
What's New in Python

发行版本 3.13.0rc2

A. M. Kuchling

九月 23, 2024

Python Software Foundation
Email: docs@python.org

Contents

1	摘要 -- 发布重点	3
2	新的特性	5
2.1	更好的交互式解释器	5
2.2	改进的错误消息	6
2.3	自由线程的 CPython	7
2.4	实验性的即时 (JIT) 编译器	7
2.5	针对 <code>locals()</code> 的已定义修改语义	8
2.6	对移动平台的支持	8
2.7	增量式垃圾回收	9
3	其他语言特性修改	9
4	新增模块	10
5	改进的模块	11
5.1	<code>argparse</code>	11
5.2	<code>array</code>	11
5.3	<code>ast</code>	11
5.4	<code>asyncio</code>	11
5.5	<code>base64</code>	12
5.6	<code>compileall</code>	12
5.7	<code>configparser</code>	12
5.8	<code>concurrent.futures</code>	12
5.9	<code>configparser</code>	12
5.10	<code>copy</code>	13
5.11	<code>dbm</code>	13
5.12	<code>dis</code>	13
5.13	<code>doctest</code>	13
5.14	<code>email</code>	14
5.15	<code>fractions</code>	14
5.16	<code>gc</code>	14
5.17	<code>glob</code>	14
5.18	<code>importlib</code>	14

5.19	io	15
5.20	ipaddress	15
5.21	itertools	15
5.22	marshal	15
5.23	math	15
5.24	mimetypes	15
5.25	mmap	16
5.26	multiprocessing	16
5.27	os	16
5.28	os.path	17
5.29	pathlib	17
5.30	pdb	17
5.31	queue	18
5.32	random	18
5.33	re	18
5.34	shutil	18
5.35	site	18
5.36	sqlite3	18
5.37	ssl	18
5.38	statistics	19
5.39	subprocess	19
5.40	sys	19
5.41	tempfile	19
5.42	time	19
5.43	tkinter	19
5.44	回溯	20
5.45	types	20
5.46	typing	20
5.47	unicodedata	21
5.48	venv	21
5.49	warnings	21
5.50	xml	21
5.51	zipimport	21
6	性能优化	21
7	被移除的模块与 API	22
7.1	PEP 594: Remove "dead batteries" from the standard library	22
7.2	2to3	23
7.3	builtins	23
7.4	configparser	23
7.5	importlib.metadata	23
7.6	locale	23
7.7	opcode	24
7.8	pathlib	24
7.9	re	24
7.10	tkinter.tix	24
7.11	turtle	24
7.12	typing	24
7.13	unittest	24
7.14	urllib	25
7.15	webbrowser	25
8	新的弃用	25

8.1	计划在 Python 3.14 中移除	27
8.2	Python 3.15 中的待移除功能	28
8.3	计划在 Python 3.16 中移除	29
8.4	计划在未来版本中移除	29
9	CPython 字节码的变化	32
10	C API 的变化	32
10.1	新的特性	32
11	构建变化	35
12	Porting to Python 3.13	35
12.1	Python API 的变化	35
12.2	C API 的变化	36
12.3	Removed C APIs	37
12.4	已弃用的 C API	39
13	回归测试的变化	42
	索引	43

编者

Thomas Wouters

本文介绍了 Python 3.13 相比 3.12 增加的新特性。Python 3.13 将于 2024 年 10 月 1 日发布。要获取详细信息，可参阅 changelog。

参见

PEP 719 -- Python 3.13 发布计划

备注

预发布版用户应当了解到此文档目前处于草稿状态。它将随着 Python 3.13 的发步进程不断更新，因此即使已经阅读过较早的版本也仍然值得再次查看。

1 摘要 -- 发布重点

Python 3.13 将是 Python 编程语言的最新稳定发布版，包含多项针对语言、实现和标准库的改变。最大的变化包括一个新的交互式解释器，以及对于在自由线程模式 (**PEP 703**) 下运行和即时编译器 (**PEP 744**) 的实验性支持。

错误消息继续得到改进，回溯信息现在默认使用彩色高亮显示。locals() 内置函数现在对于修改所返回的映射具有更细化的语法，并且类型形参现在支持设置默认值。

针对标准库的改变包括移除已弃用的 API 和模块，以及用户友好度和正确性方面的常规提升。一些旧式标准库模块自 Python 3.11 起被弃用 (**PEP 594**) 之后现在已被移除。

本文并不试图提供所有新特性的完整规范说明，而是提供一个方便的概览。要了解完整细节请参阅相应文档，如 [标准库参数和语言参考](#)。要了解某项改变的完整实现和设计理念，请参阅相应新特性的 [PEP](#)；但请注意一旦某项特性已完全实现则相应 [PEP](#) 通常不会再继续更新。请参阅[迁移到 Python 3.13](#) 了解如何从较早 Python 进行升级的指导。

解释器的改进：

- 大幅改进的[交互式解释器](#)和[改进的错误消息](#)。
- [PEP 667](#): 现在 `locals()` 内置函数在修改被返回的映射时具有[已定义语义](#)。Python 调试器及类似的工具现在即使在并发代码执行期间也能更可靠地在已优化的作用域中更新局部变量。
- [PEP 703](#): CPython 3.13 具有对在运行时禁用 `global interpreter lock` 的实验性支持。请参阅[自由线程 CPython](#) 了解详情。
- [PEP 744](#): 增加了一个基本的[JIT 编译器](#)。目前默认是禁用的（但以后可能启用）。能够小幅提升性能 -- 我们预计在接下来的几个发布版中不断改进它。
- 在新的[交互式解释器](#)中，以及[回溯信息](#)和[文档测试](#)输出中的颜色支持。这可以通过 `PYTHON_COLORS` 和 `NO_COLOR` 环境变量来禁用。

对 Python 数据模型的改进：

- `__static_attributes__` 存储了可在一个类体的任何函数中通过 `self.X` 来访问的属性名称。
- `__firstlineno__` 记录了一个类定义的首行的行号。

标准库中的重大改进：

- 新增了 `PythonFinalizationError` 异常，当操作在最终化期间被阻塞时将被引发。
- 现在 `argparse` 模块可支持弃用命令行选项、位置参数和子命令。
- 新增的函数 `base64.z85encode()` 和 `base64.z85decode()` 支持对 [Z85 数据](#) 进行编码和解码。
- 现在 `copy` 模块有一个 `copy.replace()` 函数，支持许多内置类型和任何定义了 `__replace__()` 方法的类。
- 现在 `dbm.sqlite3` 模块是默认的 `dbm` 后端。
- `os` 模块增加了一套新函数用于处理 Linux 的定时器通知文件描述符。
- 现在 `random` 模块提供了一个命令行界面。

安全改进：

- `ssl.create_default_context()` 设置了 `ssl.VERIFY_X509_PARTIAL_CHAIN` 和 `ssl.VERIFY_X509_STRICT` 作为默认的旗标。

C API 的改进：

- 现在 `Py_mod_gil` 槽位被用来指明一个扩展模块支持在禁用 GIL 的情况下运行。
- 增加了 `PyTime` C API，提供了对系统时钟的访问。
- `PyMutex` 是新增的轻量级互斥锁，只占用一个字节。

新的类型标注特性：

- [PEP 696](#): 类型形参 (`typing.TypeVar`, `typing.ParamSpec` 和 `typing.TypeVarTuple`) 现在可支持默认值。
- [PEP 702](#): 新的 `warnings.deprecated()` 装饰器在类型系统中增加了对标记为弃用的支持。
- [PEP 705](#): `typing.ReadOnly` 可被用来将 `typing.TypedDict` 的项标记为对类型检查器只读。

- **PEP 742**: `typing.TypeIs` 提供了更直观的类型细化行为，作为对 `typing.TypeGuard` 的替代。

平台支持：

- **PEP 730**: 现在 Apple 的 iOS 是官方支持的平台，处于 **第 3 层级**。官方 Android 支持 (**PEP 738**) 也在计划中。
- 现在 `wasm32-wasi` 作为 **第 2 层级** 的平台受到支持。
- `wasm32-emscripten` 不再是受到官方支持的平台。

重要的移除：

- **PEP 594**: 剩余的 19 个“死电池”已从标准库中移除: `aifc`, `audioop`, `cgi`, `cgitb`, `chunk`, `crypt`, `imghdr`, `mailcap`, `msilib`, `nis`, `nntplib`, `ossaudiodev`, `pipes`, `sndhdr`, `spwd`, `sunau`, `telnetlib`, `uu` 和 `xdrlib`。
- 移除了 **2to3** 工具和 `lib2to3` 模块（在 Python 3.11 中已被弃用）。
- 移除了 `tkinter.tix` 模块（在 Python 3.6 中已被弃用）。
- 移除了 `locale.resetlocale()`。
- 移除了 `typing.io` 和 `typing.re`。
- 移除了链式的 `classmethod` 描述器。

发布计划的变化：

PEP 602 (“Annual Release Cycle for Python”) 已被更新为将新发布版的完整支持 (‘bugfix’) 期扩展至两年。这个更新的政策意味着：

- Python 3.9--3.12 有一年半的完整支持，另加三年半的安全修正。
- Python 3.13 及以后的版本有两年的完整支持，另加三年的安全修正。

2 新的特性

2.1 更好的交互式解释器

Python 现在默认会使用新的 interactive shell，它基于来自 **PyPy** 项目的代码。当使用从交互式终端启动 REPL 时，下列新特性将受到支持：

- 多行编辑并保留历史记录。
- 对 REPL 专属的命令如 `help`, `exit` 和 `quit` 的直接支持，无需以函数形式调用它们。
- 提示和回溯 默认启用彩色显示。
- 使用 F1 浏览交互式帮助并带有单独的命令历史。
- 使用 F2 浏览去除了输出以及 `>>` 和 `...` 提示符的历史。
- 使用 F3 进入“粘贴模式”以更方便地粘贴大段代码（再次按 F3 返回常规提示符）。

要禁用新的交互式 shell，可设置 `PYTHON_BASIC_REPL` 环境变量。有关交互模式的详情，请参见 `tut-interac`。

（由 Pablo Galindo Salgado, Łukasz Langa 和 Lysandros Nikolaou 在 [gh-111201](#) 基于来自 PyPy 项目的代码贡献。Windows 支持由 Dino Viehland 和 Anthony Shaw 贡献。）

2.2 改进的错误消息

- 在终端里显示回溯时解释器现在会默认使用彩色。此特性可通过新的 `PYTHON_COLORS` 环境变量以及传统的 `NO_COLOR` 和 `FORCE_COLOR` 环境变量来进行控制。(由 Pablo Galindo Salgado 在 [gh-112730](#) 中贡献。)
- 一个常见错误是撰写的脚本和标准库中的某个模块重名。现在出现此类错误时会显示一条更有用的错误信息：

```
$ python random.py
Traceback (most recent call last):
  File "/home/me/random.py", line 1, in <module>
    import random
  File "/home/me/random.py", line 3, in <module>
    print(random.randint(5))
    ^^^^^^^^^^^^^^^^^
AttributeError: module 'random' has no attribute 'randint' (consider renaming '/home/me/random.py' since it has the same name as the standard library module named 'random' and the import system gives it precedence)
```

类似地，如果一个脚本具有与它尝试导入的第三方模块相同的名称并因此导致错误，我们也会显示一条更有帮助的错误消息：

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

(由 Shantanu Jain 在 [gh-95754](#) 中贡献。)

- 现在当向一个函数传入不正确的关键字参数时错误消息会尝试提示正确的关键字参数。

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

(由 Pablo Galindo Salgado 和 Shantanu Jain 在 [gh-107944](#) 中贡献。)

2.3 自由线程的 CPython

现在 CPython 具有对运行于自由线程模式的实验性支持，即禁用 global interpreter lock (GIL)。这是一个实验性的特性因而默认是不启用的。自由线程模式需要一个不同的可执行程序，通常名为 `python3.13t` 或 `python3.13t.exe`。标记为自由线程的预构建二进制文件可作为官方 Windows 和 macOS 安装器的一部分被安装，或者可以附带 `--disable-gil` 选项使用源代码来构建 CPython。

自由线程式的执行允许在可用的 CPU 核心上并行地运行线程从而充分利用可用的处理能力。虽然并非所有软件都能自动从中受益，但在设计时将线程纳入考虑的程序在多核心硬件上运行速度会更快。**自由线程模式是实验性的**并且处于不断改进的过程中：预计会出现一些程序错误并且在单线程场景下出现明显的性能损失。可以选择使用环境变量 `PYTHON_GIL` 或命令行选项 `-X gil` 让 CPython 的自由线程构建版支持在运行时启用 GIL。

为了检查当前解释器是否支持自由线程，`python -VV` 和 `sys.version` 将包含“experimental free-threading build”字样。可以使用新增的 `sys._is_gil_enabled()` 函数来检查正在运行的线程是否确实禁用了 GIL。

C-API 扩展模块需要针对自由线程构建版专门进行构建。支持在禁用 GIL 的情况下运行的扩展应当使用 `Py_mod_gil` 槽位。使用单阶段初始化的扩展应当使用 `PyUnstable_Module_SetGIL()` 来指明它们是支持在禁用 GIL 的情况下运行。导入不使用这些机制的 C 扩展将导致 GIL 被启用，除非通过 `PYTHON_GIL` 环境变量或 `-X gil=0` 选项显式地禁用 GIL。需要 pip 24.1 或更新的版本才能在自由线程构建版中安装带有 C 扩展的软件包。

参见

PEP 703 “Making the Global Interpreter Lock Optional in CPython” 中包含了有关此项工作的理念和信息。

[Porting Extension Modules to Support Free-Threading](#): 一份由社区维护的针对扩展开发者的移植指南。

2.4 实验性的即时 (JIT) 编译器

当 CPython 使用 `--enable-experimental-jit` 选项进行配置和构建时，会添加一个即时 (JIT) 编译器以加快某些 Python 程序的运行速度。在 Windows 上，可使用 `PCbuild/build.bat --experimental-jit` 启用 JIT 或使用 `--experimental-jit-interpreter` 启用第 2 层级解释器。构建要求和进一步的支持信息 [包含在 Tools/jit/README.md](#) 中。

