len()` possui a assinatura `len(obj, /)`. Isso exclui chamadas estranhas, como:

```
len(obj='hello')  # The "obj" keyword argument impairs readability
```

Um benefício adicional de marcar um parâmetro como somente posicional é que ele permite que o nome do parâmetro seja alterado no futuro sem risco de quebrar o código do cliente. Por exemplo, no módulo `statistics`, o nome do parâmetro *dist* pode ser alterado no futuro. Isso foi possível com a seguinte especificação de função:

```
def quantiles(dist, /, *, n=4, method='exclusive')
    ...
```

Como os parâmetros à esquerda de `/` não são expostos como possíveis palavras-chave, os nomes dos parâmetros permanecem disponíveis para uso em `**kwargs`:

```
>>> def f(a, b, /, **kwargs):
...     print(a, b, kwargs)
...
```

(continua na próxima página)

(continuação da página anterior)

```
>>> f(10, 20, a=1, b=2, c=3)         # a and b are used in two ways
10 20 {'a': 1, 'b': 2, 'c': 3}
```

Isso simplifica bastante a implementação de funções e métodos que precisam aceitar argumentos nomeados arbitrários. Por exemplo, aqui está um trecho do código no módulo `collections`:

```
class Counter(dict):

    def __init__(self, iterable=None, /, **kwds):
        # Note "iterable" is a possible keyword argument
```

Consulte a **PEP 570** para uma descrição completa.

(Contribuição de Pablo Galindo em bpo-36540.)

2.3 Cache do sistema de arquivos paralelo para arquivos de bytecode compilados

A nova configuração `PYTHONPYCACHEPREFIX` (também disponível como `-X pycache_prefix`) configura o cache implícito de bytecode para usar uma árvore paralela separada do sistema de arquivos, em vez dos subdiretórios padrão `__pycache__` dentro cada diretório fonte.

O local do cache é relatado em `sys.pycache_prefix` (`None` indica o local padrão nos subdiretórios `__pycache__`).

(Contribuição de Carl Meyer em bpo-33499.)

2.4 A compilação de depuração usa a mesma ABI da compilação de lançamento

Agora, o Python usa a mesma ABI, seja ele compilado no modo de lançamento ou depuração. No Unix, quando o Python é compilado no modo de depuração, agora é possível carregar extensões C compiladas no modo de lançamento e extensões C compiladas usando a ABI estável.

Compilações de lançamento e compilações de depuração agora são compatíveis com ABI: definir a macro `Py_DEBUG` não implica mais a macro `Py_TRACE_REFS`, que introduz a única incompatibilidade ABI. A macro `Py_TRACE_REFS`, que adiciona a função `sys.getobjects()` e a variável de ambiente `PYTHONDUMPREFS`, pode ser configurada usando a nova opção de compilação `./configure --with-trace-refs`. (Contribuição de Victor Stinner em bpo-36465.)

No Unix, as extensões C não estão mais vinculadas ao libpython, exceto no Android e Cygwin. Agora é possível que um Python vinculado estaticamente carregue uma extensão C criada usando uma biblioteca compartilhada Python. (Contribuição de Victor Stinner em bpo-21536.)

No Unix, quando o Python é compilado no modo de depuração, a importação agora também procura extensões C compiladas no modo de lançamento e extensões C compiladas com a ABI estável. (Contribuição de Victor Stinner em bpo-36722.)

Para incorporar o Python em uma aplicação, uma nova opção `--embed` deve ser passada para `python3-config --libs --embed` para obter o `-lpython3.8` (vincula a aplicação ao libpython). Para dar suporte a 3.8 e versões mais antigas, tente `python3-config --libs --embed` primeiro e use como fallback para `python3-config --libs` (sem `--embed`) se o comando anterior falhar.

Adicione um módulo pkg-config `python-3.8-embed` para incorporar o Python em uma aplicação: `pkg-config python-3.8-embed --libs` inclui `-lpython3.8`. Para suportar tanto o 3.8 quanto o anterior, tente `pkg-config python-X.Y-embed --libs` primeiro e use como fallback `pkg-config python-X.Y --libs` (sem `--embed`) se o comando anterior falhar (substitua `X.Y` pela versão do Python).

Por outro lado, `pkg-config python3.8 --libs` não contém mais `-lpython3.8`. As extensões C não devem estar vinculadas ao libpython (exceto no Android e Cygwin, cujos casos são tratados pelo script); essa alteração é retroativamente incompatível de propósito. (Contribuição de Victor Stinner em bpo-36721.)

2.5 O suporte a f-strings = para expressões de auto-documentação e depuração

Adicionado um especificador = para f-strings. Uma f-string como `f'{expr=}'` será expandida para o texto da expressão, um sinal de igual e, em seguida, a representação da expressão avaliada. Por exemplo:

```
>>> user = 'eric_idle'
>>> member_since = date(1975, 7, 31)
>>> f'{user=} {member_since=}'
"user='eric_idle' member_since=datetime.date(1975, 7, 31)"
```

O habitual f-string especificadores de formato permite mais controle sobre como o resultado da expressão é exibido:

```
>>> delta = date.today() - member_since
>>> f'{user=!s}  {delta.days=:,d}'
'user=eric_idle  delta.days=16,075'
```

O especificador = exibirá toda a expressão para que os cálculos possam ser mostrados:

```
>>> print(f'{theta=}  {cos(radians(theta))=:.3f}')
theta=30  cos(radians(theta))=0.866
```

(Contribuição de Eric V. Smith e Larry Hastings em bpo-36817.)

2.6 PEP 578: Ganchos de auditoria em tempo de execução Python

O PEP adiciona um Gancho de Auditoria e um Gancho Aberto Verificado. Ambos estão disponíveis no Python e no código nativo, permitindo que aplicações e estruturas escritas em código Python puro aproveitem as notificações extras, além de permitir que incorporadores ou administradores de sistema implementem versões do Python onde a auditoria está sempre ativada.

Veja **PEP 578** para mais detalhes.

2.7 PEP 587: Configuração de Inicialização do Python

O: pep:*587* adiciona uma nova API C para configurar a Inicialização do Python, fornecendo um controle mais preciso de toda a configuração e melhores relatórios de erros.

Novas estruturas:

- `PyConfig`
- `PyPreConfig`
- `PyStatus`
- `PyWideStringList`

Novas Funções:

- `PyConfig_Clear()`
- `PyConfig_InitIsolatedConfig()`

- `PyConfig_InitPythonConfig()`
- `PyConfig_Read()`
- `PyConfig_SetArgv()`
- `PyConfig_SetBytesArgv()`
- `PyConfig_SetBytesString()`
- `PyConfig_SetString()`
- `PyPreConfig_InitIsolatedConfig()`
- `PyPreConfig_InitPythonConfig()`
- `PyStatus_Error()`
- `PyStatus_Exception()`
- `PyStatus_Exit()`
- `PyStatus_IsError()`
- `PyStatus_IsExit()`
- `PyStatus_NoMemory()`
- `PyStatus_Ok()`
- `PyWideStringList_Append()`
- `PyWideStringList_Insert()`
- `Py_BytesMain()`
- `Py_ExitStatusException()`
- `Py_InitializeFromConfig()`
- `Py_PreInitialize()`
- `Py_PreInitializeFromArgs()`
- `Py_PreInitializeFromBytesArgs()`
- `Py_RunMain()`

Esse PEP também adiciona os campos `_PyRuntimeState.preconfig` (tipo `PyPreConfig`) e"PyInterpreterState.config" (tipo `PyConfig`) campos a essas estruturas internas. `PyInterpreterState.config` se torna a nova configuração de referência, substituindo variáveis de configuração global e outras variáveis privadas.

Veja Configuração de Inicialização do Python para a documentação.

Consulte a **PEP 587** para uma descrição completa.

(Contribuição de Victor Stinner em bpo-36763.)

2.8 Vectorcall: um protocolo de chamada rápida para CPython

O protocolo "vectorcall" é adicionado à API Python/C. Ele visa formalizar as otimizações existentes que já foram feitas para várias classes. Qualquer tipo de extensão que implemente um chamável pode usar esse protocolo.

Atualmente, isso é provisório. O objetivo é torná-lo totalmente público no Python 3.9.

Consulte a **PEP 590** para uma descrição completa.

(Contribuição de Jeroen Demeyer e Mark Shannon em :issue: *36974*.)

2.9 Protocolo de Pickle 5 com buffers de dados fora da banda

Quando `pickle` é usado para transferir grande quantidade de dados entre processos Python, a fim de tirar proveito do processamento com vários núcleos ou com várias máquinas, é importante otimizar a transferência reduzindo cópias de memória e, possivelmente, aplicando técnicas personalizadas, como como compactação dependente de dados.

O protocolo `pickle` 5 introduz suporte para buffers fora de banda, onde :dados compatíveis com pep:*3118* podem ser transmitidos separadamente do fluxo de pickle principal, a critério da camada de comunicação.

Consulte a **PEP 574** para uma descrição completa.

(Contribuição de Antoine Pitrou em bpo-36785.)

3 Outras mudanças na linguagem

- A instrução `continue` era ilegal na cláusula `finally` devido a um problema com a implementação. No Python 3.8, essa restrição foi levantada. (Contribuição de Serhiy Storchaka em bpo-32489.)
- Os tipos `bool`, `int` e `fractions.Fraction` agora têm o método `as_integer_ratio()` como o encontrado em `float` e `decimal.Decimal`. Esta pequena extensão da API torna possível escrever `numerator, denominator = x.as_integer_ratio()` e fazer com que funcione em vários tipos numéricos. (Contribuição de Lisa Roach em bpo-33073 e Raymond Hettinger em bpo-37819.)
- Os construtores de `int`, `float` e `complex` agora usarão o método especial `__index__()`, se disponível, e o método correspondente `__int__()`, `__float__()` ou `__complex__()` não está disponível. (Contribuição de Serhiy Storchaka em bpo-20092.)
- Adicionado suporte a escapes `\N{name}` em `expressões regulares`:

```
>>> notice = 'Copyright © 2019'
>>> copyright_year_pattern = re.compile(r'\N{copyright sign}\s*(\d{4})')
>>> int(copyright_year_pattern.search(notice).group(1))
2019
```

  (Contribuição de Jonathan Eunice e Serhiy Storchaka em bpo-30688.)
- Dict e dictviews agora são iteráveis em ordem de inserção reversa usando `reversed()`. (Contribuição de Rémi Lapeyre em bpo-33462.)
- A sintaxe permitida para nomes de palavras-chave em chamadas de função foi ainda mais restrita. Em particular, `f((keyword)=arg)` não é mais permitido. Ele nunca teve a intenção de permitir mais do que um nome simples no lado esquerdo de um termo de atribuição de argumento nomeado. (Contribuição de Benjamin Peterson em bpo-34641.)
- A descompactação iterável generalizada das instruções `yield` e `return` não requer mais parênteses. Isso traz a sintaxe *yield* e *return* uma melhor concordância com a sintaxe de atribuição normal:

```
>>> def parse(family):
        lastname, *members = family.split()
        return lastname.upper(), *members

>>> parse('simpsons homer marge bart lisa sally')
('SIMPSONS', 'homer', 'marge', 'bart', 'lisa', 'sally')
```

(Contribuição de David Cuthbert e Jordan Chapman em bpo-32117.)