`--enable-experimental-jit` 选项接受这些（可选）值，如果不带可选值地预设 `--enable-experimental-jit` 则默认为 `yes`。

- `no`: 禁用整个第 2 层级和 JIT 管线。
- `yes`: 启用 JIT。要在运行时禁用 JIT，则传入环境变量 `PYTHON_JIT=0`。
- `yes-off`: 构建 JIT 但默认禁用它。要在运行时启用 JIT，则传入环境变量 `PYTHON_JIT=1`。
- `interpreter`: 启用第 2 层级解释器但是禁用 JIT。可以在运行时传入 `PYTHON_JIT=0` 来禁用该解释器。

其内部架构大致如下：

- 我们将从特化的第 1 层级字节码开始。请参阅 3.11 有什么新变化了解详情。
- 当第 1 层级字节码达到足够热度，它将被翻译为新的纯内部的中间表示形式 (IR)，称为第 2 层级 IR，有时也称为微操作码 (“uops”)。
- 第 2 层级 IR 使用与第 1 层级相同的基于栈的虚拟机，但其指令格式更适合被翻译为机器码。
- 在第 2 层级 IR 被解释或翻译为机器码之前，我们会预先应用一些优化通路。

- 虽然第 2 层级解释器存在，但它主要用于对优化管线的先前阶段进行调试。可通过为 Python 配置 `--enable-experimental-jit=interpreter` 选项启用第 2 层级解释器。
- 启用 JIT 时，经优化的第 2 层级 IR 将被翻译为机器码后再执行。
- 这个机器码翻译过程使用了名为 拷贝并打补丁 的技巧。它没有运行时依赖，但增加了构建时对 LLVM 的依赖。

参见

PEP 744

(JIT 来自 Brandt Bucher 且受到 Haoran Xu 和 Fredrik Kjolstad 论文的启发。第 2 层级 IR 来自 Mark Shannon 和 Guido van Rossum。第 2 层级解释器来自 Ken Jin。)

2.5 针对 `locals()` 的已定义修改语义

在历史上，改变 `locals()` 的返回值的预期结果是留给具体的 Python 实现来定义的。从 Python 3.13 开始，**PEP 667** 标准化了 CPython 对于大多数代码执行作用域的历史行为，但也将已优化作用域（函数、生成器、协程、推导式和生成器表达式）修改为显式地返回当前已局部变量的独立快照，包括局部引用的在闭包中捕获的非局部变量。

在已优化的作用域中对 `locals()` 语义的这项修改也会影响隐式地以 `locals()` 为目标的代码执行函数的默认行为，如果没有提供显式命名空间的话（例如 `exec()` 和 `eval()` 等）。在之前的版本中，在调用代码执行函数后是否可以通过调用 `locals()` 访问更改情况取决于具体的实现。具体到 CPython 而言，此类代码通常会按预期运作，但有时可能会在基于其他代码（包括调试器和代码执行跟踪工具）的已优化作用域中失败，因为代码有可能重置该作用域中的共享快照。现在，代码在已优化作用域中将始终针对局部变量的独立快照运行，因为在后续调用 `locals()` 时将永远看不到更改。要访问在这些情况下所做的更改，现在必须将一个显式命名空间引用传递给相关的函数。或者，也可以更新受影响的代码以使用更高层级的代码执行 API 返回结果代码执行命名空间（例如，当执行磁盘上的 Python 文件时使用 `runpy.run_path()` 函数）。

为确保调试器和类似工具能可靠地更新受到此变化影响的作用域中的局部变量，现在 `FrameType.f_locals` 将返回一个针对此种作用域中的帧的局部变量和在局部引用的非局部变量的直通写入代理对象，而不是返回一个非持续更新的具有规定的运行时语义的共享 dict 实例。

请参阅 **PEP 667** 了解详情，包括相关的 C API 更改和弃用。下文还针对受影响的 *Python API* 和 *C API* 提供了移植说明。

(PEP 和实现由 Mark Shannon 和 Tian Gao 在 [gh-74929](#) 中贡献。文档更新由 Guido van Rossum 和 Alyssa Coghlan 提供。)

2.6 对移动平台的支持

PEP 730: iOS 现在是 **PEP 11** 所支持的平台，包括第 3 层级的 `arm64-apple-ios` 和 `arm64-apple-ios-simulator` 等目标（分别为 2013 年后的 iPhone 和 iPad 设备以及运行于 Apple silicon 硬件的 Xcode iOS 模拟器）。`x86_64-apple-ios-simulator`（运行于较旧的 x86_64 硬件的 Xcode iOS 模拟器）不是第 3 层级的受支持平台，但也将尽可能地支持。（PEP 撰写及实现由 Russell Keith-Magee 在 [gh-114099](#) 中贡献。）

PEP 738: Android 支持正在积极开发中，但该平台尚未得到官方支持。（PEP 撰写与实现由 Malcolm Smith 在 [gh-116622](#) 中贡献。）

参见

PEP 730, PEP 738

2.7 增量式垃圾回收

循环垃圾回收器现在是增量式的。这意味着对更大的堆进行垃圾收集的最大暂停时间会减小一个数量级以上。

现在只分两代：新的和老的。当 `gc.collect()` 不是被直接调用时，GC 被发起调用的频次将略微减少。当被发起调用时，它会回收新一代并累加老的一代，而不是回收一代或多代。

`gc.collect()` 的行为略有变化：

- `gc.collect(1)`：执行一次 GC 累加，而不是回收第 1 代。
- 其他对 `gc.collect()` 的调用没有变化。

（由 Mark Shannon 在 [gh-108362](#) 中贡献。）

3 其他语言特性修改

- 编译器现在将从文档字符串的每一行去除共有的前导空格。这会减少字节码缓存的大小（例如 `.pyc` 文件），例如在 SQLAlchemy 2.0 的 `sqlalchemy.orm.session` 中文件大小将减少约 5%。这项改变将影响各种使用了文档字符串的工具，如 `doctest`。

```
>>> def spam():
...     """
...         This is a docstring with
...         leading whitespace.
...
...         It even has multiple paragraphs!
...     """
...
>>> spam.__doc__
'\nThis is a docstring with\n  leading whitespace.\n\nIt even has multiple_\n paragraphs!\n'
```

（由 Inada Naoki 在 [gh-81283](#) 中贡献。）

- 类作用域内的标注作用域现在可以包含 `lambda` 和推导式。位于类作用域内的推导式不会内联到其父作用域中。

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

（由 Jelle Zijlstra 在 [gh-109118](#) 和 [gh-118160](#) 中贡献。）

- `future` 语句不再会被 `__future__` 模块的相对导入触发，意味着 `from __future__ import ...` 形式的语句现在只是标准的相对导入，而不会激活任何特殊特性。（由 Jeremiah Gabriel Pascual 在 [gh-118216](#) 中贡献。）
- 现在 `global` 声明当其被用于 `else` 代码块中时也将被允许在 `except` 代码块中使用。在之前版本中这会错误地引发 `SyntaxError`。（由 Irit Katriel 在 [gh-111123](#) 中贡献。）

- 增加了新的环境变量 `PYTHON_FROZEN_MODULES`，它确定冻结模块是否会被导入机制所忽略，等价于 `-X frozen_modules` 命令行选项。（由 Yilei Yang 在 [gh-111374](#) 中贡献。）
- 通过新的环境变量 `PYTHON_PERF_JIT_SUPPORT` 和命令行选项 `-X perf_jit` 添加无需 帧指针 即可工作的对 `perf` 性能分析器的支持。（由 Pablo Galindo 在 [gh-118518](#) 中贡献。）
- 可通过新的 `PYTHON_HISTORY` 环境变量来更改 `.python_history` 文件的位置。（由 Levi Sabah, Zackery Spytz 和 Hugo van Kemenade 在 [gh-73965](#) 中贡献。）
- 类新增了一个 `__static_attributes__` 属性。这由编译器以类属性名称的元组来填充，这些名称是从类体中的任何函数通过 `self.<name>` 来赋值的。（由 Irit Katriel 在 [gh-115775](#) 中贡献。）
- 编译器现在会在类上创建一个 `__firstlineno__` 属性，其值为类定义第一行的行号。（由 Serhiy Storchaka 在 [gh-118465](#) 中贡献。）
- 现在 `exec()` 和 `eval()` 内置函数接受以关键字形式传入的 `globals` 和 `locals` 参数。（由 Raphael Gaschignard 在 [gh-105879](#) 中贡献。）
- 现在 `compile()` 内置函数接受一个新的旗标 `ast.PyCF_OPTIMIZED_AST`，它类似于 `ast.PyCF_ONLY_AST` 但区别在于返回的 AST 是根据 `optimize` 参数的值进行优化的。（由 Irit Katriel 在 [gh-108113](#) 中贡献。）
- 增加了新的异常 `PythonFinalizationError`，它派生自 `RuntimeError`，用于当操作在最终化期间被阻塞时发出信号。下列可调用对象现在将引发 `PythonFinalizationError`，而不是 `RuntimeError`：
 - `_thread.start_new_thread()`
 - `os.fork()`
 - `os.forkpty()`
 - `subprocess.Popen`
 （由 Victor Stinner 在 [gh-114570](#) 中贡献。）
- 允许 `str.replace()` 的 `count` 参数为关键字参数。（由 Hugo van Kemenade 在 [gh-106487](#) 中贡献。）
- 现在许多函数会对将布尔值作为文件描述符参数发出警告。这可以帮助尽早发现一些错误。（由 Serhiy Storchaka 在 [gh-82626](#) 中贡献。）
- 为 `bz2`, `lzma`, `tarfile` 和 `zipfile` 等模块中的已压缩和已归档文件型对象添加了 `name` 和 `mode` 属性。（由 Serhiy Storchaka 在 [gh-115961](#) 中贡献。）
- 在 `property` 对象上增加了 `__name__` 属性。（由 Eugene Toder 在 [gh-101860](#) 中贡献。）

4 新增模块

- `dbm.sqlite3`: 针对 `dbm` 的 SQLite 后端。（由 Raymond Hettinger 和 Erlend E. Aasland 在 [gh-100414](#) 中贡献。）

5 改进的模块

5.1 argparse

- 为 `add_argument()` 和 `add_parser()` 方法添加了 *deprecated* 形参，以允许弃用命令行选项、位置参数和子命令。(由 Serhiy Storchaka 在 [gh-83648](#) 中贡献。)

5.2 array

- 增加了 'w' 类型码 (Py_UCS4) 表示 Unicode 字符。它应被用来代替已弃用的 'u' 类型码。(由 Inada Naoki 在 [gh-80480](#) 中贡献。)
- 通过实现 `clear()` 方法将 `array.array` 注册为 `MutableSequence`。(由 Mike Zimin 在 [gh-114894](#) 中贡献。)

5.3 ast

- 现在 `ast` 模块中节点类型的构造器对其接受的参数要求更为严格，并在参数被省略时具有更易理解的行为。

如果在构造实例时某个 AST 节点上的可选字段没有被作为参数包括在内，则该字段现在将被设为 `None`。类似地，如果某个列表字段被省略，则该字段现在将被设为空列表，而如果某个 `expr_context` 字段被省略，则它将默认为 `Load()`。(之前，在所有情况下，新构造的 AST 节点实例上的相应属性都将缺失。)

在所有其他情况下，当需要的参数被省略时，节点构造器将发出 `DeprecationWarning`。这在 Python 3.15 中将会引发异常。类似地，将关键字参数传入一个未映射到 AST 节点上的字段的构造器的做法现在已被弃用，并且在 Python 3.15 中将会引发异常。

这些更改将不会应用于用户自定义的 `ast.AST` 子类，除非该类选择通过设置 `AST._field_types` 映射的方式加入新的行为。

(由 Jelle Zijlstra 在 [gh-105858](#), [gh-117486](#) 和 [gh-118851](#) 中贡献。)

- 现在 `ast.parse()` 接受一个可选参数 *optimize*，它会被传递给 `compile()`。这使得获取已优化的 AST 成为可能。(由 Irit Katriel 在 [gh-108113](#) 中贡献。)

5.4 asyncio

- 现在 `asyncio.as_completed()` 将返回一个即是 *asynchronous iterator* 又是基本的产生可等待对象的 *iterator* 的对象。由异常迭代产生的可等待对象包括被传入的原始 `Task` 或 `Future` 对象，使得将结果与正在完成的任务相关联更为容易。(由 Justin Arthur 在 [gh-77714](#) 中贡献。)
- 现在当服务器被关闭时 `asyncio.loop.create_unix_server()` 会自动移除 Unix 套接字。(由 Pierre Ossman 在 [gh-111246](#) 中贡献。)
- 现在当附带一个空字节串对象调用时 `DatagramTransport.sendto()` 将发送零长度的数据报。现在当计算缓冲区大小时传输控制流还会将数据报包头纳入考量。(由 Jamie Phan 在 [gh-115199](#) 中贡献。)
- 增加了 `Queue.shutdown` 和 `QueueShutDown` 用于管理队列终结。(由 Laurie Opperman 和 Yves Duprat 在 [gh-104228](#) 中贡献。)
- 增加了 `Server.close_clients()` 和 `Server.abort_clients()` 方法，它们会以更强制的方式关闭 `asyncio` 服务器。(由 Pierre Ossman 在 [gh-113538](#) 中贡献。)

- 在 `StreamReader.readuntil()` 中接受一个由分隔符组成的元组, 当遇到其中之一时就会停止。(由 Bruce Merry 在 [gh-81322](#) 中贡献。)
- 改进了 `TaskGroup` 在外部的取消操作与内部的取消操作发生冲突时的行为。例如, 当嵌套两个任务分组并且两者同时在某个子任务中遇到异常时, 外层的任务分组有可能被挂起, 因为其内部的取消操作已由内层的任务分组进行处理。

对于任务分组在外部被取消时同时必须引发 `ExceptionGroup` 的情况, 现在它将调用父任务的 `cancel()` 方法。这样可以确保 `CancelledError` 会在下一次 `await` 时被引发, 因此取消操作不会丢失。 , so the cancellation is not lost.