- Quando uma vírgula está faltando em código como `[(10, 20) (30, 40)]`, o compilador exibe um `SyntaxWarning` com uma sugestão útil. Isso melhora apenas tendo um `TypeError` indicando que a primeira tupla não era chamável. (Contribuição de Serhiy Storchaka em bpo-15248.)
- As operações aritméticas entre as subclasses de `datetime.date` ou `datetime.datetime` e `datetime.timedelta` agora retornam uma instância da subclasse, em vez da classe base. Isso também afeta o tipo de retorno de operações cuja implementação (direta ou indiretamente) utiliza aritmética de `datetime.timedelta`, como `astimezone()`. (Contribuição de Paul Ganssle em bpo-32417.)
- Quando o interpretador Python é interrompido por Ctrl-C (SIGINT) e a exceção resultante `KeyboardInterrupt` não é capturada, o processo Python agora sai por meio de um sinal SIGINT ou com o código de saída correto, para que o processo de chamada possa detectar que foi encerrado devido a um Ctrl-C. Os shells no POSIX e Windows usam isso para finalizar corretamente os scripts em sessões interativas. (Contribuição do Google via Gregory P. Smith em bpo-1054041.)
- Alguns estilos avançados de programação requerem a atualização do objeto `types.CodeType` para uma função existente. Como os objetos código são imutáveis, um novo objeto código precisa ser criado, modelado no objeto código existente. Com 19 parâmetros, isso foi um pouco tedioso. Agora, o novo método `replace()` torna possível criar um clone com alguns parâmetros alterados.

  Aqui está um exemplo que altera a função `Statistics.mean()` para impedir que o parâmetro *data* seja usado como argumento nomeado:

```
>>> from statistics import mean
>>> mean(data=[10, 20, 90])
40
>>> mean.__code__ = mean.__code__.replace(co_posonlyargcount=1)
>>> mean(data=[10, 20, 90])
Traceback (most recent call last):
  ...
TypeError: mean() got some positional-only arguments passed as keyword arguments:
↪'data'
```

(Contribuição de Victor Stinner em bpo-37032.)

- Para números inteiros, a forma de três argumentos da função `pow()` agora permite que o expoente seja negativo no caso em que a base é relativamente primária ao módulo. Em seguida, calcula um inverso modular para a base quando o expoente é `-1` e uma potência adequada desse inverso para outros expoentes negativos. Por exemplo, para calcular o inverso multiplicativo modular de 38 módulo 137, escreva:

```
>>> pow(38, -1, 137)
119
>>> 119 * 38 % 137
1
```

Inversos modulares surgem na solução de equações diofantinas lineares. Por exemplo, para encontrar soluções inteiras para `4258𝑥 + 147𝑦 = 369`, primeiro reescreva como `4258𝑥 ≡ 369 (mod 147)` e depois resolva:

```
>>> x = 369 * pow(4258, -1, 147) % 147
>>> y = (4258 * x - 369) // -147
>>> 4258 * x + 147 * y
369
```

(Contribuição de Mark Dickinson em bpo-36027.)

- As compreensões de dict foram sincronizadas com literais de dict, para que a chave seja computada primeiro e o valor em segundo:

```
>>> # Dict comprehension
>>> cast = {input('role? '): input('actor? ') for i in range(2)}
role? King Arthur
actor? Chapman
role? Black Knight
actor? Cleese

>>> # Dict literal
>>> cast = {input('role? '): input('actor? ')}
role? Sir Robin
actor? Eric Idle
```

A ordem de execução garantida é útil nas expressões de atribuição, porque as variáveis atribuídas na expressão de chave estarão disponíveis na expressão de valor:

```
>>> names = ['Martin von Löwis', 'Łukasz Langa', 'Walter Dörwald']
>>> {(n := normalize('NFC', name)).casefold() : n for name in names}
{'martin von löwis': 'Martin von Löwis',
 'łukasz langa': 'Łukasz Langa',
 'walter dörwald': 'Walter Dörwald'}
```

(Contribuição de Jörn Heissler em bpo-35224.)

- O método `object.__reduce__()` agora pode retornar uma tupla de dois a seis elementos. Anteriormente, cinco era o limite. O novo sexto elemento opcional é um chamável com uma assinatura `(obj, state)`. Isso permite o controle direto sobre o comportamento de atualização de estado de um objeto específico. Caso contrário, *None*, esse chamável terá prioridade sobre o método `__setstate__()` do objeto. (Contribuição de Pierre Glaser e Olivier Grisel em bpo-35900.)

4 Novos módulos

- O novo módulo `importlib.metadata` fornece suporte (provisório) para a leitura de metadados de pacotes de terceiros. Por exemplo, ele pode extrair o número da versão de um pacote instalado, a lista de pontos de entrada e muito mais:

```
>>> # Note following example requires that the popular "requests"
>>> # package has been installed.
>>>
>>> from importlib.metadata import version, requires, files
>>> version('requests')
'2.22.0'
>>> list(requires('requests'))
['chardet (<3.1.0,>=3.0.2)']
>>> list(files('requests'))[:5]
[PackagePath('requests-2.22.0.dist-info/INSTALLER'),
```

(continua na próxima página)

(continuação da página anterior)

```
PackagePath('requests-2.22.0.dist-info/LICENSE'),
PackagePath('requests-2.22.0.dist-info/METADATA'),
PackagePath('requests-2.22.0.dist-info/RECORD'),
PackagePath('requests-2.22.0.dist-info/WHEEL')]
```

(Contribuição de Barry Warsaw e Jason R. Coombs em bpo-34632.)

5 Módulos melhorados

5.1 ast

Os nós de AST agora têm os atributos `end_lineno` e `end_col_offset`, que fornecem a localização precisa do final do nó. (Isso se aplica apenas aos nós que possuem os atributos `lineno` e `col_offset`.)

Nova função `ast.get_source_segment()` retorna o código-fonte para um nó AST específico.

(Contribuição de Ivan Levkivskyi em bpo-33416.)

A função `ast.parse()` possui alguns novos sinalizadores:

- `type_comments=True` faz com que retorne o texto dos comentários de tipos de **PEP 484** e **PEP 526** associados a determinados nós AST;
- `mode='func_type'` pode ser usado para analisar "comentários do tipo de assinatura" do **PEP 484** (retornados para nós AST de definição de função);
- `feature_version=(3, N)` permite especificar uma versão anterior do Python 3. Por exemplo, `feature_version=(3, 4)` tratará `async` e `await` como palavras não reservadas.

(Contribuição de Guido van Rossum em bpo-35766.)

5.2 asyncio

`asyncio.run()` passou da API provisória para a estável. Esta função pode ser usada para executar uma corrotina e retornar o resultado enquanto gerencia automaticamente o loop de eventos. Por exemplo:

```
import asyncio

async def main():
    await asyncio.sleep(0)
    return 42

asyncio.run(main())
```

Isso é *aproximadamente* equivalente a:

```
import asyncio

async def main():
    await asyncio.sleep(0)
    return 42

loop = asyncio.new_event_loop()
asyncio.set_event_loop(loop)
```

(continua na próxima página)

(continuação da página anterior)

```
try:
    loop.run_until_complete(main())
finally:
    asyncio.set_event_loop(None)
    loop.close()
```

A implementação real é significativamente mais complexa. Assim, `asyncio.run()` deve ser a maneira preferida de executar programas assíncronos.

(Contribuição de Yury Selivanov em bpo-32314.)

A execução de `python -m asyncio` lança um REPL nativo assíncrono. Isso permite a experimentação rápida de código que possui um nível superior `await`. Não é mais necessário chamar diretamente `asyncio.run()`, o que geraria um novo loop de evento a cada chamada:

```
$ python -m asyncio
asyncio REPL 3.8.0
Use "await" directly instead of "asyncio.run()".
Type "help", "copyright", "credits" or "license" for more information.
>>> import asyncio
>>> await asyncio.sleep(10, result='hello')
hello
```

(Contribuição de Yury Selivanov em bpo-37028.)

The exception `asyncio.CancelledError` now inherits from `BaseException` rather than `Exception` and no longer inherits from `concurrent.futures.CancelledError`. (Contributed by Yury Selivanov in bpo-32528.)

No Windows, o loop de eventos padrão é agora `ProactorEventLoop`. (Contribuição de Victor Stinner em bpo-34687.)

`ProactorEventLoop` agora também tem suporte a UDP. (Contribuição de Adam Meily e Andrew Svetlov em bpo-29883.)

`ProactorEventLoop` pode agora ser interrompido por `KeyboardInterrupt` ("CTRL+C"). (Contribuição de Vladimir Matveev em bpo-23057.)

Adicionado `asyncio.Task.get_coro()` para obter a corrotina de dentro de um `asyncio.Task`. (Contribuição de Alex Grönholm em bpo-36999.)

As tarefas de asyncio agora podem ser nomeadas, passando o argumento nomeado `name` para `asyncio.create_task()` ou o método de loop de evento `create_task()` ou chamando o método `set_name()` no objeto de tarefa. O nome da tarefa é visível na saída de `repr()` de `asyncio.Task` e também pode ser recuperado usando o método `get_name()`. (Contribuição de Alex Grönholm em bpo-34270.)

Adicionado suporte a Happy Eyeballs para `asyncio.loop.create_connection()`. Para especificar o comportamento, dois novos parâmetros foram adicionados: *happy_eyeballs_delay* e *interleave*. O algoritmo Happy Eyeballs melhora a capacidade de resposta em aplicativos compatíveis com IPv4 e IPv6, tentando conectar-se simultaneamente usando os dois. (Contribuição de twisteroid ambassador em bpo-33530.)

5.3 builtins

A função embutida `compile()` foi aprimorada para aceitar o sinalizador `ast.PyCF_ALLOW_TOP_LEVEL_AWAIT`. Com esse novo sinalizador passado, `compile()` permitirá construções de nível superior `await`, `async for` e `async with` que geralmente são consideradas sintaxe inválida. O objeto de código assíncrono marcado com o sinalizador `CO_COROUTINE` pode ser retornado. (Contribuição de Matthias Bussonnier em bpo-34616)

5.4 collections

O método `_asdict()` para `collections.namedtuple()` agora retorna a `dict` em vez de um `collections.OrderedDict`. Isso funciona porque os dict regulares garantem pedidos desde o Python 3.7. Se os recursos extras de `OrderedDict` forem necessários, a solução sugerida é converter o resultado no tipo desejado: `OrderedDict(nt._asdict())`. (Contribuição de Raymond Hettinger em bpo-35864.)

5.5 cProfile

A classe `cProfile.Profile` agora pode ser usada como um gerenciador de contexto. Perfile um bloco de código executando:

```
import cProfile

with cProfile.Profile() as profiler:
      # code to be profiled
      ...
```

(Contribuição de Scott Sanderson em bpo-29235.)

5.6 csv

O `csv.DictReader` agora retorna instâncias de `dict` em vez de um `collections.OrderedDict`. Agora, a ferramenta é mais rápida e utiliza menos memória enquanto preserva a ordem dos campos. (Contribuição de Michael Selik em bpo-34003.)

5.7 curses

Adicionada uma nova variável contendo informações de versão estruturada para a biblioteca ncurses subjacente: `ncurses_version`. (Contribuição de Serhiy Storchaka em bpo-31680.)

5.8 ctypes

No Windows, `CDLL` e subclasses agora aceitam um parâmetro *winmode* para especificar sinalizadores para a chamada subjacente `LoadLibraryEx`. Os sinalizadores padrão são definidos para carregar apenas dependências da DLL em locais confiáveis, incluindo o caminho em que a DLL está armazenada (se um caminho completo ou parcial for usado para carregar a DLL inicial) e os caminhos adicionados por `add_dll_directory()`. (Contribuição de Steve Dower em bpo-36085.)

5.9 datetime

Adicionados novos construtores alternativos `datetime.date.fromisocalendar()` e `datetime.datetime.fromisocalendar()`, que constroem objetos de `date` e `datetime` respectivamente do ano ISO, número da semana e dia da semana; estes são o inverso do método `isocalendar` de cada classe. (Contribuição de Paul Ganssle em bpo-36004.)

5.10 functools

`functools.lru_cache()` agora pode ser usado como um decorador direto, e não como uma função retornando um decorador. Portanto, agora ambos são suportados:

```
@lru_cache
def f(x):
    ...

@lru_cache(maxsize=256)
def f(x):
    ...
```

(Contribuição de Raymond Hettinger em bpo-36772.)

Adicionado um novo decorador `functools.cached_property()`, para propriedades computadas armazenadas em cache durante toda a vida útil da instância.

```
import functools
import statistics

class Dataset:
   def __init__(self, sequence_of_numbers):
      self.data = sequence_of_numbers

   @functools.cached_property
   def variance(self):
      return statistics.variance(self.data)
```

(Contribuição de Carl Meyer em bpo-21145)

Adicionado um novo decorador `functools.singledispatchmethod()` que converte métodos em funções genéricas usando despacho único:

```
from functools import singledispatchmethod
from contextlib import suppress

class TaskManager:

    def __init__(self, tasks):
        self.tasks = list(tasks)

    @singledispatchmethod
    def discard(self, value):
        with suppress(ValueError):
            self.tasks.remove(value)

    @discard.register(list)
    def _(self, tasks):
```

(continua na próxima página)

(continuação da página anterior)

```
        targets = set(tasks)
        self.tasks = [x for x in self.tasks if x not in targets]
```

(Contribuição de Ethan Smith em bpo-32380)

5.11 gc

`get_objects()` agora pode receber um parâmetro opcional *generation* indicando uma geração da qual obter objetos. (Contribuição de Pablo Galindo em bpo-36016.)

5.12 gettext

Adicionado `pgettext()` e suas variantes. (Contribuição de Franz Glasner, Éric Araujo e Cheryl Sabella em bpo-2504.)

5.13 gzip

Adicionado o parâmetro *mtime* em `gzip.compress()` para saída reproduzível. (Contribuição de Guo Ci Teo em bpo-34898.)

A exceção `BadGzipFile` agora é levantada em vez de `OSError` para certos tipos de arquivos gzip inválidos ou corrompidos. (Contribuição de Filip Gruszczyński, Michele Orrù e Zackery Spytz em bpo-6584.)

5.14 IDLE e idlelib

A saída em N linhas (50 por padrão) é compactada até um limite inferior. N pode ser alterado na seção PyShell da página General da caixa de diálogo Settings. Menos linhas, mas possivelmente extra longas, podem ser espremidas clicando com o botão direito do mouse na saída. A saída compactada pode ser expandida no local clicando duas vezes no botão ou na área de transferência ou em uma janela separada clicando com o botão direito do mouse. (Contribuição de Tal Einat em bpo-1529353.)

Adiciona "Run Customized" ao menu Executar para executar um módulo com configurações personalizadas. Quaisquer argumentos de linha de comando inseridos são adicionados ao sys.argv. Eles também reaparecem na caixa para a próxima execução personalizada. Pode-se também suprimir a reinicialização normal do módulo principal do console. (Contribuição de Cheryl Sabella, Terry Jan Reedy e outros em bpo-5680 e bpo-37627.)

Adicionados números de linha opcionais para janelas do editor IDLE. O Windows é aberto sem números de linha, a menos que definido de outra forma na guia General da caixa de diálogo de configuração. Os números de linha de uma janela existente são mostrados e ocultos no menu Opções. (Contribuição de Tal Einat e Saimadhav Heblikar em bpo-17535.)

A codificação nativa do sistema operacional agora é usada para converter entre strings Python e objetos Tcl. Isso permite que o IDLE trabalhe com emoji e outros caracteres não BMP. Esses caracteres podem ser exibidos ou copiados e colados na ou da área de transferência. A conversão de strings de Tcl para Python e de volta agora nunca falha. (Muitas pessoas trabalharam nisso por oito anos, mas o problema foi finalmente resolvido por Serhiy Storchaka em bpo-13153.)

New in 3.8.1:

Add option to toggle cursor blink off. (Contributed by Zackery Spytz in bpo-4603.)

Escape key now closes IDLE completion windows. (Contributed by Johnny Najera in bpo-38944.)

As alterações acima foram suportadas para versões de manutenção 3.7.

Add keywords to module name completion list. (Contributed by Terry J. Reedy in bpo-37765.)

5.15 inspect

A função `inspect.getdoc()` agora pode encontrar docstrings para `__slots__` se esse atributo for `dict` onde os valores são docstrings. Isso fornece opções de documentação semelhantes às que já temos para `property()`, `classmethod()` e `staticmethod()`:

```
class AudioClip:
    __slots__ = {'bit_rate': 'expressed in kilohertz to one decimal place',
                 'duration': 'in seconds, rounded up to an integer'}
    def __init__(self, bit_rate, duration):
        self.bit_rate = round(bit_rate / 1000.0, 1)
        self.duration = ceil(duration)
```

(Contribuição de Raymond Hettinger em bpo-36326.)

5.16 io

No modo de desenvolvimento (`-X env`) e na compilação de depuração, o finalizador `io.IOBase` agora registra a exceção se o método `close()` falhar. A exceção é ignorada silenciosamente por padrão na compilação de lançamento. (Contribuição de Victor Stinner em bpo-18748.)

5.17 itertools

A função `itertools.accumulate()` adicionou uma opção de argumento nomeado *initial* para especificar um valor inicial:

```
>>> from itertools import accumulate
>>> list(accumulate([10, 5, 30, 15], initial=1000))
[1000, 1010, 1015, 1045, 1060]
```

(Contribuído por Lisa Roach em bpo-34659.)

5.18 json.tool

Adiciona a opção `--json-lines` para analisar cada linha de entrada como um objeto JSON separado. (Contribuição de Weipeng Hong em bpo-31553.)

5.19 logging

Adicionado um argumento nomeado *force* em `logging.basicConfig()` Quando definido como true, qualquer manipulador existente anexado ao logger raiz é removido e fechado antes de executar a configuração especificada pelos outros argumentos.

Isso resolve um problema de longa data. Depois que um logger ou *basicConfig()* foi chamado, as chamadas subsequentes para *basicConfig()* foram ignoradas silenciosamente. Isso dificultava a atualização, a experimentação ou o ensino das várias opções de configuração de log usando o prompt interativo ou um bloco de notas de Jupyter.

(Sugestão de Raymond Hettinger, implementação de Dong-hee Na e revisão de Vinay Sajip em bpo-33897.)

5.20 math

Adicionada nova função `math.dist()` para calcular a distância euclidiana entre dois pontos. (Contribuição de Raymond Hettinger em bpo-33089.)

Expandida a função `math.hypot()` para manipular várias dimensões. Anteriormente, suportava apenas o caso 2-D. (Contribuição de Raymond Hettinger em bpo-33089.)

Adicionada nova função, `math.prod()`, como função análoga a `sum()` que retorna o produto de um valor 'start' (padrão: 1) vezes uma iterável de números:

```
>>> prior = 0.8
>>> likelihoods = [0.625, 0.84, 0.30]
>>> math.prod(likelihoods, start=prior)
0.126
```

(Contribuição de Pablo Galindo em bpo-35606.)

Adicionadas duas novas funções combinatórias `math.perm()` e `math.comb()`:

```
>>> math.perm(10, 3)     # Permutations of 10 things taken 3 at a time
720
>>> math.comb(10, 3)     # Combinations of 10 things taken 3 at a time
120
```

(Contribuição de Yash Aggarwal, Keller Fuchs, Serhiy Storchaka e Raymond Hettinger em bpo-37128, bpo-37178 e bpo-35431.)

Adicionada uma nova função `math.isqrt()` para calcular raízes quadradas inteiras precisas sem conversão para ponto flutuante. A nova função suporta números inteiros arbitrariamente grandes. É mais rápido que `floor(sqrt(n))`, mas mais lento que `math.sqrt()`:

```
>>> r = 650320427
>>> s = r ** 2
>>> isqrt(s - 1)          # correct
650320426
>>> floor(sqrt(s - 1))    # incorrect
650320427
```

(Contribuição de Mark Dickinson em bpo-36887.)

A função `math.factorial()` não aceita mais argumentos que não são parecidos com int. (Contribuição de Pablo Galindo em bpo-33083.)

5.21 mmap

A classe `mmap.mmap` agora possui o método `madvise()` para acessar a chamada do sistema `madvise()`. (Contribuição de Zackery Spytz em bpo-32941.)

5.22 multiprocessing

Adicionado novo módulo `multiprocessing.shared_memory`. (Contribuição de Davin Potts em bpo-35813.)

No macOS, o método inicial *spawn* agora é usado por padrão. (Contribuição de Victor Stinner em bpo-33725.)

5.23 os

Adicionada nova função `add_dll_directory()` no Windows para fornecer caminhos de pesquisa adicionais para dependências nativas ao importar módulos de extensão ou carregar DLLs usando `ctypes`. (Contribuição de Steve Dower em bpo-36085.)

Uma nova função `os.memfd_create()` foi adicionada para envolver a chamada de sistema `memfd_create()`. (Contribuição de Zackery Spytz e Christian Heimes em bpo-26836.)

No Windows, grande parte da lógica manual para manipular pontos de nova análise (incluindo links simbólicos e junções de diretório) foi delegada ao sistema operacional. Especificamente, `os.stat()` agora percorrerá qualquer coisa suportada pelo sistema operacional, enquanto `os.lstat()` abrirá apenas pontos de nova análise que se identificam como "substitutos de nome" enquanto outros são abertos como `os.stat()`. Em todos os casos, `stat_result.st_mode` terá apenas `S_IFLNK` definido para links simbólicos e não para outros tipos de pontos de nova análise. Para identificar outros tipos de pontos de nova análise, verifique o novo atributo `stat_result.st_reparse_tag`.

No Windows, `os.readlink()` agora pode ler junções de diretório. Observe que `islink()` retornará `False` para junções de diretório e, portanto, o código que verifica primeiro `islink` continuará tratando junções como diretórios, enquanto o código que lida com erros de `os.readlink()` agora pode tratar junções como links.

(Contribuição de Steve Dower em bpo-37834.)