这些更改的一个附加好处是现在任务组会保留取消操作计数 (`cancelling()`)。

为了处理某些边界情况, 现在 `uncancel()` 可以在取消操作计数达到零时重置未写入文档的 `_must_cancel` 旗标。

(受到由 Arthur Tacca 在 [gh-116720](#) 中报告的问题的启发。)

- 当在一个未激活的 `TaskGroup` 上调用 `TaskGroup.create_task()` 时, 给定的协程将被关闭 (这将防止引发有关给定的协程从未被等待的 `RuntimeWarning`)。 (由 Arthur Tacca 和 Jason Zhang 在 [gh-115957](#) 中贡献。)

5.5 base64

- 增加了 `z85encode()` 和 `z85decode()` 函数用于将 `bytes` 编码为 `Z85 data` 和将 `Z85` 编码的数据解码为 `bytes`。 (由 Matan Perelman 在 [gh-75299](#) 中贡献。)

5.6 compileall

- 工作线程和进程的默认数据现在使用的是 `os.process_cpu_count()` 而不是 `os.cpu_count()` 来选择的。 (由 Victor Stinner 在 [gh-109649](#) 中贡献。)

5.7 configparser

- `configparser.ConfigParser` 现在可以被配置为在命名部分之前接受未命名部分。 (由 Pedro Sousa Lacerda 在 [gh-66449](#) 中贡献。)

5.8 concurrent.futures

- 工作线程和进程的默认数据现在使用的是 `os.process_cpu_count()` 而不是 `os.cpu_count()` 来选择的。 (由 Victor Stinner 在 [gh-109649](#) 中贡献。)

5.9 configparser

- 现在 `ConfigParser` 具有对未命名节的支持, 这将允许使用最高层级的键值对。此特性可通过新增的 `allow_unnamed_section` 形参来启用。 (由 Pedro Sousa Lacerda 在 [gh-66449](#) 中贡献。)

5.10 copy

- 新增的 `replace()` 函数和 `replace` 协议使得创建经修改的对象副本更为简单。这在操作不可变对象时特别有用。以下类型将支持 `replace()` 函数并实现了 `replace` 协议：

- `collections.namedtuple()`
- `dataclasses.dataclass`
- `datetime.datetime, datetime.date, datetime.time`
- `inspect.Signature, inspect.Parameter`
- `types.SimpleNamespace`
- 代码对象

任何用户自定义类也可以通过定义 `__replace__()` 方法来支持 `copy.replace()`。（由 Serhiy Storchaka 在 [gh-108751](#) 中贡献。）

5.11 dbm

- 增加 `dbm.sqlite3`，一个实现了 SQLite 后端的新模块，并将其设为默认的 `dbm` 后端。（由 Raymond Hettinger 和 Erlend E. Aasland 在 [gh-100414](#) 中贡献。）
- 允许通过新增的 `gdbm.clear()` 和 `ndbm.clear()` 方法移除数据库中的所有条目。（由 Donghee Na 在 [gh-107122](#) 中贡献。）

5.12 dis

- 将 `dis` 模块的函数的输出修改为显示跳转目标和异常处理器的逻辑标签，而不是偏移量。可以使用新的 `-O` 命令行选项或 `show_offsets` 参数来添加偏移量。（由 Irit Katriel 在 [gh-112137](#) 中贡献。）
- `get_instructions()` 不再将缓存条目表示为单独的指令。作为替代，它会将它们作为 `Instruction` 的组成部分返回，放在新的 `cache_info` 字段中。传给 `get_instructions()` 的 `show_caches` 参数已被弃用并且不再有任何效果。（由 Irit Katriel 在 [gh-112962](#) 中贡献。）

5.13 doctest

- 现在 `doctest` 输出默认是彩色的。此特性可通过新增的 `PYTHON_COLORS` 环境变量和传统的 `NO_COLOR` 和 `FORCE_COLOR` 环境变量来控制。另请参阅 [using-on-controlling-color](#)。（由 Hugo van Kemenade 在 [gh-117225](#) 中贡献。）
- 现在 `DocTestRunner.run()` 方法会统计已跳过测试的数量。增加了 `DocTestRunner.skips` 和 `TestResults.skipped` 属性。（由 Victor Stinner 在 [gh-108794](#) 中贡献。）

5.14 email

- 现在带有嵌入的换行符的标头在输出时会加上引号。现在 `generator` 会拒绝序列化（写入）不正确地折叠或分隔的标头，例如将被解析为多个标头或与相邻数据合并的标头等。如果你需要禁用此安全特性，请设置 `verify_generated_headers`。（由 Bas Bloemsaat 和 Petr Viktorin 在 [gh-121650](#) 中贡献。）
- 现在 `getaddresses()` 和 `parseaddr()` 会在更多遇到无效 email 地址的情况下返回 `('', '')` 对而不是返回可能不准确的值。这两个函数新增了可选的 `strict` 形参（默认为 `True`）。要获取旧版本的行为（接受错误格式的输入），请使用 `strict=False`。可以使用 `getattr(email.utils, 'supports_strict_parsing', False)` 来检查 `strict` 形参是否可用。（由 Thomas Dwyer 和 Victor Stinner 针对 [gh-102988](#) 贡献以改进 [CVE-2023-27043](#) 修正。）

5.15 fractions

- 现在 `Fraction` 对象支持用于填充、对齐、正负号处理、最小宽度和分组的标准格式说明迷你语言规则。（由 Mark Dickinson 在 [gh-111320](#) 中贡献。）

5.16 gc

循环垃圾回收器现在是递增式的，这改变了 `get_threshold()` 和 `set_threshold()` 以及 `get_count()` 和 `get_stats()` 的结果的具体含义。

- 为保持向下兼容，`get_threshold()` 将继续返回一个由三项组成的元组。第一项的值是新收集项的阈值，与之前一样；第二项的值决定旧收集项被扫描的频度（默认为 10，更高的值意味着旧收集项被扫描的频度更慢）。第三项的值没有意义且始终为零。
- `set_threshold()` 会忽略第二项之后的任何项。
- `get_count()` 和 `get_stats()` 将继续返回相同格式的结果。唯一的区别在于结果不再是指年轻代、中年代和老年代，现在结果是指向年轻代和中年代以及老年代的回收空间。

总而言之，尝试操纵循环 GC 的行为的代码可能无法完全如预期一样有效，但也不太可能产生什么损害。所有其他代码都会正常执行。

5.17 glob

- 增加了 `translate()`，这是个用来将具有 shell 风格通配符的路径说明转换为正则表达式的函数。（由 Barney Gale 在 [gh-72904](#) 中贡献。）

5.18 importlib

- 现在 `importlib.resources` 中的下列函数允许访问资源目录（或树），并使用多个位置参数（现在文本读取函数中的 `encoding` 和 `errors` 参数是仅限关键字参数）：

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

这些函数将不再被弃用也不会被加入移除计划。(由 Petr Viktorin 在 [gh-106532](#) 中贡献。)

- `contents()` 仍然被弃用而应改用功能完整的 `Traversable` API。不过，目前还没有移除它的计划。(由 Petr Viktorin 在 [gh-106532](#) 中贡献。)

5.19 io

- 现在 `IOBase` 最终化器会使用 `sys.unraisablehook` 来将由 `close()` 方法引发的错误写入日志。在之前版本中，错误在默认情况下会被静默地忽略，而只有在 `Python` 开发模式或在使用 `Python` 调试构建版时才会被写入日志。(由 Victor Stinner 在 [gh-62948](#) 中贡献。)

5.20 ipaddress

- 增加了 `IPv4Address.ipv6_mapped` 特征属性，它将返回映射 `IPv4` 的 `IPv6` 地址。(由 Charles Machalow 在 [gh-109466](#) 中贡献。)
- 修正了 `IPv4Address`, `IPv6Address`, `IPv4Network` 和 `IPv6Network` 中 `is_global` 和 `is_private` 的行为。(由 Jakub Stasiak 在 [gh-113171](#) 中贡献。)

5.21 itertools

- `batched()` 新增了 `strict` 形参，它会在最后一批次数据小于指定批准大小时引发 `ValueError`。(由 Raymond Hettinger 在 [gh-113202](#) 中贡献。)

5.22 marshal

- 在模块函数中增加了 `allow_code` 形参。传入 `allow_code=False` 将防止在 `Python` 各版本间不兼容的代码对象的序列化和反序列化。(由 Serhiy Storchaka 在 [gh-113626](#) 中贡献。)

5.23 math

- 新增函数 `fma()` 可执行合并的乘法-加法运算。此函数只需一轮操作即可计算 $x * y + z$ ，从而避免了任何中间步骤导致的精度损失。它包装了 `C99` 所提供的 `fma()` 函数，并且遵从针对特殊情况的 `IEEE 754 "fusedMultiplyAdd"` 运算规范。(由 Mark Dickinson 和 Victor Stinner 在 [gh-73468](#) 中贡献。)

5.24 mimetypes

- 增加了 `guess_file_type()` 函数用于根据文件系统路径来猜测 `MIME` 类型。在 `guess_type()` 中使用路径的做法现在已是 `soft deprecated`。(由 Serhiy Storchaka 在 [gh-66543](#) 中贡献。)

5.25 mmap

- 现在 `mmap` 在 Windows 上当被映射的内存由于文件系统错误或访问限制而不可访问时将获得保护以避免崩溃。(由 Jannis Weigend 在 [gh-118209](#) 中贡献。)
- `mmap` 具有新的 `seekable()` 方法将在需要可定位的文件型对象时被使用。现在 `seek()` 方法将返回一个新的绝对位置。(由 Donghee Na 和 Sylvie Liberman 在 [gh-111835](#) 中贡献。)
- `mmap` 新增了 UNIX 专属的 `trackfd` 形参用来控制文件描述符的复制；如为假值，则由 `fileno` 指定的文件描述符将不会被复制。(由 Zackery Spytz 和 Petr Viktorin 在 [gh-78502](#) 中贡献。)

5.26 multiprocessing

- 工作线程和进程的默认数据现在是使用 `os.process_cpu_count()` 而不是 `os.cpu_count()` 来选择的。(由 Victor Stinner 在 [gh-109649](#) 中贡献。)

5.27 os

- 增加了 `process_cpu_count()` 函数用于获取当前进程的调用方线程可以使用的逻辑 CPU 核心数量。(由 Victor Stinner 在 [gh-109649](#) 中贡献。)
- `cpu_count()` 和 `process_cpu_count()` 可通过新的环境变量 `PYTHON_CPU_COUNT` 或新的命令行选项 `-X cpu_count` 来覆盖。此选项对于需要在不修改应用程序代码或容器本身的情况下限制一个容器系统的 CPU 资源的用户会很有用处。(由 Donghee Na 在 [gh-109595](#) 中贡献。)
- 通过 `timerfd_create()`, `timerfd_settime()`, `timerfd_gettime_ns()`, `timerfd_gettime()`, `timerfd_gettime_ns()`, `TFD_NONBLOCK`, `TFD_CLOEXEC`, `TFD_TIMER_ABSTIME` 和 `TFD_TIMER_CANCEL_ON_SET` 增加了针对 Linux 的 计算器文件描述符 的低层级接口。(由 Masaru Tsuchiyama 在 [gh-108277](#) 中贡献。)
- 在 Windows 上现在同时增加了对 `lchmod()` 和 `chmod()` 的 `follow_symlinks` 参数的支持。请注意在 Windows 上 `lchmod()` 中的 `follow_symlinks` 的默认值为 `False`。(由 Serhiy Storchaka 在 [gh-59616](#) 中贡献。)
- 在 Windows 上现在同时增加了 `fchmod()` 和对 `chmod()` 中的文件描述符的支持。(由 Serhiy Storchaka 在 [gh-113191](#) 中贡献。)
- 在 Windows 上, `mkdir()` 和 `makedirs()` 现在支持传入 `mode` 值 `0o700` 以对新目录应用访问控制。这会隐式地影响 `tempfile.mkdtemp()` 并可缓解 [CVE-2024-4030](#)。其他的 `mode` 值仍然会被忽略。(由 Steve Dower 在 [gh-118486](#) 中贡献。)
- 现在 `posix_spawn()` 可接受 `None` 作为 `env` 参数，这将让新产生的进程使用当前进程的环境。(由 Jakub Kulik 在 [gh-113119](#) 中贡献。)
- 在支持 `posix_spawn_file_actions_addclosefrom_np()` 的平台上 `posix_spawn()` 现在可以在 `file_actions` 形参中使用 `POSIX_SPAWN_CLOSEFROM` 属性。(由 Jakub Kulik 在 [gh-113117](#) 中贡献。)

5.28 os.path

- 增加了 `isreserved()` 用于检查一个路径在当前系统中是否为保留路径。此函数仅在 Windows 上可用。(由 Barney Gale 在 [gh-88569](#) 中贡献。)
- 在 Windows 上, `isabs()` 将不再把以恰好一个斜杠(\ 或 /) 开头的路径视为绝对路径。(由 Barney Gale 和 Jon Foster 在 [gh-44626](#) 中贡献。)
- 现在即使文件不可访问 `realpath()` 也能够解析 MS-DOS 风格的文件名。(由 Moonsik Park 在 [gh-82367](#) 中贡献。)

5.29 pathlib

- 增加了 `UnsupportedOperation`, 它会在一个路径操作不受支持时代替 `NotImplementedError` 被引发。(由 Barney Gale 在 [gh-89812](#) 中贡献。)
- 新增了一个用于根据 'file' URI (`file:///`) 来创建 `Path` 对象的构造器 `Path.from_uri()`。(由 Barney Gale 在 [gh-107465](#) 中贡献。)
- 增加了 `PurePath.full_match()` 用于匹配带有 shell 风格通配符的路径, 包括递归通配符 `***`。(由 Barney Gale 在 [gh-73435](#) 中贡献。)
- 增加了 `PurePath.parser` 类属性以存储用于低层级路径解析与合并的 `os.path` 实现。这可以是 `posixpath` 或 `ntpath`。
- 为 `Path.glob()` 和 `rglob()` 增加了 `recurse_symlinks` 仅限关键字参数。(由 Barney Gale 在 [gh-77609](#) 中贡献。)
- 现在当给出以 `***` 结束的模式时 `Path.glob()` 和 `rglob()` 将返回文件和目录。在之前版本中, 仅会返回目录。(由 Barney Gale 在 [gh-70303](#) 中贡献。)
- 为 `Path.is_file`, `Path.is_dir`, `Path.owner()` 和 `Path.group()` 增加了 `follow_symlinks` 仅限关键字参数。(由 Barney Gale 在 [gh-105793](#) 中, 以及 Kamil Turek 在 [gh-107962](#) 中贡献。)