5.24 os.path

Funções de `os.path` que retornam um resultado booleano como `exists()`, `lexists()`, `isdir()`, `isfile()`, `islink()` e `ismount()` agora retornam `False` em vez de levantar `ValueError` ou suas subclasses `UnicodeEncodeError` e `UnicodeDecodeError` para caminhos que contêm caracteres ou bytes não representáveis no nível do SO. (Contribuição de Serhiy Storchaka em bpo-33721.)

`expanduser()` no Windows agora prefere a variável de ambiente `USERPROFILE` e não usa `HOME`, que normalmente não é definido para contas de usuário comuns. (Contribuição de Anthony Sottile em bpo-36264.)

`isdir()` no Windows não retorna mais `True` para um link para um diretório inexistente.

`realpath()` no Windows agora resolve pontos de nova análise, incluindo links simbólicos e junções de diretório.

(Contribuição de Steve Dower em bpo-37834.)

5.25 pathlib

`pathlib.Path` métodos que retornam um resultado booleano como `exists()`, `is_dir()`, `is_file()`, `is_mount()`, `is_symlink()`, `is_block_device()`, `is_char_device()`, `is_fifo()`, `is_socket()` agora retorna `False` ao invés de levantar `ValueError` ou sua subclasse `UnicodeEncodeError` para caminhos que contêm caracteres não representáveis no nível do SO. (Contribuição de Serhiy Storchaka em bpo-33721.)

Adicionado `pathlib.Path.link_to()` que cria um link rígido apontando para um caminho. (Contribuição de Joannah Nanjekye in bpo-26978)

5.26 pickle

Extensões de `pickle`, que atuem como subclasse da classe otimizada para C `Pickler`, agora podem substituir a lógica de "pickling" de funções e classes definindo o método especial `reducer_override()`. (Contribuição de Pierre Glaser e Olivier Grisel em bpo-35900.)

5.27 plistlib

Adicionado novo `plistlib.UID` e habilitado suporte para leitura e escrita de plists binárias codificadas em NSKeyedArchiver. (Contribuição de Jon Janzen em bpo-26707.)

5.28 pprint

O módulo `pprint` adicionou um parâmetro *sort_dicts* a várias funções. Por padrão, essas funções continuam a classificar dicionários antes da renderização ou impressão. No entanto, se *sort_dicts* estiver definido como false, os dicionários manterão a ordem em que as chaves foram inseridas. Isso pode ser útil para comparação com entradas JSON durante a depuração.

Além disso, existe uma nova função de conveniência, `pprint.pp()`, que é como `pprint.pprint()`, mas com *sort_dicts* padronizando como `False`:

```
>>> from pprint import pprint, pp
>>> d = dict(source='input.txt', operation='filter', destination='output.txt')
>>> pp(d, width=40)                  # Original order
{'source': 'input.txt',
 'operation': 'filter',
 'destination': 'output.txt'}
>>> pprint(d, width=40)              # Keys sorted alphabetically
{'destination': 'output.txt',
 'operation': 'filter',
 'source': 'input.txt'}
```

(Contribuição de Rémi Lapeyre em bpo-30670.)

5.29 py_compile

`py_compile.compile()` agora possui suporte a um modo silencioso. (Contribuição de Joannah Nanjekye em bpo-22640.)

5.30 shlex

A nova função `shlex.join()` atua como o inverso da `shlex.split()`. (Contribuição de Bo Bayles em bpo-32102.)

5.31 shutil

`shutil.copytree()` agora aceita um novo argumento nomeado `dirs_exist_ok`. (Contribuição de Josh Bronson em bpo-20849.)

`shutil.make_archive()` agora tem como pádrão o formato pax moderno (POSIX.1-2001) de novos arquivos para melhorar a portabilidade e a conformidade com os padrões, herdados da alteração correspondente no módulo `tarfile`. (Contribuição de C.A.M. Gerlach em bpo-30661.)

`shutil.rmtree()` no Windows agora remove junções de diretórios sem remover recursivamente o conteúdo primeiro. (Contribuição de Steve Dower em bpo-37834.)

5.32 socket

Adicionadas as funções de conveniência `create_server()` e `has_dualstack_ipv6()` para automatizar as tarefas necessárias geralmente envolvidas na criação de um soquete de servidor, incluindo a aceitação de conexões IPv4 e IPv6 no mesmo soquete. (Contribuição de Giampaolo Rodolà em bpo-17561.)

As funções `socket.if_nameindex()`, `socket.if_nametoindex()` e `socket.if_indextoname()` foram agora implementadas no Windows. (Contribuição de Zackery Spytz em bpo-37007.)

5.33 ssl

Adicionado `post_handshake_auth` para ativar e `verify_client_post_handshake()` para iniciar a autenticação pós-handshake do TLS 1.3. (Contribuição de Christian Heimes em bpo-34670.)

5.34 statistics

Adicionada `statistics.fmean()` como uma variante de ponto flutuante mais rápida de `statistics.mean()`. (Contribuição de Raymond Hettinger e Steven D'Aprano em bpo-35904.)

Adicionada `statistics.geometric_mean()` (Contribuição de Raymond Hettinger em bpo-27181.)

Adicionada `Statistics.multimode()` que retorna uma lista dos valores mais comuns. (Contribuição de Raymond Hettinger em bpo-35892.)

Adicionada `statistics.quantiles()` que divide dados ou uma distribuição em intervalos equivalentes (por exemplo, quartis, decis ou percentis). (Contribuição de Raymond Hettinger em bpo-36546.)

Adicionada `statistics.NormalDist`, uma ferramenta para criar e manipular distribuições normais de uma variável aleatória. (Contribuição de Raymond Hettinger em bpo-36018.)

```
>>> temperature_feb = NormalDist.from_samples([4, 12, -3, 2, 7, 14])
>>> temperature_feb.mean
6.0
>>> temperature_feb.stdev
6.356099432828281

>>> temperature_feb.cdf(3)           # Chance of being under 3 degrees
0.3184678262814532
>>> # Relative chance of being 7 degrees versus 10 degrees
>>> temperature_feb.pdf(7) / temperature_feb.pdf(10)
1.2039930378537762

>>> el_niño = NormalDist(4, 2.5)
```

(continua na próxima página)

(continuação da página anterior)

```
>>> temperature_feb += el_niño        # Add in a climate effect
>>> temperature_feb
NormalDist(mu=10.0, sigma=6.830080526611674)

>>> temperature_feb * (9/5) + 32      # Convert to Fahrenheit
NormalDist(mu=50.0, sigma=12.294144947901014)
>>> temperature_feb.samples(3)        # Generate random samples
[7.672102882379219, 12.000027119750287, 4.647488369766392]
```

5.35 sys

Adiciona uma nova função `sys.unraisablehook()`, que pode ser substituída para controlar como as "exceções não passíveis de levantamento" são tratadas. É chamada quando uma exceção ocorre, mas não há como o Python manipulá-la. Por exemplo, quando um destruidor gera uma exceção ou durante a coleta de lixo (`gc.collect()`). (Contribuição de Victor Stinner em bpo-36829.)

5.36 tarfile

O módulo `tarfile` agora tem como padrão o formato pax moderno (POSIX.1-2001) para novos arquivos, em vez do anterior específico do GNU. Isso melhora a portabilidade de plataforma cruzada com uma codificação consistente (UTF-8) em um formato padronizado e extensível e oferece vários outros benefícios. (Contribuição de C.A.M. Gerlach em bpo-36268.)

5.37 threading

Adiciona uma nova função `threading.excepthook()` que manipula a exceção não capturada `threading.Thread.run()`. Pode ser substituída para controlar como exceções não capturadas `threading.Thread.run()` são manipuladas. (Contribuição de Victor Stinner em bpo-1230540.)

Adiciona uma nova função `threading.get_native_id()` e um atributo `native_id` à classe `threading.Thread`. Eles retornam o Thread ID integral nativo do thread atual atribuído pelo kernel. Este recurso está disponível apenas em determinadas plataformas, consulte `get_native_id` para mais informações. (Contribuição de Jake Tesler em bpo-36084.)

5.38 tokenize

O módulo `tokenize` agora emite implicitamente um token `NEWLINE` quando fornecido com entrada que não possui uma nova linha à direita. Esse comportamento agora corresponde ao que o tokenizador do C faz internamente. (Contribuição de Ammar Askar em bpo-33899.)

5.39 tkinter

Adicionados os métodos `selection_from()`, `selection_present()`, `selection_range()` e `selection_to()` à classe `tkinter.Spinbox`. (Contribuição de Juliette Monsel em bpo-34829.)

Adicionado o método `moveto()` na classe `tkinter.Canvas`. (Contribuição de Juliette Monsel em bpo-23831.)

A classe `tkinter.PhotoImage` agore tem os métodos `transparency_get()` e `transparency_set()`. (Contribuição de Zackery Spytz em bpo-25451.)

5.40 time

Adiciona o novo relógio `CLOCK_UPTIME_RAW` ao macOS 10.12. (Contribuição de Joannah Nanjekye em bpo-35702.)

5.41 typing

O módulo `typing` incorpora vários novos recursos:

- Um tipo de dicionário com tipos por chave. Veja **PEP 589** e `typing.TypedDict`. TypedDict usa apenas chaves de strings. Por padrão, é necessário que todas as chaves estejam presentes. Especifique "total=False" para permitir que as chaves sejam opcionais:

```
class Location(TypedDict, total=False):
    lat_long: tuple
    grid_square: str
    xy_coordinate: tuple
```

- Tipos literais. Veja **PEP 586** e `typing.Literal`. Tipos literais indicam que um parâmetro ou valor de retorno está restrito a um ou mais valores literais específicos:

```
def get_status(port: int) -> Literal['connected', 'disconnected']:
    ...
```

- Variáveis, funções, métodos e classes "finais". Veja **PEP 591**, `typing.Final` e `typing.final()`. O qualificador final instrui um verificador de tipo estático a restringir criação de subclasse, substituição ou reatribuição:

```
pi: Final[float] = 3.1415926536
```

- Definições de protocolo. Veja **PEP 544**, `typing.Protocol` e `typing.runtime_checkable()`. ABCs simples como `typing.SupportsInt` agora são subclasses de `Protocol`.
- Nova classe de protocolo `typing.SupportsIndex`.
- Novas funções `typing.get_origin()` e `typing.get_args()`.

5.42 unicodedata

O módulo `unicodedata` agora foi atualizado para usar o Unicode 12.1.0.

A nova função `is_normalized()` pode ser usada para verificar se uma string está em uma forma normal específica, geralmente muito mais rápida do que normalizar a string. (Contribuição de Max Belanger, David Euresti e Greg Price em bpo-32285 e bpo-37966).

5.43 unittest

Adicionada `AsyncMock` para dar suporte a uma versão assíncrona de `Mock`. Também foram adicionadas novas funções apropriadas para teste. (Contribuição de Lisa Roach em bpo-26467).

Adicionados `addModuleCleanup()` e `addClassCleanup()` a unittest para dar suporte a limpezas para `setUpModule()` e `setUpClass()`. (Contribuição de Lisa Roach em bpo-24412.)

Várias funções de declaração de mock agora também exibem uma lista de chamadas reais em caso de falha. (Contribuição de Petter Strandmark em bpo-35047.)

O módulo `unittest` ganhou suporte a corrotinas a serem usadas como casos de teste com `unittest.IsolatedAsyncioTestCase`. (Contribuição de Andrew Svetlov em bpo-32972.)

Exemplo:

```
import unittest


class TestRequest(unittest.IsolatedAsyncioTestCase):

    async def asyncSetUp(self):
        self.connection = await AsyncConnection()

    async def test_get(self):
        response = await self.connection.get("https://example.com")
        self.assertEqual(response.status_code, 200)

    async def asyncTearDown(self):
        await self.connection.close()


if __name__ == "__main__":
    unittest.main()
```

5.44 venv

`venv` agora inclui um script `Activate.ps1` em todas as plataformas para ativar ambientes virtuais no PowerShell Core 6.1. (Contribuição de Brett Cannon em bpo-32718.)

5.45 weakref

Os objetos proxy retornados por `weakref.proxy()` agora possuem suporte aos operadores de multiplicação de matrizes `@` e `@=` além dos outros operadores numéricos. (Contribuição de Mark Dickinson em bpo-36669.)

5.46 xml

Como atenuação a DTD e recuperação de entidade externa, os módulos `xml.dom.minidom` e `xml.sax` não processam mais entidades externas por padrão. (Contribuição de Christian Heimes em bpo-17239.)

Os métodos `.find*()` no módulo `xml.etree.ElementTree` possuem suporte a pesquisas curinga como `{*}tag`, que ignora o espaço para nome e `{namespace}*`, que retorna todas as tags no espaço para nome especificado. (Contribuição de Stefan Behnel em bpo-28238.)

O módulo `xml.etree.ElementTree` fornece uma nova função `–xml.etree.ElementTree.canonicalize()` que implementa o C14N 2.0. (Contribuição de Stefan Behnel em bpo-13611.)

O objeto de destino de `xml.etree.ElementTree.XMLParser` pode receber eventos de declaração de espaço de nomes através dos novos métodos de retorno de chamada `start_ns()` e `end_ns ()`. Além disso, o destino `xml.etree.ElementTree.TreeBuilder` pode ser configurado para processar eventos sobre comentários e instruções de processamento para incluí-los na árvore gerada. (Contribuição de Stefan Behnel em bpo-36676 e bpo-36673.)

5.47 xmlrpc

`xmlrpc.client.ServerProxy` agora possui suporte a um argumento nomeado *headers* opcional para que uma sequência de cabeçalhos HTTP seja enviada a cada solicitação. Entre outras coisas, isso possibilita a atualização da autenticação básica padrão para uma autenticação de sessão mais rápida. (Contribuição de Cédric Krier em bpo-35153.)

6 Otimizações

- O módulo `subprocess` agora pode usar a função `os.posix_spawn()` em alguns casos para obter melhor desempenho. Atualmente, ele é usado apenas no macOS e Linux (usando glibc 2.24 ou mais recente) se todas essas condições forem atendidas:
  - *close_fds* ser false;
  - os parâmetros *preexec_fn*, *pass_fds*, *cwd* and *start_new_session* não estarem definidos;
  - o caminho *executable* conter um diretório.

  (Contribuição de Joannah Nanjekye e Victor Stinner em bpo-35537.)
- `shutil.copyfile()`, `shutil.copy()`, `shutil.copy2()`, `shutil.copytree()` e `shutil.copytree()` usam chamadas de sistema "fast-copy" específicas da plataforma no Linux e no macOS para copiar o arquivo com mais eficiência. "fast-copy" significa que a operação de cópia ocorre dentro do kernel, evitando o uso de buffers de espaço do usuário no Python como em "`outfd.write(infd.read())`". No Windows, `shutil.copyfile()` usa um buffer padrão com tamanho maior (1 MiB em vez de 16 KiB) e é usada uma variante de `shutil.copyfileobj()` baseada em `memoryview()`. A velocidade de cópia de um arquivo 512 MiB na mesma partição é de aproximadamente +26% no Linux, +50% no macOS e +40% no Windows. Além disso, muito menos ciclos de CPU são consumidos. Veja a seção shutil-platform-dependent-efficient-copy-operations. (Contribuição de Giampaolo Rodolà em bpo-33671.)
- `shutil.copytree()` usa a função `os.scandir()` e todas as funções de cópia que dependem dela usam valores em cache de `os.stat()`. A velocidade para copiar um diretório com 8000 arquivos é de cerca de +9% no Linux, +20% no Windows e +30% em um compartilhamento SMB do Windows. Além disso, o número de

chamadas de sistema `os.stat()` é reduzido em 38%, tornando `shutil.copytree()` especialmente mais rápida em sistemas de arquivos em rede. (Contribuição de Giampaolo Rodolà em bpo-33695.)

- O protocolo padrão no módulo `pickle` agora é o Protocolo 4, introduzido pela primeira vez no Python 3.4. Ele oferece melhor desempenho e tamanho menor em comparação com o Protocolo 3 disponível desde o Python 3.0.
- Removido um membro `Py_ssize_t` de `PyGC_Head`. O tamanho de todos os objetos rastreados pelo coletor de lixo (por exemplo, tupla, lista, ditado) é reduzido em 4 ou 8 bytes. (Contribuição de Inada Naoki em bpo-33597.)
- `uuid.UUID` agora usa `__slots__` para reduzir seu uso de espaço de memória. (Contribuição de Wouter Bolsterlee e Tal Einat em bpo-30977)
- Melhor desempenho de `operator.itemgetter()` em 33%. Manipulação de argumento otimizada e incluído um caminho rápido para o caso comum de um único índice inteiro não negativo em uma tupla (que é o caso de uso típico na biblioteca padrão). (Contribuição de Raymond Hettinger em bpo-35664.)
- Pesquisas de campo aceleradas em `collections.namedtuple()`. Agora elas são duas vezes mais rápidas, tornando-as a forma mais rápida de pesquisa de variáveis de instância no Python. (Contribuição de Raymond Hettinger, Pablo Galindo e Joe Jevnik, Serhiy Storchaka em bpo-32492.)
- O construtor `list` não atribui globalmente o buffer interno do item se a entrada iterável tiver um comprimento conhecido (a entrada implementa `__len__`). Isso torna a lista criada 12% menor em média. (Contribuição de Raymond Hettinger e Pablo Galindo em bpo-33234.)
- Duplicada a velocidade de gravação de variável de classe. Quando um atributo não-dunder era atualizado, havia uma chamada desnecessária para atualizar os slots. (Contribuição de Stefan Behnel, Pablo Galindo Salgado, Raymond Hettinger, Neil Schemenauer e Serhiy Storchaka em bpo-36012.)
- Reduzida a sobrecarga dos argumentos de conversão passados para muitas funções e métodos internos. Isso acelerou chamando algumas funções e métodos internos simples de 20–50%. (Contribuição de Serhiy Storchaka em bpo-23867, bpo-35582 e bpo-36127.)
- A instrução `LOAD_GLOBAL` agora usa o novo mecanismo "por cache de código de operação". É cerca de 40% mais rápido agora. (Contribuição de Yury Selivanov e Inada Naoki em :issue: *26219*.)

7 Alterações a compilações e API C

- O `sys.abiflags` padrão se tornou uma string vazia: o sinalizador `m` para pymalloc se tornou inútil (compilações com e sem pymalloc são compatíveis com ABI) e, portanto, foi removido. (Contribuição de Victor Stinner em bpo-36707.)

  Exemplos de alterações:

  - Somente o programa `python3.8` é instalado, o programa `python3.8m` se foi.
  - Somente o script `python3.8-config` é instalado, o programa `python3.8m-config` se foi.
  - O sinalizador `m` foi removido do sufixo dos nomes de arquivos da biblioteca dinâmica: módulos de extensão na biblioteca padrão, bem como daqueles produzidos e instalados por pacotes de terceiros, como os baixados do PyPI. No Linux, por exemplo, o sufixo de `.cpython-37m-x86_64-linux-gnu.so` do Python 3.7 se tornou `.cpython-38-x86_64-linux-gnu.so` no Python 3.8.
- Os arquivos de cabeçalho foram reorganizados para separar melhor os diferentes tipos de APIs:
  - `Include/*.h` deve ser a API C pública estável e portátil.
  - `Include/cpython/*.h` deve ser a API C instável específica do CPython; API pública, com alguma API privada prefixada por `_Py` ou `_PY`.

  - `Include/internal/*.h` é a API C interna privada muito específica do CPython. Essa API é fornecida sem garantia de compatibilidade com versões anteriores e não deve ser usada fora do CPython. Ele é exposto apenas a necessidades muito específicas, como depuradores e perfis, que precisam acessar os componentes internos do CPython sem chamar funções. Esta API agora está instalada por `make install`.

  (Contribuição de Victor Stinner em bpo-35134 e bpo-35081, trabalho iniciado por Eric Snow em Python 3.7.)

- Algumas macros foram convertidas em funções in-line estáticas: os tipos de parâmetros e o tipo de retorno são bem definidos, eles não têm problemas específicos para as macros, as variáveis têm escopos locais. Exemplos:

  - `Py_INCREF()`, `Py_DECREF()`
  - `Py_XINCREF()`, `Py_XDECREF()`
  - `PyObject_INIT()`, `PyObject_INIT_VAR()`
  - Funções privadas: `_PyObject_GC_TRACK()`, `_PyObject_GC_UNTRACK()`, `_Py_Dealloc()`

  (Contribuição de Victor Stinner em bpo-35059.)

- As funções `PyByteArray_Init()` e `PyByteArray_Fini()` foram removidas. Elas não faziam nada desde o Python 2.7.4 e o Python 3.2.0, foram excluídas da API limitada (ABI estável) e não foram documentados. (Contribuição de Victor Stinner em bpo-35713.)

- O resultado de `PyExceptionClass_Name()` agora é do tipo `const char *` em vez de `char *`. (Contribuição de Serhiy Storchaka em bpo-33818.)

- A dualidade de `Modules/Setup.dist` e `Modules/Setup` foi removida. Anteriormente, ao atualizar a árvore de fontes do CPython, era necessário copiar manualmente `Modules/Setup.dist` (dentro da árvore de fontes) para `Modules/Setup` (dentro da árvore de compilação), a fim de refletir todas as alterações upstream. Isso foi um pequeno benefício para os empacotadores, à custa de um aborrecimento frequente para os desenvolvedores após o desenvolvimento do CPython, pois esquecer de copiar o arquivo pode causar falhas na compilação.

  Agora, o sistema de compilação sempre lê `Modules/Setup` dentro da árvore de fontes. As pessoas que desejam personalizar esse arquivo são incentivadas a manter suas alterações em um fork do git do CPython ou como arquivos de patch, como faria em qualquer outra alteração na árvore de fontes.

  (Contribuição de Antoine Pitrou em bpo-32430.)

- Funções que convertem um número Python em número inteiro C, como `PyLong_AsLong()`, e funções de análise de argumentos, como `PyArg_ParseTuple()`, com unidades de formato de conversão de números inteiros, como `'i'`, agora usarão o método especial `__index__()` em vez de `__int__()`, se disponível. O aviso de descontinuação será emitido para objetos com o método `__int__()`, mas sem o método `__index__()` (como `Decimal` e `Fraction`). `PyNumber_Check()` agora retornará `1` para objetos implementando `__index__()`. `PyNumber_Long()`, `PyNumber_Float()` e `PyFloat_AsDouble()` agora também usam o método `__index__()`, se disponível. (Contribuição de Serhiy Storchaka em bpo-36048 e bpo-20092.)