5.30 pdb

- 现在 `breakpoint()` 和 `set_trace()` 会立即进入调试器而不是在被执行代码的下一行进入。这一更改可防止当 `breakpoint()` 位于上下文末尾时调试器在上下文以外被中断。(由 Tian Gao 在 [gh-118579](#) 中贡献。)
- 当设置了 `sys.flags.safe_path` 时 `sys.path[0]` 将不会再被替换为被调试脚本的目录。(由 Tian Gao 和 Christian Walther 在 [gh-111762](#) 中贡献。)
- 现在支持将 `zipapp` 作为调试目标。(由 Tian Gao 在 [gh-118501](#) 中贡献。)
- 添加了在 `pm()` 中进行事后调试期间使用 `Pdb` 新增的 `exceptions [exc_number]` 命令在串连的异常之间移动的能力。(由 Matthias Bussonnier 在 [gh-106676](#) 中贡献。)
- 以一条 `pdb` 命令打头的表达式和语句现在会被正确地标识并执行。(由 Tian Gao 在 [gh-108464](#) 中贡献。)

5.31 queue

- 增加了 `Queue.shutdown` 和 `ShutDown` 用于管理队列的终结。(由 Laurie Opperman 和 Yves Duprat 在 [gh-104750](#) 中贡献。)

5.32 random

- 增加了一个 命令行接口。(由 Hugo van Kemenade 在 [gh-118131](#) 中贡献。)

5.33 re

- 将 `re.error` 重命名为 `PatternError` 以改善准确性。`re.error` 仍被保留用于向下兼容。

5.34 shutil

- 在 `chown()` 中增加了对 `dir_fd` 和 `follow_symlinks` 关键字参数的支持。(由 Berker Peksag 和 Tahia K 在 [gh-62308](#) 中贡献。)

5.35 site

- 现在 `.pth` 文件将先使用 UTF-8 来解码，如果 UTF-8 解码失败再使用 `locale encoding`。(由 Inada Naoki 在 [gh-117802](#) 中贡献。)

5.36 sqlite3

- 现在当一个 `Connection` 对象未被显式地 关闭时将发出 `ResourceWarning`。(由 Erlend E. Aasland 在 [gh-105539](#) 中贡献。)
- 为 `Connection.iterdump()` 增加了 `filter` 仅限关键字形参用于过滤要转储的数据库对象。(由 Mariusz Felisiak 在 [gh-91602](#) 中贡献。)

5.37 ssl

- 现在 `create_default_context()` API 将在其默认旗标中包括 `VERIFY_X509_PARTIAL_CHAIN` 和 `VERIFY_X509_STRICT`。

备注

`VERIFY_X509_STRICT` 可能会拒绝下层 OpenSSL 实现本来会接受的 **RFC 5280** 以前的证书或格式错误的证书。虽然不建议禁用此功能，但你可以使用以下方式禁用它：

```
import ssl

ctx = ssl.create_default_context()
ctx.verify_flags &= ~ssl.VIFY_X509_STRICT
```

(由 William Woodruff 在 [gh-112389](#) 贡献。)

5.38 statistics

- 增加了用于核密度估计的 `kde()`。这使得根据固定数量的离散样本估计连续概率密度函数成为可能。(由 Raymond Hettinger 在 [gh-115863](#) 中贡献。)
- 增加了 `kde_random()` 用来从 `kde()` 创建的估计概率密度函数进行取样。(由 Hettinger 在 [gh-115863](#) 中贡献。)

5.39 subprocess

- 现在 `subprocess` 模块会在更多场合下使用 `posix_spawn()` 函数。

Notably, when `close_fds` is `True` (the default), `posix_spawn()` will be used when the C library provides `posix_spawn_file_actions_addclosefrom_np()`, which includes recent versions of Linux, FreeBSD, and Solaris. On Linux, this should perform similarly to the existing Linux `vfork()` based code.

A private control knob `subprocess._USE_POSIX_SPAWN` can be set to `False` if you need to force `subprocess` to never use `posix_spawn()`. Please report your reason and platform details in the issue tracker if you set this so that we can improve our API selection logic for everyone. (Contributed by Jakub Kulik in [gh-113117](#).)

5.40 sys

- 增加了 `_is_interned()` 函数用于检测字符串是否被内部化。此函数不保证在所有的 Python 实现中均存在。(由 Serhiy Storchaka 在 [gh-78573](#) 中贡献。)

5.41 tempfile

- 在 Windows 上, `tempfile.mkdtemp()` 所使用的默认模式 `0o700` 由于 `os.mkdir()` 的更改现在将限制对新目录的访问。这是对 [CVE-2024-4030](#) 的缓解措施。(由 Steve Dower 在 [gh-118486](#) 中贡献。)

5.42 time

- 在 Windows 上, `monotonic()` 现在将使用精度为 1 微秒的 `QueryPerformanceCounter()` 时钟, 而不是精度只有 15.6 毫秒的 `GetTickCount64()` 时钟。(由 Victor Stinner 在 [gh-88494](#) 中贡献。)
- 在 Windows 上, `time()` 现在将使用精度为 1 微秒的 `GetSystemTimePreciseAsFileTime()` 时钟, 代替精度为 15.6 毫秒的 `GetSystemTimeAsFileTime()` 时钟。(由 Victor Stinner 在 [gh-63207](#) 中贡献。)

5.43 tkinter

- 增加了 tkinter 控件方法: `tk_busy_hold()`, `tk_busy_configure()`, `tk_busy_cget()`, `tk_busy_forget()`, `tk_busy_current()` 和 `tk_busy_status()`。(由 Miguel, klappnase 和 Serhiy Storchaka 在 [gh-72684](#) 中贡献。)
- The tkinter widget method `wm_attributes()` now accepts the attribute name without the minus prefix to get window attributes, for example `w.wm_attributes('alpha')` and allows specifying attributes and values to set as keyword arguments, for example `w.wm_attributes(alpha=0.5)`. (Contributed by Serhiy Storchaka in [gh-43457](#).)

- `wm_attributes()` can now return attributes as a `dict`, by using the new optional keyword-only parameter `return_python_dict`. (Contributed by Serhiy Storchaka in [gh-43457](#).)
- `Text.count()` can now return a simple `int` when the new optional keyword-only parameter `return_ints` is used. Otherwise, the single count is returned as a 1-tuple or `None`. (Contributed by Serhiy Storchaka in [gh-97928](#).)
- 在 `tkinter.ttk.Style` 的 `element_create()` 方法中增加了对“`vsapi`”元素类型的支持。(由 Serhiy Storchaka 在 [gh-68166](#) 中贡献。)
- 为 Tkinter 的控件增加了 `after_info()` 方法。(由 Cheryl Sabella 在 [gh-77020](#) 中贡献。)
- 为 `PhotoImage` 新增 `copy_replace()` 方法用于将一个图像的某个区域拷贝到另一个图像，可能带有像素缩放、子采样，或两者皆有。(由 Serhiy Storchaka 在 [gh-118225](#) 中贡献。)
- 为 `PhotoImage` 的方法 `copy()`, `zoom()` 和 `subsample()` 增加了 `from_coords` 形参。为 `PhotoImage` 的方法 `copy()` 增加了 `zoom` 和 `subsample` 形参。(由 Serhiy Storchaka 在 [gh-118225](#) 中贡献。)
- 增加了 `PhotoImage` 方法 `read()` 用于从文件读取图像以及 `data()` 用于获取图像数据。为 `write()` 方法增加了 `background` 和 `grayscale` 形参。(由 Serhiy Storchaka 在 [gh-118271](#) 中贡献。)

5.44 回溯

- Add the `exc_type_str` attribute to `TracebackException`, which holds a string display of the `exc_type`. Deprecate the `exc_type` attribute, which holds the type object itself. Add parameter `save_exc_type` (default `True`) to indicate whether `exc_type` should be saved. (Contributed by Irit Katriel in [gh-112332](#).)
- Add a new `show_group` keyword-only parameter to `TracebackException.format_exception_only()` to (recursively) format the nested exceptions of a `BaseExceptionGroup` instance. (Contributed by Irit Katriel in [gh-105292](#).)

5.45 types

- `SimpleNamespace` can now take a single positional argument to initialise the namespace’s arguments. This argument must either be a mapping or an iterable of key-value pairs. (Contributed by Serhiy Storchaka in [gh-108191](#).)

5.46 typing

- **PEP 705**: Add `ReadOnly`, a special typing construct to mark a `TypedDict` item as read-only for type checkers.
- **PEP 742**: Add `TypeIs`, a typing construct that can be used to instruct a type checker how to narrow a type.
- Add `NoDefault`, a sentinel object used to represent the defaults of some parameters in the `typing` module. (Contributed by Jelle Zijlstra in [gh-116126](#).)
- Add `get_protocol_members()` to return the set of members defining a `typing.Protocol`. (Contributed by Jelle Zijlstra in [gh-104873](#).)
- Add `is_protocol()` to check whether a class is a `Protocol`. (Contributed by Jelle Zijlstra in [gh-104873](#).)
- `ClassVar` can now be nested in `Final`, and vice versa. (Contributed by Mehdi Drissi in [gh-89547](#).)

5.47 unicodedata

- Update the Unicode database to [version 15.1.0](#). (Contributed by James Gerity in [gh-109559](#).)

5.48 venv

- Add support for creating source control management (SCM) ignore files in a virtual environment's directory. By default, Git is supported. This is implemented as opt-in via the API, which can be extended to support other SCMs (`EnvBuilder` and `create()`), and opt-out via the CLI, using `--without-scm-ignore-files`. (Contributed by Brett Cannon in [gh-108125](#).)

5.49 warnings

- **PEP 702**: The new `warnings.deprecated()` decorator provides a way to communicate deprecations to a static type checker and to warn on usage of deprecated classes and functions. A `DeprecationWarning` may also be emitted when a decorated function or class is used at runtime. (Contributed by Jelle Zijlstra in [gh-104003](#).)

5.50 xml

- 通过添加五个新方法允许控制 Expat $\geq 2.6.0$ 重解析延迟 ([CVE-2023-52425](#)):
 - `xml.etree.ElementTree.XMLParser.flush()`
 - `xml.etree.ElementTree.XMLPullParser.flush()`
 - `xml.parsers.expat.xmlparser.GetReparseDeferralEnabled()`
 - `xml.parsers.expat.xmlparser.SetReparseDeferralEnabled()`
 - `xml.sax.expatreader.ExpatParser.flush()`(由 Sebastian Pipping 在 [gh-115623](#) 中贡献。)
- Add the `close()` method for the iterator returned by `iterparse()` for explicit cleanup. (Contributed by Serhiy Storchaka in [gh-69893](#).)

5.51 zipimport

- Add support for [ZIP64](#) format files. Everybody loves huge data, right? (Contributed by Tim Hatch in [gh-94146](#).)

6 性能优化

- The new *incremental garbage collector* means that maximum pause times are reduced by an order of magnitude or more for larger heaps. (Contributed by Mark Shannon in [gh-108362](#).)
- Several standard library modules have had their import times significantly improved. For example, the import time of the `typing` module has been reduced by around a third by removing dependencies on `re` and `contextlib`. Other modules to enjoy import-time speedups include `email.utils`, `enum`, `functools`, `importlib.metadata`, and `threading`. (Contributed by Alex Waygood, Shantanu Jain, Adam Turner, Daniel Hollas, and others in [gh-109653](#).)
- `textwrap.indent()` is now around 30% faster than before for large input. (Contributed by Inada Naoki in [gh-107369](#).)

- The `subprocess` module now uses the `posix_spawn()` function in more situations, including when `close_fds` is `True` (the default) on many modern platforms. This should provide a notable performance increase when launching processes on FreeBSD and Solaris. See the [subprocess](#) section above for details. (Contributed by Jakub Kulik in [gh-113117](#).)

7 被移除的模块与 API

7.1 PEP 594: Remove “dead batteries” from the standard library

PEP 594 proposed removing 19 modules from the standard library, colloquially referred to as ‘dead batteries’ due to their historic, obsolete, or insecure status. All of the following modules were deprecated in Python 3.11, and are now removed:

- `aifc`
- `audioop`
- `chunk`
- `cgi` 和 `cgitb`
 - `cgi.FieldStorage` can typically be replaced with `urllib.parse.parse_qs()` for GET and HEAD requests, and the `email.message` module or the [multipart](#) library for POST and PUT requests.
 - `cgi.parse()` can be replaced by calling `urllib.parse.parse_qs()` directly on the desired query string, unless the input is `multipart/form-data`, which should be replaced as described below for `cgi.parse_multipart()`.
 - `cgi.parse_header()` can be replaced with the functionality in the `email` package, which implements the same MIME RFCs. For example, with `email.message.EmailMessage`:

```
from email.message import EmailMessage

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

- `cgi.parse_multipart()` can be replaced with the functionality in the `email` package, which implements the same MIME RFCs, or with the [multipart](#) library. For example, the `email.message.EmailMessage` and `email.message.Message` classes.
- `crypt` and the private `_crypt` extension. The `hashlib` module may be an appropriate replacement when simply hashing a value is required. Otherwise, various third-party libraries on PyPI are available:
 - `bcrypt`: 用于软件和服务器的现代密码哈希算法。
 - `passlib`: 支持超过 over 30 种方案的综合密码哈希算法框架。
 - `argon2-cffi`: 安全的 Argon2 密码哈希算法。
 - `legacycrypt`: 针对 POSIX 加密库调用和相关功能的 `ctypes` 包装器。
 - `crypt_r`: `crypt` 模块的分叉，针对 `crypt_r(3)` 库调用和相关功能和包装器。
- `imghdr`: The [filetype](#), [puremagic](#), or [python-magic](#) libraries should be used as replacements. For example, the `puremagic.what()` function can be used to replace the `imghdr.what()` function for all file formats that were supported by `imghdr`.
- `mailcap`: 改用 `mimetypes` 模块。
- `msilib`

- `nis`
- `nntplib`: 改用 PyPI 上的 `nntplib` 库。
- `ossaudiodev`: 对于音频回放, 改用 PyPI 上的 `pygame` 库。
- `pipes`: 改用 `subprocess` 模块。
- `sndhdr`: 应当使用 `filetype`, `puremagic` 或 `python-magic` 库作为替代。
- `spwd`: Use the `python-pam` library from PyPI instead.
- `sunau`
- `telnetlib`, 改用 PyPI 上的 `telnetlib3` 或 `Exscript` 库。
- `uu`: 改用 the `base64` 模块, 作为一款现代化的替代。
- `xdrlib`

(由 Victor Stinner 和 Zachary Ware 在 [gh-104773](#) 和 [gh-104780](#) 中贡献。)

7.2 2to3

- Remove the `2to3` program and the `lib2to3` module, previously deprecated in Python 3.11. (Contributed by Victor Stinner in [gh-104780](#).)

7.3 builtins

- Remove support for chained `classmethod` descriptors (introduced in [gh-63272](#)). These can no longer be used to wrap other descriptors, such as `property`. The core design of this feature was flawed and led to several problems. To "pass-through" a `classmethod`, consider using the `__wrapped__` attribute that was added in Python 3.10. (Contributed by Raymond Hettinger in [gh-89519](#).)

7.4 configparser

- Remove the undocumented `LegacyInterpolation` class, deprecated in the docstring since Python 3.2, and at runtime since Python 3.11. (Contributed by Hugo van Kemenade in [gh-104886](#).)

7.5 importlib.metadata

- Remove deprecated subscript (`__getitem__()`) access for `EntryPoint` objects. (Contributed by Jason R. Coombs in [gh-113175](#).)

7.6 locale

- Remove the `locale.resetlocale()` function, deprecated in Python 3.11. Use `locale.setlocale(locale.LC_ALL, "")` instead. (Contributed by Victor Stinner in [gh-104783](#).)

7.7 opcode

- Move `opcode.ENABLE_SPECIALIZATION` to `_opcode.ENABLE_SPECIALIZATION`. This field was added in 3.12, it was never documented, and is not intended for external use. (Contributed by Irit Katriel in [gh-105481](#).)
- Remove `opcode.is_pseudo()`, `opcode.MIN_PSEUDO_OPCODE`, and `opcode.MAX_PSEUDO_OPCODE`, which were added in Python 3.12, but were neither documented nor exposed through `dis`, and were not intended to be used externally. (Contributed by Irit Katriel in [gh-105481](#).)

7.8 pathlib

- Remove the ability to use `Path` objects as context managers. This functionality was deprecated and has had no effect since Python 3.9. (Contributed by Barney Gale in [gh-83863](#).)

7.9 re

- Remove the undocumented, deprecated, and broken `re.template()` function and `re.TEMPLATE / re.T` flag. (Contributed by Serhiy Storchaka and Nikita Sobolev in [gh-105687](#).)

7.10 tkinter.tix

- Remove the `tkinter.tix` module, deprecated in Python 3.6. The third-party Tix library which the module wrapped is unmaintained. (Contributed by Zachary Ware in [gh-75552](#).)

7.11 turtle

- Remove the `RawTurtle.settiltangle()` method, deprecated in the documentation since Python 3.1 and at runtime since Python 3.11. (Contributed by Hugo van Kemenade in [gh-104876](#).)

7.12 typing

- Remove the `typing.io` and `typing.re` namespaces, deprecated since Python 3.8. The items in those namespaces can be imported directly from the `typing` module. (Contributed by Sebastian Rittau in [gh-92871](#).)
- Remove the keyword-argument method of creating `TypedDict` types, deprecated in Python 3.11. (Contributed by Tomas Roun in [gh-104786](#).)

7.13 unittest

- 移除了下列 `unittest` 函数，它们在 Python 3.11 中已被弃用：

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

请改用 `TestLoader` 方法：

- `loadTestsFromModule()`

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

(由 Hugo van Kemenade 在 [gh-104835](#) 中贡献。)

- Remove the untested and undocumented `TestProgram.usageExit()` method, deprecated in Python 3.11. (Contributed by Hugo van Kemenade in [gh-104992](#).)

7.14 urllib

- Remove the *cafile*, *capath*, and *cadefault* parameters of the `urllib.request.urlopen()` function, deprecated in Python 3.6. Use the *context* parameter instead with an `SSLContext` instance. The `ssl.SSLContext.load_cert_chain()` function can be used to load specific certificates, or let `ssl.create_default_context()` select the operating system's trusted certificate authority (CA) certificates. (Contributed by Victor Stinner in [gh-105382](#).)

7.15 webbrowser

- Remove the untested and undocumented `MacOSX` class, deprecated in Python 3.11. Use the `MacOSXOSAScript` class (introduced in Python 3.2) instead. (Contributed by Hugo van Kemenade in [gh-104804](#).)
- Remove the deprecated `MacOSXOSAScript._name` attribute. Use the `MacOSXOSAScript.name` attribute instead. (Contributed by Nikita Sobolev in [gh-105546](#).)

8 新的弃用

- **array**: `array` 的 'u' 格式码, 自 Python 3.3 起在文档中声明弃用, 自 3.13 开始发出 `DeprecationWarning` 并将在 Python 3.16 中移除。请改用 'w' 格式码。(由 Hugo van Kemenade 在 [gh-80480](#) 中贡献。)
- **ctypes**: 弃用未写入文档的 `ctypes.SetPointerType()` 函数。软弃用 `ctypes.ARRAY()` 函数并改用乘法。(由 Victor Stinner 在 [gh-105733](#) 中贡献。)
- **decimal**: 弃用 `decimal.Decimal` 的非标准格式说明符 "N"。它未被写入文档并且仅在 C 实现中受支持。(由 Serhiy Storchaka 在 [gh-89902](#) 中贡献。)
- **dis**: `dis.HAVE_ARGUMENT` 分隔符已被弃用。改为在 `hasarg` 中检查成员。(由 Irit Katriel 在 [gh-109319](#) 中贡献。)
- **frame-objects**: Calling `frame.clear()` on a suspended frame raises `RuntimeError` (as has always been the case for an executing frame). (Contributed by Irit Katriel in [gh-79932](#).)
- **getopt and optparse modules**: They are now soft deprecated: the `argparse` module should be used for new projects. Previously, the `optparse` module was already deprecated, its removal was not scheduled, and no warnings was emitted: so there is no change in practice. (Contributed by Victor Stinner in [gh-106535](#).)
- **gettext**: Emit deprecation warning for non-integer numbers in `gettext` functions and methods that consider plural forms even if the translation was not found. (Contributed by Serhiy Storchaka in [gh-88434](#).)
- **glob**: The undocumented `glob.glob0()` and `glob.glob1()` functions are deprecated. Use `glob.glob()` and pass a directory to its *root_dir* argument instead. (Contributed by Barney Gale in [gh-117337](#).)

- `http.server`: `http.server.CGIHTTPRequestHandler` now emits a `DeprecationWarning` as it will be removed in 3.15. Process-based CGI HTTP servers have been out of favor for a very long time. This code was outdated, unmaintained, and rarely used. It has a high potential for both security and functionality bugs. This includes removal of the `--cgi` flag to the `python -m http.server` command line in 3.15.
- `mimetypes`: Passing file path instead of URL in `guess_type()` is soft deprecated. Use `guess_file_type()` instead. (Contributed by Serhiy Storchaka in [gh-66543](#).)
- `re`: Passing optional arguments `maxsplit`, `count` and `flags` in module-level functions `re.split()`, `re.sub()` and `re.subn()` as positional arguments is now deprecated. In future Python versions these parameters will be keyword-only. (Contributed by Serhiy Storchaka in [gh-56166](#).)
- `pathlib`: `pathlib.PurePath.is_reserved()` 已被弃用并计划在 Python 3.15 中移除。请使用 `os.path.isreserved()` 来检测 Windows 上的保留路径。
- `platform`: `java_ver()` 已被弃用并将在 3.15 中移除。它几乎未经测试，具有令人困惑的 API，并且仅适用于 Jython 支持。(由 Nikita Sobolev 在 [gh-116349](#) 中贡献。)
- `pydoc`: 弃用未记载的 `pydoc.ispackage()` 方法。(由 Zackery Spytz 在 [gh-64020](#) 中贡献。)
- `sqlite3`: 向 `sqlite3.connect()` 和 `sqlite3.Connection` 构造器传入多个位置参数的做法已被弃用。其他的形参在 Python 3.15 中将成为仅限关键字形参。

Deprecate passing name, number of arguments, and the callable as keyword arguments for the following `sqlite3.Connection` APIs:

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

已弃用通过关键字参数形式为下列 `sqlite3.Connection` API 传入可调用对象:

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

受影响的形参将在 Python 3.15 中成为仅限位置形参。

(由 Erlend E. Aasland 在 [gh-107948](#) 和 [gh-108278](#) 中贡献。)

- `sys`: The `sys._enablelegacywindowsfsencoding()` function is deprecated. Replace it with the `PYTHONLEGACYWINDOWSFSENCODING` environment variable. (Contributed by Inada Naoki in [gh-73427](#).)
- `tarfile`: 在 `tarfile.TarFile` 类中未记载亦未使用的 `tarfile` 属性现已弃用，并计划于 Python 3.16 版本中移除。
- `traceback`: The field `exc_type` of `traceback.TracebackException` is deprecated. Use `exc_type_str` instead.
- `typing`:
 - Creating a `typing.NamedTuple` class using keyword arguments to denote the fields (`NT = NamedTuple("NT", x=int, y=int)`) is deprecated, and will be disallowed in Python 3.15. Use the class-based syntax or the functional syntax instead. (Contributed by Alex Waygood in [gh-105566](#).)
 - When using the functional syntax to create a `typing.NamedTuple` class or a `typing.TypedDict` class, failing to pass a value to the 'fields' parameter (`NT = NamedTuple("NT")` or `TD = TypedDict("TD")`) is deprecated. Passing `None` to the 'fields' parameter (`NT = NamedTuple("NT", None)` or `TD = TypedDict("TD", None)`) is also deprecated. Both will be disallowed in Python 3.15. To create a `NamedTuple` class with zero fields, use `class NT(NamedTuple):` pass or `NT = NamedTuple("NT", [])`. To create a `TypedDict` class with

zero fields, use `class TD(TypedDict): pass` or `TD = TypedDict("TD", {})`. (Contributed by Alex Waygood in [gh-105566](#) and [gh-105570](#).)

- `typing.no_type_check_decorator()` is deprecated, and scheduled for removal in Python 3.15. After eight years in the `typing` module, it has yet to be supported by any major type checkers. (Contributed by Alex Waygood in [gh-106309](#).)
- `typing.AnyStr` is deprecated. In Python 3.16, it will be removed from `typing.__all__`, and a `DeprecationWarning` will be emitted when it is imported or accessed. It will be removed entirely in Python 3.18. Use the new type parameter syntax instead. (Contributed by Michael The in [gh-107116](#).)
- user-defined-funcs: Assignment to a function's `__code__` attribute where the new code object's type does not match the function's type is deprecated. The different types are: plain function, generator, async generator and coroutine. (Contributed by Irit Katriel in [gh-81137](#).)
- wave: 已弃用 `wave.Wave_read` 和 `wave.Wave_write` 类的 `getmark()`, `setmark()` 和 `getmarkers()` 方法。它们将在 Python 3.15 中被移除。(由 Victor Stinner 在 [gh-105096](#) 中贡献。)

8.1 计划在 Python 3.14 中移除

- argparse: `argparse.BooleanOptionalAction` 的 *type*, *choices* 和 *metavar* 形参已被弃用并将在 3.14 中移除。(由 Nikita Sobolev 在 [gh-92248](#) 中贡献。)
- ast: 以下特性自 Python 3.8 起已在文档中声明弃用，现在当运行时如果它们被访问或使用时将发出 `DeprecationWarning`，并将在 Python 3.14 中移除：

- `ast.Num`
- `ast.Str`
- `ast.Bytes`
- `ast.NameConstant`
- `ast.Ellipsis`

请改用 `ast.Constant`。(由 Serhiy Storchaka 在 [gh-90953](#) 中贡献。)

- asyncio:
 - 子监视器类 `MultiLoopChildWatcher`, `FastChildWatcher`, `AbstractChildWatcher` 和 `SafeChildWatcher` 已被弃用并将在 Python 3.14 中移除。(由 Kumar Aditya 在 [gh-94597](#) 中贡献。)
 - `asyncio.set_child_watcher()`、`asyncio.get_child_watcher()`、`asyncio.AbstractEventLoopPolicy.set_child_watcher()` 和 `asyncio.AbstractEventLoopPolicy.get_child_watcher()` 已弃用，并将在 Python 3.14 中移除。(由 Kumar Aditya 在 [gh-94597](#) 中贡献。)
 - 现在默认事件循环策略的 `get_event_loop()` 方法在当前事件循环未设置并决定创建一个时将发出 `DeprecationWarning`。(由 Serhiy Storchaka 和 Guido van Rossum 在 [gh-100160](#) 中贡献。)
- collections.abc: 已弃用 `ByteString`。推荐改用 `Sequence` 或 `Buffer`。用于类型标注时，则推荐并集运算符，如 `bytes | bytearray`，或 `collections.abc.Buffer`。(由 Shantanu Jain 在 [gh-91896](#) 中贡献。)
- email: 已弃用 `email.utils.localtime()` 中的 *isdst* 形参。(由 Alan Williams 在 [gh-72346](#) 中贡献。)
- importlib: `__package__` 和 `__cached__` 将不再被设置或是被导入系统纳入考量 ([gh-97879](#))。)
- importlib.abc 中已弃用的类：
 - `importlib.abc.ResourceReader`

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

使用 `importlib.resources.abc` 类代替:

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

(由 Jason R. Coombs 和 Hugo van Kemenade 在 [gh-93963](#) 中贡献。)

- `itertools` 具有对 `copy`, `deepcopy` 和 `pickle` 等操作的未写入文档的、低效的、历史上充满问题的且不支持稳定的支持。这将在 3.14 中移除以显著减少代码量和维护负担。(由 Raymond Hettinger 在 [gh-101588](#) 中贡献。)
- `multiprocessing`: 默认的启动方法在目前默认使用 `'fork'` 的 Linux, BSD 和其他非 macOS POSIX 平台上将改为更安全的方法 ([gh-84559](#))。为此添加运行时警告将带来糟糕的体验因为大部分代码并不会关心这个问题。当你的代码需要 `'fork'` 时请使用 `get_context()` 或 `set_start_method()` API 来显式地指明。参见 `multiprocessing-start-methods`。
- `pathlib`: `is_relative_to()` 和 `relative_to()`: 传入额外参数的做法已被弃用。
- `pkgutil`: 现在 `find_loader()` 和 `get_loader()` 将引发 `DeprecationWarning`; 请改用 `importlib.util.find_spec()`。(由 Nikita Sobolev 在 [gh-97850](#) 中贡献。)
- `pty`:
 - `master_open()`: 使用 `pty.openpty()`。
 - `slave_open()`: 使用 `pty.openpty()`。
- `sqlite3`:
 - `version` 和 `version_info`。
 - 如果使用了命名占位符且 `parameters` 是一个序列而不是 `dict` 则选择 `execute()` 和 `executemany()`。
 - `date` 和 `datetime` 适配器, `date` 和 `timestamp` 转换器: 请参阅 `sqlite3` 文档了解推荐的替代方案。
- `types.CodeType`: 访问 `co_lnotab` 的做法自 3.10 起已在 [PEP 626](#) 中被弃用并曾计划在 3.12 中移除, 但实际上在 3.12 中仅设置了 `DeprecationWarning`。可能会在 3.14 中移除。(由 Nikita Sobolev 在 [gh-101866](#) 中贡献。)
- `typing`: `ByteString` 自 Python 3.9 起已被弃用, 现在当被使用时将会发出 `DeprecationWarning`。
- `urllib`: `urllib.parse.Quoter` 已被弃用: 它不应被作为公有 API。(由 Gregory P. Smith 在 [gh-88168](#) 中贡献。)

8.2 Python 3.15 中的待移除功能

- `http.server.CGIHTTPRequestHandler` 将同 `python -m http.server` 中与其相关的 `--cgi` 旗标一起被移除。它已经过时并且很少被使用。不存在直接的替代品。对于建立 Web 服务器与请求处理器的接口的任务来说任何东西都比 CGI 要好。
- `locale`: `locale.getdefaultlocale()` 在 Python 3.11 中已被弃用并且原计划在 Python 3.13 中移除 ([gh-90817](#)), 但移除时间已被推迟至 Python 3.15。请改用 `locale.setlocale()`, `locale.getencoding()` 和 `locale.getlocale()`。(由 Hugo van Kemenade 在 [gh-111187](#) 中贡献。)
- `pathlib`: `pathlib.PurePath.is_reserved()` 已被弃用并计划在 Python 3.15 中移除。请使用 `os.path.isreserved()` 来检测 Windows 上的保留路径。

- `platform.java_ver()` 已被弃用并将在 3.15 中移除。它几乎未经测试，具有令人困惑的 API，并且仅适用于 Jython 支持。（由 Nikita Sobolev 在 [gh-116349](#) 中贡献。）
- `threading`: 向 `threading.RLock()` 传入任意参数的做法现已被弃用。C 版本允许任意数量的 `args` 和 `kwargs`，但它们都会被忽略。Python 版本不允许任何参数。在 Python 3.15 中将从 `threading.RLock()` 移除所有参数。（由 Nikita Sobolev 在 [gh-102029](#) 中贡献。）
- `typing.NamedTuple`:
 - 创建 `NamedTuple` 类 (`NT = NamedTuple("NT", x=int)`) 的关键字参数语法从未写入文档且已被弃用，并将在 3.15 中被禁止。请改用基于类的语法或函数语法。
 - 当使用函数式语法创建 `NamedTuple` 类时，不向 *fields* 形参传值的做法 (`NT = NamedTuple("NT")`) 已被弃用。将 `None` 传给 *fields* 形参的做法 (`NT = NamedTuple("NT", None)`) 也已被弃用。两者都将在 Python 3.15 中被禁止。要创建拥有 0 个字段的 `NamedTuple` 类，请使用 `class NT(NamedTuple): pass` 或 `NT = NamedTuple("NT", [])`。
- `typing.TypedDict`: 当使用函数语法创建 `TypedDict` 类时，不向 *fields* 形参传值 (`TD = TypedDict("TD")`) 的行为已被弃用。将 `None` 传给 *fields* 形参 (`TD = TypedDict("TD", None)`) 的行为也已被弃用。两者都将在 Python 3.15 中被禁止。要创建拥有 0 个字段的 `TypedDict` 类，请使用 `class TD(TypedDict): pass` 或 `TD = TypedDict("TD", {})`。
- `wave`: 已弃用 `wave.Wave_read` 和 `wave.Wave_write` 类的 `getmark()`、`setmark()` 和 `getmarkers()` 方法。它们将在 Python 3.15 中被移除。（由 Victor Stinner 在 [gh-105096](#) 中贡献。）

8.3 计划在 Python 3.16 中移除

- `array`: `array.array 'u'` 类型 (`wchar_t`): 改用 `'w'` 类型 (`Py_UCS4`)。
- `builtins`: `~bool`，对布尔值按位取反。
- `symtable`: `symtable.Class.get_methods()` 因缺少适当的场景已被弃用。（由 Bénédict Tran 在 [gh-119698](#) 中贡献。）

8.4 计划在未来版本中移除

以下 API 将会被移除，尽管具体时间还未确定。

- `argparse`: 嵌套参数分组和嵌套互斥分组的做法已被弃用。
- `array` 的 `'u'` 格式代码 ([gh-57281](#))
- `builtins`:
 - `bool(NotImplemented)`。
 - 生成器: `throw(type, exc, tb)` 和 `athrow(type, exc, tb)` 签名已被弃用: 请改用 `throw(exc)` 和 `athrow(exc)`，即单参数签名。
 - 目前 Python 接受数字类字面值后面紧跟关键字的写法，例如 `0 in x, 1 or x, 0 if 1 else 2`。它允许像 `[0x1 for x in y]` 这样令人困惑且有歧义的表达式 (它可以被解读为 `[0x1 for x in y]` 或者 `[0x1f or x in y]`)。如果数字类字面值后面紧跟关键字 `and`, `else`, `for`, `if`, `in`, `is` 和 `or` 中的一个将会引发语法警告。在未来的版本中它将改为语法错误。([gh-87999](#))
 - 对 `__index__()` 和 `__int__()` 方法返回非 `int` 类型的支持: 将要求这些方法必须返回 `int` 的子类的实例。
 - 对 `__float__()` 方法返回 `float` 的子类的支持: 将要求这些方法必须返回 `float` 的实例。