- Objetos de tipo alocado a heap agora aumentarão sua contagem de referências em `PyObject_Init()` (e sua macro paralela `PyObject_INIT`) em vez de em `PyType_GenericAlloc()`. Os tipos que modificam a alocação ou desalocação de instância podem precisar ser ajustados. (Contribuição de Eddie Elizondo em bpo-35810.)

- A nova função `PyCode_NewWithPosOnlyArgs()` permite criar objetos de código como `PyCode_New()`, mas com um parâmetro extra *posonlyargcount* para indicar o número de argumentos somente posicionais. (Contribuição de Pablo Galindo em bpo-37221.)

- `Py_SetPath()` agora define `sys.executable` para o caminho completo do programa (`Py_GetProgramFullPath()`) em vez do nome do programa (`Py_GetProgramName()`). (Contribuição de Victor Stinner em bpo-38234.)

8 Descontinuados

- O comando distutils `bdist_wininst` agora está descontinuado; use `bdist_wheel` (pacotes wheel). (Contribuição de Victor Stinner em bpo-37481.)
- Os métodos descontinuados `getchildren()` e `getiterator()` no módulo `ElementTree` agora emitem um `DeprecationWarning` em vez de `PendingDeprecationWarning`. Eles serão removidos no Python 3.9. (Contribuição de Serhiy Storchaka em bpo-29209.)
- Passar um objeto que não é uma instância de `concurrent.futures.ThreadPoolExecutor` para `loop.set_default_executor()` foi descontinuado e será proibido no Python 3.9. (Contribuição de Elvis Pranskevichus em bpo-34075.)
- Os métodos `__getitem__()` de `xml.dom.pulldom.DOMEventStream`, `wsgiref.util.FileWrapper` e `fileinput.FileInput` foram descontinuados.

  As implementações desses métodos têm ignorado seu parâmetro *index* e retornado o próximo item. (Contribuição de Berker Peksag em bpo-9372.)
- A classe `typing.NamedTuple` descontinuou o atributo `_field_types` em favor do atributo `__annotations__` que possui as mesmas informações. (Contribuição de Raymond Hettinger em bpo-36320.)
- As classes `Num`, `Str`, `Bytes`, `NameConstant` e `Ellipsis` do módulo `ast` são consideradas descontinuadas e serão removidos em versões futuras do Python. A `Constant` deve ser usado. (Contribuição de Serhiy Storchaka em bpo-32892.)
- Os métodos `visit_Num()`, `visit_Str()`, `visit_Bytes()`, `visit_NameConstant()` e `visit_Ellipsis()` da classe `ast.NodeVisitor` estão agora descontinuados e não serão chamados nas versões futuras do Python. Adicione o método `visit_Constant()` para manipular todos os nós constantes. (Contribuição de Serhiy Storchaka em bpo-36917.)
- O decorador de `asyncio.coroutine()` foi descontinuado e será removido na versão 3.10. Em vez de `@asyncio.coroutine`, use `async def`. (Contribuição de Andrew Svetlov em bpo-36921.)
- Em `asyncio`, a passagem explícita de um argumento *loop* foi descontinuado e será removido na versão 3.10 para os seguintes: `asyncio.sleep()`, `asyncio.gather()`, `asyncio.shield()`, `asyncio.wait_for()`, `asyncio.wait()`, `asyncio.as_completed()`, `asyncio.Task`, `asyncio.Lock`, `asyncio.Event`, `asyncio.Condition`, `asyncio.Semaphore`, `asyncio.BoundedSemaphore`, `asyncio.Queue`, `asyncio.create_subprocess_exec()` e `asyncio.create_subprocess_shell()`.
- A passagem explícita dos objetos da corrotina para `asyncio.wait()` foi descontinuada e será removida na versão 3.11. (Contribuição de Yury Selivanov em bpo-34790.)
- As seguintes funções e os seguintes métodos estão descontinuados no módulo `gettext`: `lgettext()`, `ldgettext()`, `lngettext()` e `ldngettext()`. Eles retornam bytes codificados e é possível que você obtenha exceções relacionadas ao Unicode inesperadas se houver problemas de codificação com as strings traduzidas. É muito melhor usar alternativas que retornam strings Unicode no Python 3. Essas funções estiveram quebradas por um longo tempo.

  Function `bind_textdomain_codeset()`, methods `output_charset()` and `set_output_charset()`, and the *codeset* parameter of functions `translation()` and `install()` are also deprecated, since they are only used for the `l*gettext()` functions. (Contributed by Serhiy Storchaka in bpo-33710.)
- O método `isAlive()` da classe `threading.Thread` foi descontinuado. (Contribuição de Dong-hee Na em bpo-35283.)

- Muitas funções internas e de extensão que recebem argumentos inteiros agora emitem um aviso de descontinuidade para objetos de `Decimal`, de `Fraction` e quaisquer outros objetos que possam ser convertidos apenas em números inteiros com perda (por exemplo, que possuem o método `__int__()`, mas não têm o método `__index__()`). Em uma versão futura, haverá erros. (Contribuição de Serhiy Storchaka em bpo-36048.)
- Descontinuada a passagem dos seguintes argumentos como argumentos nomeados:
  - *func* em `functools.partialmethod()`, `weakref.finalize()`, `profile.Profile.runcall()`, `cProfile.Profile.runcall()`, `bdb.Bdb.runcall()`, `trace.Trace.runfunc()` e `curses.wrapper()`.
  - *function* em `unittest.TestCase.addCleanup()`.
  - *fn* no método `submit()` de `concurrent.futures.ThreadPoolExecutor` e `concurrent.futures.ProcessPoolExecutor`.
  - *callback* em `contextlib.ExitStack.callback()`, `contextlib.AsyncExitStack.callback()` e `contextlib.AsyncExitStack.push_async_callback()`.
  - *c* e *typeid* no método `create()` de `multiprocessing.managers.Server` e `multiprocessing.managers.SharedMemoryServer`.
  - *obj* em `weakref.finalize()`.

  Em versões futuras do Python, elas serão somente-posicionais. (Contribuição de Serhiy Storchaka em bpo-36492.)

9 Remoção de APIs e recursos

Os seguintes recursos e APIs foram removidos do Python 3.8:

- A partir do Python 3.3, a importação de ABCs de `collections` foi descontinuada e a importação deve ser feita de `collections.abc`. A possibilidade de importar de coleções foi marcada para remoção em 3.8, mas foi adiada para 3.9. (Veja bpo-36952.)
- O módulo `macpath`, descontinuado no Python 3.7, foi removido. (Contribuição de Victor Stinner em bpo-35471.)
- A função `platform.popen()` foi removida, após ter sido descontinuada desde o Python 3.3: use `os.popen()`. (Contribuição de Victor Stinner em bpo-35345.)
- A função `time.clock()` foi removida, após ter sido descontinuada desde o Python 3.3: use `time.perf_counter()` ou `time.process_time()`, dependendo de seus requisitos, para ter o comportamento já definido. (Contribuição de Matthias Bussonnier em bpo-36895.)
- O script `pyvenv` foi removido em favor do `python3.8 -m venv` para ajudar a eliminar a confusão sobre a qual interpretador Python o script `pyvenv` está associado. (Contribuição de Brett Cannon em bpo-25427.)
- `parse_qs`, `parse_qsl` e `escape` foram removidos do módulo `cgi`. Eles foram descontinuados no Python 3.2 ou mais antigo. Eles devem ser importados dos módulos `urllib.parse` e `html`.
- A função `filemode` foi removida do módulo `tarfile`. Não é documentada e foi descontinuada desde o Python 3.3.
- O construtor `XMLParser` não aceita mais o argumento *html*. Isso nunca teve efeito e foi descontinuado no Python 3.4. Todos os outros parâmetros agora são somente-nomeados. (Contribuição de Serhiy Storchaka em bpo-29209.)
- Removido o método `doctype()` de `XMLParser`. (Contribuição de Serhiy Storchaka em bpo-29209.)
- O codec "unicode_internal" foi removido. (Contribuição de Inada Naoki em bpo-36297.)
- Os objetos `Cache` e `Statement` do módulo `sqlite3` não expostos para o usuário. (Contribuição de Aviv Palivoda em bpo-30262.)

- O argumento nomeado `bufsize` de `fileinput.input()` e `fileinput.FileInput()`, que era ignorado e estava descontinuado desde o Python 3.6, foi removido. bpo-36952 (Contribuição de Matthias Bussonnier.)
- As funções descontinuadas `sys.set_coroutine_wrapper()` e `sys.get_coroutine_wrapper()` no Python 3.7 foram removidas; bpo-36933 (Contribuição de Matthias Bussonnier.)

10 Portando para o Python 3.8

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

10.1 Alterações no comportamento do Python

- Expressões *yield* (ambas as cláusulas `yield` e `yield from`) agora não são permitidas nas compreensões e expressões geradoras (além da expressão iterável na cláusula mais à esquerda `for`). (Contribuição de Serhiy Storchaka em bpo-10544.)
- Agora, o compilador produz um `SyntaxWarning` quando verificações de identidade (`is` e `is not`) são usadas com certos tipos de literais (por exemplo, strings, números). Geralmente, eles podem funcionar acidentalmente no CPython, mas não são garantidos pelas especificações da linguagem. O aviso aconselha os usuários a usar testes de igualdade (== e `!=`). (Contribuição de Serhiy Storchaka em bpo-34850.)
- O interpretador do CPython pode engolir exceções em algumas circunstâncias. No Python 3.8, isso acontece em menos casos. Em particular, as exceções levantadas ao obter o atributo do dicionário de tipos não são mais ignoradas. (Contribuição de Serhiy Storchaka em bpo-35459.)
- Removidas as implementações `__str__` dos tipos embutidos `bool`, `int`, `float`, `complex` e poucas classes da biblioteca padrão. Agora eles herdam `__str__()` de `object`. Como resultado, definir o método `__repr__()` na subclasse dessas classes afetará sua representação de string. (Contribuição de Serhiy Storchaka em issue:*36793*.)
- No AIX, `sys.platform` não contém mais a versão principal. É sempre `'aix'`, em vez de `'aix3'` .. `'aix7'`. Como as versões mais antigas do Python incluem o número da versão, é recomendável usar sempre `sys.platform.startswith('aix')`. (Contribuição de M. Felt em bpo-36588.)
- `PyEval_AcquireLock()` e `PyEval_AcquireThread()` agora encerram a thread atual se chamado enquanto o interpretador está finalizando, tornando-os consistentes com `PyEval_RestoreThread()`, `Py_END_ALLOW_THREADS()` e `PyGILState_Ensure()`. Se esse comportamento não for desejado, proteja a chamada verificando `_Py_IsFinalizing()` ou `sys.is_finalizing()`. (Contribuição de Joannah Nanjekye em bpo-36475.)

10.2 Alterações na API Python

- A função `os.getcwdb()` agora usa a codificação UTF-8 no Windows, em vez da página de códigos ANSI: veja **PEP 529** para a justificativa. A função não é mais descontinuada no Windows. (Contribuição de Victor Stinner em bpo-37412.)
- `subprocess.Popen` agora pode usar `os.posix_spawn()` em alguns casos, para obter melhor desempenho. No Windows Subsystem para Linux e emulação de usuário QEMU, o construtor `Popen` usando `os.posix_spawn()` não levanta mais uma exceção em erros como "missing program". Em vez disso, o processo filho falha com um diferente de zero `returncode`. (Contribuição de Joannah Nanjekye e Victor Stinner em bpo-35537.)

- O argumento *preexec_fn* de * `subprocess.Popen` não é mais compatível com subinterpretadores. O uso do parâmetro em um subinterpretador agora levanta `RuntimeError`. (Contribuição de Eric Snow em bpo-34651, modificado por Christian Heimes em bpo-37951.)
- O método `imap.IMAP4.logout()` não mais ignora silenciosamente exceções arbitrárias. (Contribuição de Victor Stinner em bpo-36348.)
- A função `platform.popen()` foi removida, após ter sido descontinuada desde o Python 3.3: use `os.popen()`. (Contribuição de Victor Stinner em bpo-35345.)
- A função `Statistics.mode()` não levanta mais uma exceção quando dados multimodais são fornecidos. Em vez disso, ela retorna o primeiro modo encontrado nos dados de entrada. (Contribuição de Raymond Hettinger em bpo-35892.)
- O método `selection()` da classe `tkinter.ttk.Treeview` não recebe mais argumentos. Seu uso com argumentos para alterar a seleção foi descontinuado no Python 3.6. Use métodos especializados como `selection_set()` para alterar a seleção. (Contribuição de Serhiy Storchaka em bpo-31508.)
- Os métodos `writexml()`, `toxml()` e `toprettyxml()` de `xml.dom.minidom`, e o método `write()` de `xml.etree`, agora preservam a ordem dos atributos especificados pelo usuário. (Contribuição de Diego Rojas e Raymond Hettinger em bpo-34160.)
- Um banco de dados `dbm.dumb` aberto com sinalizadores `'r'` agora é somente leitura. `dbm.dumb.open()` com os sinalizadores `'r'` e `'w'` não cria mais um banco de dados se ele não existir. (Contribuição de Serhiy Storchaka em bpo-32749.)
- O método `doctype()` definido em uma subclasse de `XMLParser` não será mais chamado e emitirá um `RuntimeWarning` em vez de um `DeprecationWarning`. Defina o método `doctype()` em um alvo para manipular uma declaração doctype XML. (Contribuição de Serhiy Storchaka em bpo-29209.)
- Uma `RuntimeError` agora é levantada quando a metaclasse personalizada não fornece a entrada `__classcell__` no espaço de nomes passado para `type.__new__`. Uma `DeprecationWarning` era emitida no Python 3.6-3.7. (Contribuição de Serhiy Storchaka em bpo-23722.)
- A classe `cProfile.Profile` agora pode ser usada como um gerenciador de contexto. (Contribuição de Scott Sanderson em bpo-29235.)
- `shutil.copyfile()`, `shutil.copy()`, `shutil.copy2()`, `shutil.copytree()` e `shutil.copytree()` usam chamadas de sistema de "cópia rápido" específica da plataforma (veja a seção shutil-platform-dependent-efficient-copy-operations).
- O tamanho padrão do buffer de `shutil.copyfile()` no Windows foi alterado de 16 KiB para 1 MiB.
- A estrutura de `PyGC_Head` mudou completamente. Todo o código que tocava no membro struct deve ser reescrito. (Veja bpo-33597.)
- A estrutura de `PyInterpreterState` foi movida para os arquivos de cabeçalho "internal" (especificamente Include/internal/pycore_pystate.h). Um `PyInterpreterState` opaco ainda está disponível como parte da API pública (e ABI estável). Os documentos indicam que nenhum dos campos da estrutura é público, por isso esperamos que ninguém os tenha usado. No entanto, se você confia em um ou mais desses campos em particular e não tiver alternativa, abra um relatório de erro no BPO. Vamos trabalhar para ajudá-lo a ajustar (possivelmente incluindo a adição de funções de acessador à API pública). (Veja bpo-35886.)
- O método `mmap.flush()` agora retorna `None` com êxito e levanta uma exceção no erro em todas as plataformas. Anteriormente, seu comportamento dependia da plataforma: um valor diferente de zero era retornado com êxito; zero era retornado com erro no Windows. Um valor zero era retornado com sucesso; uma exceção era levanta no caso de erro no Unix. (Contribuição de Berker Peksag em bpo-2122.)
- Os módulos `xml.dom.minidom` e `xml.sax` não mais processam entidades externas por padrão. (Contribuição de Christian Heimes em bpo-17239.)

- A exclusão de uma chave de um banco de dados somente leitura `dbm` (`dbm.dumb`, `dbm.gnu` ou `dbm.ndbm`) levanta `error` (`dbm.dumb.error`, `dbm.gnu.error` ou `dbm.ndbm.error`) em vez de `KeyError`. (Contribuição de Xiang Zhang em bpo-33106.)
- AST foi simplificado para literais. Todas as constantes serão representadas como instâncias de `ast.Constant`. Instanciar classes antigas `Num`, `Str`, `Bytes`, `NameConstant` e `Ellipsis` retornará uma instância de `Constant`. (Contribuição de Serhiy Storchaka em bpo-32892.)
- `expanduser()` no Windows agora prefere a variável de ambiente `USERPROFILE` e não usa `HOME`, que normalmente não é definido para contas de usuário comuns. (Contribuição de Anthony Sottile em bpo-36264.)
- The exception `asyncio.CancelledError` now inherits from `BaseException` rather than `Exception` and no longer inherits from `concurrent.futures.CancelledError`. (Contributed by Yury Selivanov in bpo-32528.)
- A função `asyncio.wait_for()` agora aguarda corretamente o cancelamento ao usar uma instância de `asyncio.Task`. Anteriormente, ao atingir tempo limite, ele era cancelado e devolvido imediatamente. (Contribuição de Elvis Pranskevichus em bpo-32751.)
- A função `asyncio.BaseTransport.get_extra_info()` agora retorna um objeto soquete seguro de usar quando 'socket' é passado para o parâmetro *name*. (Contribuição de Yury Selivanov em bpo-37027.)
- `asyncio.BufferedProtocol` foi promovida a API estável.
- As dependências de DLL para módulos de extensão e DLLs carregadas com `ctypes` no Windows agora são resolvidas com mais segurança. Somente os caminhos do sistema, o diretório que contém o arquivo DLL ou PYD e os diretórios adicionados com `add_dll_directory()` são pesquisados por dependências de tempo de carregamento. Especificamente, `PATH` e o diretório de trabalho atual não são mais usados, e as modificações nelas não terão mais efeito na resolução normal da DLL. Se seu aplicativo conta com esses mecanismos, você deve verificar `add_dll_directory()` e, se existir, use-o para adicionar seu diretório DLLs ao carregar sua biblioteca. Observe que os usuários do Windows 7 precisarão garantir que o Windows Update KB2533623 esteja instalado (isso também é verificado pelo instalador). (Contribuição de Steve Dower em bpo-36085.)
- Os arquivos de cabeçalho e as funções relacionadas ao pgen foram removidos após sua substituição por uma implementação pura do Python. (Contribuição de Pablo Galindo em bpo-36623.)
- `types.CodeType` possui um novo parâmetro na segunda posição do construtor (*posonlyargcount *) para suportar argumentos apenas de posição definidos em :pep:'570'. O primeiro argumento (*argcount*) agora representa o número total de argumentos posicionais (incluindo argumentos somente posicionais). O novo método `replace()` de `types.CodeType` pode ser usado para tornar o código à prova de futuro.
- The parameter `digestmod` for `hmac.new()` no longer uses the MD5 digest by default.

10.3 Alterações na API C

- A estrutura `PyCompilerFlags` recebeu um novo campo *cf_feature_version*. Ele deve ser inicializado para `PY_MINOR_VERSION`. O campo é ignorado por padrão e é usado se e somente se o sinalizador `PyCF_ONLY_AST` estiver definido em *cf_flags*. (Contribuição de Guido van Rossum em bpo-35766.)
- A função `PyEval_ReInitThreads()` foi removida da API C. Não deve ser chamada explicitamente: use `PyOS_AfterFork_Child()`. (Contribuição de Victor Stinner em bpo-36728.)
- No Unix, as extensões C não estão mais vinculadas ao libpython, exceto no Android e Cygwin. Quando o Python é incorporado, `libpython` não deve ser carregado com `RTLD_LOCAL`, mas `RTLD_GLOBAL`. Anteriormente, usando `RTLD_LOCAL`, já não era possível carregar extensões C que não estavam vinculadas a `libpython`, como extensões C da biblioteca padrão criada pela seção `*shared*` de `Modules/Setup`. (Contribuição de Victor Stinner em bpo-21536.)

- O uso de variantes # de formatos na análise ou construção de valor (por exemplo, `PyArg_ParseTuple()`, `Py_BuildValue()`, `PyObject_CallFunction()`, etc.) sem `PY_SSIZE_T_CLEAN` definido agora levanta `DeprecationWarning`. Será removido em 3.10 ou 4.0. Leia arg-parsing para detalhes. (Contribuição de Inada Naoki em bpo-36381.)

- Instâncias de tipos alocados por heap (como aqueles criados com `PyType_FromSpec()`) mantêm uma referência ao seu objeto de tipo. O aumento da contagem de referências desses objetos de tipo foi movido de `PyType_GenericAlloc()` para as funções de nível mais baixo, `PyObject_Init()` e `PyObject_INIT()`. Isso faz com que os tipos criados através de `PyType_FromSpec()` se comportem como outras classes no código gerenciado.

  Tipos alocados estaticamente não são afetados.

  Para a grande maioria dos casos, não deve haver efeito colateral. No entanto, tipos que aumentam manualmente a contagem de referência após alocar uma instância (talvez para solucionar o bug) agora podem se tornar imortais. Para evitar isso, essas classes precisam chamar Py_DECREF no objeto de tipo durante a desalocação da instância.

  Para portar corretamente esses tipos para o 3.8, aplique as seguintes alterações:

  - Remove `Py_INCREF` no objeto de tipo depois de alocar uma instância – se houver. Isso pode acontecer após a chamada `PyObject_New()`, `PyObject_NewVar()`, `PyObject_GC_New()`, `PyObject_GC_NewVar()` ou qualquer outro alocador personalizado que use `PyObject_Init()` ou `PyObject_INIT()`.

    Exemplo:

    ```
    static foo_struct *
    foo_new(PyObject *type) {
        foo_struct *foo = PyObject_GC_New(foo_struct, (PyTypeObject *) type);
        if (foo == NULL)
            return NULL;
    #if PY_VERSION_HEX < 0x03080000
        // Workaround for Python issue 35810; no longer necessary in Python 3.8
        PY_INCREF(type)
    #endif
        return foo;
    }
    ```

  - Certifica-se de que todas as funções `tp_dealloc` personalizadas de tipos alocados em heap diminuam a contagem de referências do tipo.

    Exemplo:

    ```
    static void
    foo_dealloc(foo_struct *instance) {
        PyObject *type = Py_TYPE(instance);
        PyObject_GC_Del(instance);
    #if PY_VERSION_HEX >= 0x03080000
        // This was not needed before Python 3.8 (Python issue 35810)
        Py_DECREF(type);
    #endif
    }
    ```

  (Contribuição de Eddie Elizondo em bpo-35810.)

- A macro `Py_DEPRECATED()` foi implementada para MSVC. A macro agora deve ser colocada antes do nome do símbolo.

  Exemplo:

```
Py_DEPRECATED(3.8) PyAPI_FUNC(int) Py_OldFunction(void);
```

(Contribuição de Zackery Spytz em bpo-33407.)

- O interpretador não pretende mais dar suporte à compatibilidade binária de tipos de extensão nas liberações de recursos. O `PyTypeObject` exportado por um módulo de extensão de terceiros deve ter todos os slots esperados na versão atual do Python, incluindo `tp_finalize` (`Py_TPFLAGS_HAVE_FINALIZE` não é mais verificado antes de ler `tp_finalize`).

  (Contribuição de Antoine Pitrou em bpo-32388.)

- As funções `PyNode_AddChild()` e `PyParser_AddToken()` agora aceitam dois argumentos adicionais `int` *end_lineno* e *end_col_offset*.

- O arquivo `libpython38.a` para permitir que as ferramentas do MinGW se vinculem diretamente a `python38.dll` não está mais incluído na distribuição regular do Windows. Se você precisar deste arquivo, ele poderá ser gerado com as ferramentas `gendef` e `dlltool`, que fazem parte do pacote binutils do MinGW:

```
gendef - python38.dll > tmp.def
dlltool --dllname python38.dll --def tmp.def --output-lib libpython38.a
```

  A localização de um arquivo `pythonXY.dll` dependerá das opções de instalação e da versão e idioma do Windows. Consulte using-on-windows para mais informações. A biblioteca resultante deve ser colocada no mesmo diretório que `pythonXY.lib`, que geralmente é o diretório `libs` na sua instalação do Python.

  (Contribuição de Steve Dower em bpo-37351.)

10.4 Alterações de bytecode do CPython

- O loop do interpretador foi simplificado movendo a lógica de desenrolar a pilha de blocos no compilador. O compilador agora emite instruções explícitas para ajustar a pilha de valores e chamar o código de limpeza para `break`, `continue` e `return`.

  Removidos os códigos de operação `BREAK_LOOP`, `CONTINUE_LOOP`, `SETUP_LOOP` e `SETUP_EXCEPT`. Adicionados novos códigos de operação `ROT_FOUR`, `BEGIN_FINALLY`, `CALL_FINALLY` e `POP_FINALLY`. Alterado o comportamento de `END_FINALLY` e `WITH_CLEANUP_START`.

  (Contribuição de Mark Shannon, Antoine Pitrou e Serhiy Storchaka em bpo-17611.)

- Adicionado novo código de operação `END_ASYNC_FOR` para lidar com exceções geradas ao aguardar um próximo item em um loop `async for`. (Contribuição de Serhiy Storchaka em bpo-33041.)

- O `MAP_ADD` agora espera o valor como o primeiro elemento na pilha e a chave como o segundo elemento. Essa alteração foi feita para que a chave seja sempre avaliada antes do valor na compreensão do dicionário, conforme proposto em **PEP 572**. (Contribuição de Jörn Heissler em bpo-35224.)

10.5 Ferramentas e daemons

Adicionado um script de benchmark para cronometrar várias maneiras de acessar variáveis: `Tools/scripts/var_access_benchmark.py`. (Contribuição de Raymond Hettinger em bpo-35884.)

Aqui está um resumo das melhorias de desempenho desde Python 3.3:

```
Python version                       3.3     3.4     3.5     3.6     3.7     3.8
--------------                       ---     ---     ---     ---     ---     ---

Variable and attribute read access:
```

(continua na próxima página)

(continuação da página anterior)

```
    read_local                          4.0      7.1      7.1      5.4      5.1      3.9
    read_nonlocal                       5.3      7.1      8.1      5.8      5.4      4.4
    read_global                        13.3     15.5     19.0     14.3     13.6      7.6
    read_builtin                       20.0     21.1     21.6     18.5     19.0      7.5
    read_classvar_from_class           20.5     25.6     26.5     20.7     19.5     18.4
    read_classvar_from_instance        18.5     22.8     23.5     18.8     17.1     16.4
    read_instancevar                   26.8     32.4     33.1     28.0     26.3     25.4
    read_instancevar_slots             23.7     27.8     31.3     20.8     20.8     20.2
    read_namedtuple                    68.5     73.8     57.5     45.0     46.8     18.4
    read_boundmethod                   29.8     37.6     37.9     29.6     26.9     27.7

Variable and attribute write access:
    write_local                         4.6      8.7      9.3      5.5      5.3      4.3
    write_nonlocal                      7.3     10.5     11.1      5.6      5.5      4.7
    write_global                       15.9     19.7     21.2     18.0     18.0     15.8
    write_classvar                     81.9     92.9     96.0    104.6    102.1     39.2
    write_instancevar                  36.4     44.6     45.8     40.0     38.9     35.5
    write_instancevar_slots            28.7     35.6     36.1     27.3     26.6     25.7

Data structure read access:
    read_list                          19.2     24.2     24.5     20.8     20.8     19.0
    read_deque                         19.9     24.7     25.5     20.2     20.6     19.8
    read_dict                          19.7     24.3     25.7     22.3     23.0     21.0
    read_strdict                       17.9     22.6     24.3     19.5     21.2     18.9

Data structure write access:
    write_list                         21.2     27.1     28.5     22.5     21.6     20.0
    write_deque                        23.8     28.7     30.1     22.7     21.8     23.5
    write_dict                         25.9     31.4     33.3     29.3     29.2     24.7
    write_strdict                      22.9     28.4     29.9     27.5     25.2     23.1

Stack (or queue) operations:
    list_append_pop                   144.2     93.4    112.7     75.4     74.2     50.8
    deque_append_pop                   30.4     43.5     57.0     49.4     49.2     42.5
    deque_append_popleft               30.8     43.7     57.3     49.7     49.7     42.8

Timing loop:
    loop_overhead                       0.3      0.5      0.6      0.4      0.3      0.3
```

Os benchmarks foram medidos em um processador Intel® Core™ i7-4960HQ nos sistemas macOS 64-bit encontrados em python.org. O script de benchmark exibe tempos em nanossegundos.

11 Alterações notáveis no Python 3.8.1

Devido a significativas preocupações de segurança, o parâmetro *reuse_address* de `asyncio.loop.create_datagram_endpoint()` não é mais suportado. Isso ocorre devido ao comportamento da opção de soquete `SO_REUSEADDR` no UDP. Para mais detalhes, consulte a documentação de `loop.create_datagram_endpoint()`. (Contribuição de Kyle Stanley, Antoine Pitrou e Yury Selivanov em bpo-37228.)

12 Alterações notáveis no Python 3.8.2

Corrigida uma regressão com o retorno de chamada `ignore` de `shutil.copytree()`. Os tipos de argumento agora são str e List[str] novamente. (Contribuição de Manuel Barkhau e Giampaolo Rodola em bpo-39390.)

13 Alterações notáveis no Python 3.8.3

Os valores constantes dos futuros sinalizadores no módulo `__future__` fpram atualizados para evitar colisão com os sinalizadores do compilador. Anteriormente, `PyCF_ALLOW_TOP_LEVEL_AWAIT` estava colidindo com `CO_FUTURE_DIVISION`. (Contribuição de Batuhan Taskaya em bpo-39562)

14 Notable changes in Python 3.8.8

Earlier Python versions allowed using both `;` and `&` as query parameter separators in `urllib.parse.parse_qs()` and `urllib.parse.parse_qsl()`. Due to security concerns, and to conform with newer W3C recommendations, this has been changed to allow only a single separator key, with `&` as the default. This change also affects `cgi.parse()` and `cgi.parse_multipart()` as they use the affected functions internally. For more details, please see their respective documentation. (Contributed by Adam Goldschmidt, Senthil Kumaran and Ken Jin in bpo-42967.)

15 Notable changes in Python 3.8.9

A security fix alters the `ftplib.FTP` behavior to not trust the IPv4 address sent from the remote server when setting up a passive data channel. We reuse the ftp server IP address instead. For unusual code requiring the old behavior, set a `trust_server_pasv_ipv4_address` attribute on your FTP instance to `True`. (See bpo-43285)

16 Notable changes in Python 3.8.10

16.1 macOS 11.0 (Big Sur) and Apple Silicon Mac support

As of 3.8.10, Python now supports building and running on macOS 11 (Big Sur) and on Apple Silicon Macs (based on the `ARM64` architecture). A new universal build variant, `universal2`, is now available to natively support both `ARM64` and `Intel 64` in one set of executables. Note that support for "weaklinking", building binaries targeted for newer versions of macOS that will also run correctly on older versions by testing at runtime for missing features, is not included in this backport from Python 3.9; to support a range of macOS versions, continue to target for and build on the oldest version in the range.

(Originally contributed by Ronald Oussoren and Lawrence D'Anna in bpo-41100, with fixes by FX Coudert and Eli Rykoff, and backported to 3.8 by Maxime Bélanger and Ned Deily)

17 Notable changes in Python 3.8.10

17.1 urllib.parse

The presence of newline or tab characters in parts of a URL allows for some forms of attacks. Following the WHATWG specification that updates **RFC 3986**, ASCII newline `\n`, `\r` and tab `\t` characters are stripped from the URL by the parser in `urllib.parse` preventing such attacks. The removal characters are controlled by a new module level variable `urllib.parse._UNSAFE_URL_BYTES_TO_REMOVE`. (See bpo-43882)

18 Notable changes in Python 3.8.12

18.1 Alterações na API Python

Starting with Python 3.8.12 the `ipaddress` module no longer accepts any leading zeros in IPv4 address strings. Leading zeros are ambiguous and interpreted as octal notation by some libraries. For example the legacy function `socket.inet_aton()` treats leading zeros as octal notation. glibc implementation of modern `inet_pton()` does not accept any leading zeros.

(Originally contributed by Christian Heimes in bpo-36384, and backported to 3.8 by Achraf Merzouki)

19 Notable security feature in 3.8.14

Converting between `int` and `str` in bases other than 2 (binary), 4, 8 (octal), 16 (hexadecimal), or 32 such as base 10 (decimal) now raises a `ValueError` if the number of digits in string form is above a limit to avoid potential denial of service attacks due to the algorithmic complexity. This is a mitigation for CVE-2020-10735. This limit can be configured or disabled by environment variable, command line flag, or `sys` APIs. See the integer string conversion length limitation documentation. The default limit is 4300 digits in string form.

20 Notable Changes in 3.8.17

20.1 tarfile

- The extraction methods in `tarfile`, and `shutil.unpack_archive()`, have a new a *filter* argument that allows limiting tar features than may be surprising or dangerous, such as creating files outside the destination directory. See tarfile-extraction-filter for details. In Python 3.12, use without the *filter* argument will show a `DeprecationWarning`. In Python 3.14, the default will switch to `'data'`. (Contributed by Petr Viktorin in **PEP 706**.)

Índice

H

P

R

U

V