- 对 `__complex__()` 方法返回 `complex` 的子类的支持：将要求这些方法必须返回 `complex` 的实例。
- 将 `int()` 委托给 `__trunc__()` 方法。
- 传入一个复数作为 `complex()` 构造器中的 *real* 或 *imag* 参数的做法现在已被弃用；它应当仅作为单个位置参数被传入。（由 Serhiy Storchaka 在 [gh-109218](#) 中贡献。）
- `calendar`: `calendar.January` 和 `calendar.February` 常量已被弃用并由 `calendar.JANUARY` 和 `calendar.FEBRUARY` 替代。（由 Prince Roshan 在 [gh-103636](#) 中贡献。）
- `codeobject.co_lnotab`: 改用 `codeobject.co_lines()` 方法。
- `datetime`:
 - `utcnow()`: 使用 `datetime.datetime.now(tz=datetime.UTC)`。
 - `utcfromtimestamp()`: 使用 `datetime.datetime.fromtimestamp(timestamp, tz=datetime.UTC)`。
- `gettext`: 复数值必须是一个整数。
- `importlib`:
 - `load_module()` 方法：改用 `exec_module()`。
 - `cache_from_source()` *debug_override* 形参已被弃用：改用 *optimization* 形参。
- `importlib.metadata`:
 - `EntryPoints` 元组接口。
 - 返回值中隐式的 `None`。
- `logging`: `warn()` 方法自 Python 3.3 起已被弃用，请改用 `warning()`。
- `mailbox`: 对 `StringIO` 输入和文本模式的使用已被弃用，改用 `BytesIO` 和二进制模式。
- `os`: 在多线程的进程中调用 `os.register_at_fork()`。
- `pydoc.ErrorDuringImport`: 使用元组值作为 *exc_info* 形参的做法已被弃用，应使用异常实例。
- `re`: 现在对于正则表达式中的数字分组引用和分组名称将应用更严格的规则。现在只接受 ASCII 数字序列作为数字引用。字节串模式和替换字符串中的分组名称现在只能包含 ASCII 字母和数字以及下划线。（由 Serhiy Storchaka 在 [gh-91760](#) 中贡献。）
- `sre_compile`, `sre_constants` 和 `sre_parse` 模块。
- `shutil`: `rmtree()` 的 *onerror* 形参在 Python 3.12 中已被弃用；请改用 *onexc* 形参。
- `ssl` 选项和协议：
 - `ssl.SSLContext` 不带 `protocol` 参数的做法已被弃用。
 - `ssl.SSLContext`: `set_npn_protocols()` 和 `selected_npn_protocol()` 已被弃用：请改用 `ALPN`。
 - `ssl.OP_NO_SSL*` 选项
 - `ssl.OP_NO_TLS*` 选项
 - `ssl.PROTOCOL_SSLv3`
 - `ssl.PROTOCOL_TLS`
 - `ssl.PROTOCOL_TLSv1`
 - `ssl.PROTOCOL_TLSv1_1`

- `ssl.PROTOCOL_TLSv1_2`
- `ssl.TLSVersion.SSLv3`
- `ssl.TLSVersion.TLSv1`
- `ssl.TLSVersion.TLSv1_1`
- `sysconfig.is_python_build()` *check_home* 形参已被弃用并会被忽略。
- `threading` 的方法:
 - `threading.Condition.notifyAll()`: 使用 `notify_all()`。
 - `threading.Event.isSet()`: 使用 `is_set()`。
 - `threading.Thread.isDaemon()`, `threading.Thread.setDaemon()`: 使用 `threading.Thread.daemon` 属性。
 - `threading.Thread.getName()`, `threading.Thread.setName()`: 使用 `threading.Thread.name` 属性。
 - `threading.currentThread()`: 使用 `threading.current_thread()`。
 - `threading.activeCount()`: 使用 `threading.active_count()`。
- `typing.Text` ([gh-92332](#))。
- `unittest.IsolatedAsyncioTestCase`: 从测试用例返回不为 `None` 的值的做法已被弃用。
- `urllib.parse` 函数已被弃用: 改用 `urlparse()`
 - `splitattr()`
 - `splithost()`
 - `splitnport()`
 - `splitpasswd()`
 - `splitport()`
 - `splitquery()`
 - `splittag()`
 - `splitttype()`
 - `splituser()`
 - `splitvalue()`
 - `to_bytes()`
- `urllib.request`: 发起请求的 `URLopener` 和 `FancyURLopener` 方式已被弃用。改用更新 `urlopen()` 函数和方法。
- `wsgiref.SimpleHandler.stdout.write()` 不应执行部分写入。
- `xml.etree.ElementTree`: 对 `Element` 的真值测试已被弃用。在未来的发布版中它将始终返回 `True`。建议改用显式的 `len(elem)` 或 `elem is not None` 测试。
- `zipimport.zipimporter.load_module()` 已被弃用: 请改用 `exec_module()`。

9 CPython 字节码的变化

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

10 C API 的变化

10.1 新的特性

- You no longer have to define the PY_SSIZE_T_CLEAN macro before including `Python.h` when using `#` formats in format codes. APIs accepting the format codes always use `Py_ssize_t` for `#` formats. (Contributed by Inada Naoki in [gh-104922](#).)
- The `keywords` parameter of `PyArg_ParseTupleAndKeywords()` and `PyArg_VaParseTupleAndKeywords()` now has type `char *const*` in C and `const char *const*` in C++, instead of `char**`. It makes these functions compatible with arguments of type `const char *const*`, `const char**` or `char *const*` in C++ and `char *const*` in C without an explicit type cast. This can be overridden with the `PY_CXX_CONST` macro. (Contributed by Serhiy Storchaka in [gh-65210](#).)
- Add `PyImport_AddModuleRef()`: similar to `PyImport_AddModule()`, but return a strong reference instead of a borrowed reference. (Contributed by Victor Stinner in [gh-105922](#).)
- Add `PyWeakref_GetRef()` function: similar to `PyWeakref_GetObject()` but returns a strong reference, or NULL if the referent is no longer live. (Contributed by Victor Stinner in [gh-105927](#).)
- Add `PyObject_GetOptionalAttr()` and `PyObject_GetOptionalAttrString()`, variants of `PyObject_GetAttr()` and `PyObject_GetAttrString()` which don't raise `AttributeError` if the attribute is not found. These variants are more convenient and faster if the missing attribute should not be treated as a failure. (Contributed by Serhiy Storchaka in [gh-106521](#).)
- Add `PyMapping_GetOptionalItem()` and `PyMapping_GetOptionalItemString()`: variants of `PyObject_GetItem()` and `PyMapping_GetItemString()` which don't raise `KeyError` if the key is not found. These variants are more convenient and faster if the missing key should not be treated as a failure. (Contributed by Serhiy Storchaka in [gh-106307](#).)
- 增加了静默地忽略错误的函数的已修正变体形式：
 - `PyObject_HasAttrWithError()` 替代 `PyObject_HasAttr()`。
 - `PyObject_HasAttrStringWithError()` 替代 `PyObject_HasAttrString()`。
 - `PyMapping_HasKeyWithError()` 替代 `PyMapping_HasKey()`。
 - `PyMapping_HasKeyStringWithError()` 替代 `PyMapping_HasKeyString()`。

新的函数不仅返回 1 表示真值和 0 表示假值，还返回 -1 表示错误。

(由 Serhiy Storchaka 在 [gh-108511](#) 中贡献。)

- 如果 Python 是使用 调试模式或 附带断言构建的，`PyTuple_SET_ITEM()` 和 `PyList_SET_ITEM()` 现在将通过一个断言来检查 `index` 参数。(由 Victor Stinner 在 [gh-106168](#) 中贡献。)
- 增加了 `PyModule_Add()` 函数：类似于 `PyModule_AddObjectRef()` 和 `PyModule_AddObject()` 但总是会偷取一个对值的引用。(由 Serhiy Storchaka 在 [gh-86493](#) 中贡献。)

- Add `PyDict_GetItemRef()` and `PyDict_GetItemStringRef()` functions: similar to `PyDict_GetItemWithError()` but returning a strong reference instead of a borrowed reference. Moreover, these functions return -1 on error and so checking `PyErr_Occurred()` is not needed. (Contributed by Victor Stinner in [gh-106004](#).)
- Added `PyDict_SetDefaultRef()`, which is similar to `PyDict_SetDefault()` but returns a strong reference instead of a borrowed reference. This function returns -1 on error, 0 on insertion, and 1 if the key was already present in the dictionary. (Contributed by Sam Gross in [gh-112066](#).)
- Add `PyDict_ContainsString()` function: same as `PyDict_Contains()`, but `key` is specified as a `const char*` UTF-8 encoded bytes string, rather than a `PyObject*`. (Contributed by Victor Stinner in [gh-108314](#).)
- Added `PyList_GetItemRef()` function: similar to `PyList_GetItem()` but returns a strong reference instead of a borrowed reference.
- Add `Py_IsFinalizing()` function: check if the main Python interpreter is shutting down. (Contributed by Victor Stinner in [gh-108014](#).)
- Add `PyLong_AsInt()` function: similar to `PyLong_AsLong()`, but store the result in a C `int` instead of a C `long`. Previously, it was known as the private function `_PyLong_AsInt()` (with an underscore prefix). (Contributed by Victor Stinner in [gh-108014](#).)
- Python built with `configure --with-trace-refs` (tracing references) now supports the Limited API. (Contributed by Victor Stinner in [gh-108634](#).)
- Add `PyObject_VisitManagedDict()` and `PyObject_ClearManagedDict()` functions which must be called by the traverse and clear functions of a type using `Py_TPFLAGS_MANAGED_DICT` flag. The [pythoncapi-compat](#) project can be used to get these functions on Python 3.11 and 3.12. (Contributed by Victor Stinner in [gh-107073](#).)
- Add `PyUnicode_EqualToUTF8AndSize()` and `PyUnicode_EqualToUTF8()` functions: compare Unicode object with a `const char*` UTF-8 encoded string and return true (1) if they are equal, or false (0) otherwise. These functions do not raise exceptions. (Contributed by Serhiy Storchaka in [gh-110289](#).)
- Add `PyThreadState_GetUnchecked()` function: similar to `PyThreadState_Get()`, but don't kill the process with a fatal error if it is NULL. The caller is responsible to check if the result is NULL. Previously, the function was private and known as `_PyThreadState_UncheckedGet()`. (Contributed by Victor Stinner in [gh-108867](#).)
- Add `PySys_AuditTuple()` function: similar to `PySys_Audit()`, but pass event arguments as a Python tuple object. (Contributed by Victor Stinner in [gh-85283](#).)
- `PyArg_ParseTupleAndKeywords()` now supports non-ASCII keyword parameter names. (Contributed by Serhiy Storchaka in [gh-110815](#).)
- Add `PyMem_RawMalloc()`, `PyMem_RawCalloc()`, `PyMem_RawRealloc()` and `PyMem_RawFree()` to the limited C API (version 3.13). (Contributed by Victor Stinner in [gh-85283](#).)
- Add `PySys_Audit()` and `PySys_AuditTuple()` functions to the limited C API. (Contributed by Victor Stinner in [gh-85283](#).)
- Add `PyErr_FormatUnraisable()` function: similar to `PyErr_WriteUnraisable()`, but allow customizing the warning message. (Contributed by Serhiy Storchaka in [gh-108082](#).)
- Add `PyList_Extend()` and `PyList_Clear()` functions: similar to Python `list.extend()` and `list.clear()` methods. (Contributed by Victor Stinner in [gh-111138](#).)
- Add `PyDict_Pop()` and `PyDict_PopString()` functions: remove a key from a dictionary and optionally return the removed value. This is similar to `dict.pop()`, but without the default value and not raising `KeyError` if the key is missing. (Contributed by Stefan Behnel and Victor Stinner in [gh-111262](#).)

- 增加了 `Py_HashPointer()` 函数用于对指针执行哈希运算。(由 Victor Stinner 在 [gh-111545](#) 中贡献。)
- 增加了实现 Python 对象的默认哈希函数的 `PyObject_GenericHash()` 函数。(由 Serhiy Storchaka 在 [gh-113024](#) 中贡献。)
- 新增 PyTime C API:
 - `PyTime_t` 类型。
 - `PyTime_MIN` 和 `PyTime_MAX` 常量。
 - 新增函数:
 - * `PyTime_AsSecondsDouble()`。
 - * `PyTime_Monotonic()`。
 - * `PyTime_MonotonicRaw()`。
 - * `PyTime_PerfCounter()`。
 - * `PyTime_PerfCounterRaw()`。
 - * `PyTime_Time()`。
 - * `PyTime_TimeRaw()`。

(由 Victor Stinner 和 Petr Viktorin 在 [gh-110850](#) 中贡献。)
- 增加 `PyLong_AsNativeBytes()`, `PyLong_FromNativeBytes()` 和 `PyLong_FromUnsignedNativeBytes()` 等函数以简化原生整数类型和 Python `int` 对象之间的转换。(由 Steve Dower 在 [gh-111140](#) 中贡献。)
- 增加 `PyType_GetFullyQualifiedName()` 函数以获取类型的完整限定名称。等价于 `f"{type.__module__}.{type.__qualname__}"`, 或者如果 `type.__module__` 不是字符串或等于 `"builtins"` 则等价于 `type.__qualname__`。(由 Victor Stinner 在 [gh-111696](#) 中贡献。)
- Add `PyType_GetModuleName()` function to get the type's module name. Equivalent to getting the `type.__module__` attribute. (Contributed by Eric Snow and Victor Stinner in [gh-111696](#).)
- Add support for `%T`, `%#T`, `%N` and `%#N` formats to `PyUnicode_FromFormat()`: format the fully qualified name of an object type and of a type: call `PyType_GetModuleName()`. See [PEP 737](#) for more information. (Contributed by Victor Stinner in [gh-111696](#).)
- Add `Py_GetConstant()` and `Py_GetConstantBorrowed()` functions to get constants. For example, `Py_GetConstant(Py_CONSTANT_ZERO)` returns a strong reference to the constant zero. (Contributed by Victor Stinner in [gh-115754](#).)
- Add `PyType_GetModuleByDef()` to the limited C API (Contributed by Victor Stinner in [gh-116936](#).)
- Add two new functions to the C-API, `PyRefTracer_SetTracer()` and `PyRefTracer_GetTracer()`, that allow to track object creation and destruction the same way the `tracemalloc` module does. (Contributed by Pablo Galindo in [gh-93502](#).)
- Add `PyEval_GetFrameBuiltins()`, `PyEval_GetFrameGlobals()`, and `PyEval_GetFrameLocals()` to the C API. These replacements for `PyEval_GetBuiltins()`, `PyEval_GetGlobals()`, and `PyEval_GetLocals()` return strong references rather than borrowed references. (Added as part of [PEP 667](#).)
- 增加了 `PyMutex` API, 它是占用一个字节的轻量级互斥锁。当操作需要阻塞时 `PyMutex_Lock()` 函数将释放 (当前持有的) GIL。(由 Sam Gross 在 [gh-108724](#) 中贡献。)

11 构建变化

- 现在 `configure` 选项 `--with-system-libmpdec` 默认为 `yes`。捆绑的 `libmpdecimal` 副本将在 Python 3.15 中被移除。
- 现在需要有 `autoconf 2.71` 和 `aclocal 1.16.4` 才能重新生成 `configure` 脚本。（由 Christian Heimes 在 [gh-89886](#) 中贡献。）
- 需要 SQLite 3.15.2 或更新的版本才能构建 `sqlite3` 扩展模块。（由 Erlend Aasland 在 [gh-105875](#) 中贡献。）
- 使用 `configure --with-trace-refs` (跟踪引用) 构建的 Python 现在与 Python 发布构建版和 调试构建版是 ABI 兼容的。（由 Victor Stinner 在 [gh-108634](#) 中贡献。）
- 现在构建 CPython 需要带有 C11 atomic 库支持的编译器、GCC 内置 atomic 函数或 MSVC 互锁内生函数。
- 现在 `errno`, `fcntl`, `grp`, `md5`, `pwd`, `resource`, `termios`, `winsound`, `_ctypes_test`, `_multiprocessing.posixshm`, `_scproxy`, `_stat`, `_statistics`, `_testconsole`, `_testimportmultiple` 和 `_uuid` C 扩展是使用受限 C API 构建的。（由 Victor Stinner 在 [gh-85283](#) 中贡献。）
- 现在 `wasm32-wasi` 是 **PEP 11** 第 2 层级的平台。（由 Brett Cannon 在 [gh-115192](#) 中贡献。）
- `wasm32-emscrip` 不再是 **PEP 11** 的受支持平台。（由 Brett Cannon 在 [gh-115192](#) 中贡献。）
- Python now bundles the [mimalloc library](#). It is licensed under the MIT license; see [mimalloc license](#). The bundled mimalloc has custom changes, see [gh-113141](#) for details. (Contributed by Dino Viehland in [gh-109914](#).)
- On POSIX systems, the `pkg-config (.pc)` filenames now include the ABI flags. For example, the free-threaded build generates `python-3.13t.pc` and the debug build generates `python-3.13d.pc`.

12 Porting to Python 3.13

本节列出了先前描述的更改以及可能需要更改代码的其他错误修正。

12.1 Python API 的变化

- An `OSError` is now raised by `getpass.getuser()` for any failure to retrieve a username, instead of `ImportError` on non-Unix platforms or `KeyError` on Unix platforms where the password database is empty.
- 现在 `threading` 模块会预期 `_thread` 模块具有 `_is_main_interpreter` 属性。它是一个不带参数的函数并会在当前解释器为主解释器时返回 `True`。
Any library or application that provides a custom `_thread` module must provide `_is_main_interpreter()`, just like the module's other "private" attributes. (See [gh-112826](#).)
- `mailbox.Maildir` now ignores files with a leading dot. (Contributed by Zackery Spytz in [gh-65559](#).)
- `pathlib.Path.glob()` and `rglob()` now return both files and directories if a pattern that ends with `"**"` is given, rather than directories only. Users may add a trailing slash to match only directories.
- The value of the `mode` attribute of `gzip.GzipFile` was changed from integer (1 or 2) to string ('rb' or 'wb'). The value of the `mode` attribute of the readable file-like object returned by `zipfile.ZipFile.open()` was changed from 'r' to 'rb'. (Contributed by Serhiy Storchaka in [gh-115961](#).)

- `functools.partial` now emits a `FutureWarning` when it is used as a method. Its behavior will be changed in future Python versions. Wrap it in `staticmethod()` if you want to preserve the old behavior. (Contributed by Serhiy Storchaka in [gh-121027](#).)
- Calling `locals()` in an optimized scope now produces an independent snapshot on each call, and hence no longer implicitly updates previously returned references. Obtaining the legacy CPython behaviour now requires explicit calls to update the initially returned dictionary with the results of subsequent calls to `locals()`. Code execution functions that implicitly target `locals()` (such as `exec` and `eval`) must be passed an explicit namespace to access their results in an optimized scope. (Changed as part of [PEP 667](#).)
- Calling `locals()` from a comprehension at module or class scope (including via `exec` or `eval`) once more behaves as if the comprehension were running as an independent nested function (i.e. the local variables from the containing scope are not included). In Python 3.12, this had changed to include the local variables from the containing scope when implementing [PEP 709](#). (Changed as part of [PEP 667](#).)
- Accessing `FrameType.f_locals` in an optimized scope now returns a write-through proxy rather than a snapshot that gets updated at ill-specified times. If a snapshot is desired, it must be created explicitly with `dict` or the proxy's `.copy()` method. (Changed as part of [PEP 667](#).)

12.2 C API 的变化

- `Python.h` no longer includes the `<ieeefp.h>` standard header. It was included for the `finite()` function which is now provided by the `<math.h>` header. It should now be included explicitly if needed. Remove also the `HAVE_IEEEFP_H` macro. (Contributed by Victor Stinner in [gh-108765](#).)
- `Python.h` no longer includes these standard header files: `<time.h>`, `<sys/select.h>` and `<sys/time.h>`. If needed, they should now be included explicitly. For example, `<time.h>` provides the `clock()` and `gmtime()` functions, `<sys/select.h>` provides the `select()` function, and `<sys/time.h>` provides the `futimes()`, `gettimeofday()` and `setitimer()` functions. (Contributed by Victor Stinner in [gh-108765](#).)
- On Windows, `Python.h` no longer includes the `<stddef.h>` standard header file. If needed, it should now be included explicitly. For example, it provides `offsetof()` function, and `size_t` and `ptrdiff_t` types. Including `<stddef.h>` explicitly was already needed by all other platforms, the `HAVE_STDDEF_H` macro is only defined on Windows. (Contributed by Victor Stinner in [gh-108765](#).)
- If the `Py_LIMITED_API` macro is defined, `Py_BUILD_CORE`, `Py_BUILD_CORE_BUILTIN` and `Py_BUILD_CORE_MODULE` macros are now undefined by `<Python.h>`. (Contributed by Victor Stinner in [gh-85283](#).)
- The old trashcan macros `Py_TRASHCAN_SAFE_BEGIN` and `Py_TRASHCAN_SAFE_END` were removed. They should be replaced by the new macros `Py_TRASHCAN_BEGIN` and `Py_TRASHCAN_END`.

A `tp_dealloc` function that has the old macros, such as:

```
static void
mytype_dealloc(mytype *p)
{
    PyObject_GC_UnTrack(p);
    Py_TRASHCAN_SAFE_BEGIN(p);
    ...
    Py_TRASHCAN_SAFE_END
}
```

应当按照以下方式迁移到新版宏:


```
static void
mytype_dealloc(mytype *p)
{
    PyObject_GC_UnTrack(p);
    Py_TRASHCAN_BEGIN(p, mytype_dealloc)
    ...
    Py_TRASHCAN_END
}
```

Note that `Py_TRASHCAN_BEGIN` has a second argument which should be the deallocation function it is in. The new macros were added in Python 3.8 and the old macros were deprecated in Python 3.11. (Contributed by Irit Katriel in [gh-105111](#).)

- **Functions** `PyDict_GetItem()`, `PyDict_GetItemString()`, `PyMapping_HasKey()`, `PyMapping_HasKeyString()`, `PyObject_HasAttr()`, `PyObject_HasAttrString()`, and `PySys_GetObject()`, which clear all errors which occurred when calling them, now report them using `sys.unraisablehook()`. You may replace them with other functions as recommended in the documentation. (Contributed by Serhiy Storchaka in [gh-106672](#).)
- `PyCode_GetFirstFree()` is an unstable API now and has been renamed to `PyUnstable_Code_GetFirstFree()`. (Contributed by Bogdan Romanyuk in [gh-115781](#).)
- The effects of mutating the dictionary returned from `PyEval_GetLocals()` in an optimized scope have changed. New dict entries added this way will now *only* be visible to subsequent `PyEval_GetLocals()` calls in that frame, as `PyFrame_GetLocals()`, `locals()`, and `FrameType.f_locals` no longer access the same underlying cached dictionary. Changes made to entries for actual variable names and names added via the write-through proxy interfaces will be overwritten on subsequent calls to `PyEval_GetLocals()` in that frame. The recommended code update depends on how the function was being used, so refer to the deprecation notice on the function for details. (Changed as part of [PEP 667](#).)
- Calling `PyFrame_GetLocals()` in an optimized scope now returns a write-through proxy rather than a snapshot that gets updated at ill-specified times. If a snapshot is desired, it must be created explicitly (e.g. with `PyDict_Copy()`) or by calling the new `PyEval_GetFrameLocals()` API. (Changed as part of [PEP 667](#).)
- `PyFrame_FastToLocals()` and `PyFrame_FastToLocalsWithError()` no longer have any effect. Calling these functions has been redundant since Python 3.11, when `PyFrame_GetLocals()` was first introduced. (Changed as part of [PEP 667](#).)
- `PyFrame_LocalsToFast()` no longer has any effect. Calling this function is redundant now that `PyFrame_GetLocals()` returns a write-through proxy for optimized scopes. (Changed as part of [PEP 667](#).)

12.3 Removed C APIs

- Remove many APIs (functions, macros, variables) with names prefixed by `_Py` or `_PY` (considered as private API). If your project is affected by one of these removals and you consider that the removed API should remain available, please open a new issue to request a public C API and add `cc @vstinner` to the issue to notify Victor Stinner. (Contributed by Victor Stinner in [gh-106320](#).)
- Remove functions deprecated in Python 3.9:
 - `PyEval_CallObject()`, `PyEval_CallObjectWithKeywords()`: use `PyObject_CallNoArgs()` or `PyObject_Call()` instead. **Warning:** `PyObject_Call()` positional arguments must be a tuple and must not be `NULL`, keyword arguments must be a dict or `NULL`, whereas removed functions checked arguments type and accepted `NULL` positional and keyword arguments. To replace `PyEval_CallObjectWithKeywords(func, NULL, kwargs)` with `PyObject_Call()`, pass an empty tuple as positional arguments using `PyTuple_New(0)`.

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

(Contributed by Victor Stinner in [gh-105107](#).)

- Remove old buffer protocols deprecated in Python 3.0. Use `bufferobjects` instead.

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

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

- `PyObject_AsWriteBuffer()`: Use `PyObject_GetBuffer()` and `PyBuffer_Release()` instead:

```
Py_buffer view;
if (PyObject_GetBuffer(obj, &view, PyBUF_WRITABLE) < 0) {
    return NULL;
}
// Use `view.buf` and `view.len` to write to the buffer.
PyBuffer_Release(&view);
```

(Contributed by Inada Naoki in [gh-85275](#).)

- Remove the following old functions to configure the Python initialization, deprecated in Python 3.11:

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

Use the new `PyConfig` API of the Python Initialization Configuration instead ([PEP 587](#)), added to Python 3.8. (Contributed by Victor Stinner in [gh-105145](#).)

- Remove `PyEval_ThreadsInitialized()` function, deprecated in Python 3.9. Since Python 3.7, `Py_Initialize()` always creates the GIL: calling `PyEval_InitThreads()` does nothing and `PyEval_ThreadsInitialized()` always returned non-zero. (Contributed by Victor Stinner in [gh-105182](#).)

- Remove `PyEval_AcquireLock()` and `PyEval_ReleaseLock()` functions, deprecated in Python 3.2. They didn't update the current thread state. They can be replaced with:
 - `PyEval_SaveThread()` and `PyEval_RestoreThread()`;
 - low-level `PyEval_AcquireThread()` and `PyEval_RestoreThread()`;
 - or `PyGILState_Ensure()` and `PyGILState_Release()`.
 (Contributed by Victor Stinner in [gh-105182](#).)
- Remove private `_PyObject_FastCall()` function: use `PyObject_Vectorcall()` which is available since Python 3.8 ([PEP 590](#)). (Contributed by Victor Stinner in [gh-106023](#).)
- Remove `cpython/pytime.h` header file: it only contained private functions. (Contributed by Victor Stinner in [gh-106316](#).)
- Remove `_PyInterpreterState_Get()` alias to `PyInterpreterState_Get()` which was kept for backward compatibility with Python 3.8. The [pythoncapi-compat](#) project can be used to get `PyInterpreterState_Get()` on Python 3.8 and older. (Contributed by Victor Stinner in [gh-106320](#).)
- The `PyModule_AddObject()` function is now soft deprecated: `PyModule_Add()` or `PyModule_AddObjectRef()` functions should be used instead. (Contributed by Serhiy Storchaka in [gh-86493](#).)
- 从受限 C API 移除了未写入文档的 `PY_TIMEOUT_MAX` 常量。(由 Victor Stinner 在 [gh-110014](#) 中贡献。)

12.4 已弃用的 C API

- 已弃用旧的 `Py_UNICODE` 和 `Py_UNICODE_TYPE` 类型: 改为直接使用 `wchar_t` 类型。自 Python 3.3 起, `Py_UNICODE` 和 `Py_UNICODE_TYPE` 就只是 `wchar_t` 的别名。(由 Victor Stinner 在 [gh-105156](#) 中贡献。)
- 已弃用旧的 Python 初始化函数:
 - `PySys_ResetWarnOptions()`: 改为清除 `sys.warnoptions` 和 `warnings.filters`。
 - `Py_GetExecPrefix()`: 改为获取 `sys.exec_prefix`。
 - `Py_GetPath()`: 改为获取 `sys.path`。
 - `Py_GetPrefix()`: 改为获取 `sys.prefix`。
 - `Py_GetProgramFullPath()`: 改为获取 `sys.executable`。
 - `Py_GetProgramName()`: 改为获取 `sys.executable`。
 - `Py_GetPythonHome()`: 改为获取 `PyConfig.home` 或 `PYTHONHOME` 环境变量。

计划在 Python 3.15 中移除的函数。(由 Victor Stinner 在 [gh-105145](#) 中贡献。)

- 已弃用 `PyImport_ImportModuleNoBlock()` 函数, 它自 Python 3.3 起就只是 `PyImport_ImportModule()` 的别名。计划在 Python 3.15 中移除。(由 Victor Stinner 在 [gh-105396](#) 中贡献。)
- 已弃用返回 borrowed reference 的 `PyWeakref_GetObject()` 和 `PyWeakref_GET_OBJECT()` 函数: 改用新的返回 strong reference 的 `PyWeakref_GetRef()` 函数。在 Python 3.12 或更的旧版本中可以使用 [pythoncapi-compat](#) 项目 来获取 `PyWeakref_GetRef()`。(由 Victor Stinner 在 [gh-105927](#) 中贡献。)
- 已弃用返回 borrowed reference 的 `PyEval_GetBuiltins()`, `PyEval_GetGlobals()` 和 `PyEval_GetLocals()` 函数。请参阅每个函数的弃用消息来了解其推荐的替代物。(作为 [PEP 667](#) 的一部分被软弃用。)

计划在 Python 3.14 中移除

- PyDictObject 中的 `ma_version_tag` 字段用于扩展模块 ([PEP 699](#); [gh-101193](#))。
- 创建 immutable types 的可变基础 ([gh-95388](#))。
- 用于配置 Python 的初始化的函数，在 Python 3.11 中已弃用：

- `PySys_SetArgvEx()`: 改为设置 `PyConfig.argv`。
- `PySys_SetArgv()`: 改为设置 `PyConfig.argv`。
- `Py_SetProgramName()`: 改为设置 `PyConfig.program_name`。
- `Py_SetPythonHome()`: 改为设置 `PyConfig.home`。

`Py_InitializeFromConfig()` API 应与 `PyConfig` 一起使用。

- 全局配置变量：

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

`Py_InitializeFromConfig()` API 应与 `PyConfig` 一起使用。

Python 3.15 中的待移除功能

- 捆绑的 libmpdecimal 副本。
- `PyImport_ImportModuleNoBlock()`: 改用 `PyImport_ImportModule()`。
- `PyWeakref_GET_OBJECT()`: 改用 `PyWeakref_GetRef()`。
- `PyWeakref_GetObject()`: 改用 `PyWeakref_GetRef()`。
- `Py_UNICODE_WIDE` 类型: 改用 `wchar_t`。
- `Py_UNICODE` 类型: 改用 `wchar_t`。
- Python 初始化函数
 - `PySys_ResetWarnOptions()`: 改为清除 `sys.warnoptions` 和 `warnings.filters`。
 - `Py_GetExecPrefix()`: 改为获取 `sys.exec_prefix`。
 - `Py_GetPath()`: 改为获取 `sys.path`。
 - `Py_GetPrefix()`: 改为获取 `sys.prefix`。
 - `Py_GetProgramFullPath()`: 改为获取 `sys.executable`。
 - `Py_GetProgramName()`: 改为获取 `sys.executable`。
 - `Py_GetPythonHome()`: 改为获取 `PyConfig.home` 或 `PYTHONHOME` 环境变量。

计划在未来版本中移除

以下 API 已被弃用, 将被移除, 但目前尚未确定移除日期。

- `Py_TPFLAGS_HAVE_FINALIZE`: 自 Python 3.8 起不再需要。
- `PyErr_Fetch()`: 改用 `PyErr_GetRaisedException()`。
- `PyErr_NormalizeException()`: 改用 `PyErr_GetRaisedException()`。
- `PyErr_Restore()`: 改用 `PyErr_SetRaisedException()`。
- `PyModule_GetFilename()`: 改用 `PyModule_GetFilenameObject()`。
- `PyOS_AfterFork()`: 改用 `PyOS_AfterFork_Child()`。
- `PySlice_GetIndicesEx()`: 改用 `PySlice_Unpack()` 和 `PySlice_AdjustIndices()`。
- `PyUnicode_AsDecodedObject()`: 改用 `PyCodec_Decode()`。
- `PyUnicode_AsDecodedUnicode()`: 改用 `PyCodec_Decode()`。
- `PyUnicode_AsEncodedObject()`: 改用 `PyCodec_Encode()`。
- `PyUnicode_AsEncodedUnicode()`: 改用 `PyCodec_Encode()`。
- `PyUnicode_READY()`: 自 Python 3.12 起不再需要
- `PyErr_Display()`: 改用 `PyErr_DisplayException()`。
- `_PyErr_ChainExceptions()`: 改用 `_PyErr_ChainExceptions1`。
- `PyBytesObject.ob_shash` 成员: 改为调用 `PyObject_Hash()`。
- `PyDictObject.ma_version_tag` 成员。
- 线程本地存储 (TLS) API:
 - `PyThread_create_key()`: 改用 `PyThread_tss_alloc()`。

- `PyThread_delete_key()`: 改用 `PyThread_tss_free()`。
- `PyThread_set_key_value()`: 改用 `PyThread_tss_set()`。
- `PyThread_get_key_value()`: 改用 `PyThread_tss_get()`。
- `PyThread_delete_key_value()`: 改用 `PyThread_tss_delete()`。
- `PyThread_ReInitTLS()`: 自 **Python 3.7** 起不再需要。

13 回归测试的变化

- 现在使用 `configure --with-pydebug` 编译的 **Python** 将支持 `-X presite=package.module` 命令行选项。如果被使用，它指明一个模块应当在解释器生命周期开始时，即 `site.py` 被执行之前被导入。（由 Łukasz Langa 在 [gh-110769](#) 中贡献。）

索引

非字母

环境变量

PYTHON_BASIC_REPL, 5
PYTHON_COLORS, 4, 6, 13
PYTHON_CPU_COUNT, 16
PYTHON_FROZEN_MODULES, 10
PYTHON_GIL, 7
PYTHON_HISTORY, 10
PYTHON_PERF_JIT_SUPPORT, 10
PYTHONHOME, 39, 41
PYTHONLEGACYWINDOWSFSENCODING, 26

P

Python 增强建议; PEP 11, 8, 35
Python 增强建议; PEP 11#tier-2, 5
Python 增强建议; PEP 11#tier-3, 5
Python 增强建议; PEP 587, 38
Python 增强建议; PEP 590, 39
Python 增强建议; PEP 594, 3, 22
Python 增强建议; PEP 602, 5
Python 增强建议; PEP 626, 28
Python 增强建议; PEP 667, 4, 8, 34, 36, 37, 39
Python 增强建议; PEP 696, 4
Python 增强建议; PEP 699, 40
Python 增强建议; PEP 702, 4, 21
Python 增强建议; PEP 703, 3, 4, 7
Python 增强建议; PEP 705, 4, 20
Python 增强建议; PEP 709, 36
Python 增强建议; PEP 719, 3
Python 增强建议; PEP 730, 5, 8, 9
Python 增强建议; PEP 737, 34
Python 增强建议; PEP 738, 5, 8, 9
Python 增强建议; PEP 742, 5, 20
Python 增强建议; PEP 744, 3, 4, 8
PYTHON_BASIC_REPL, 5
PYTHON_COLORS, 4, 6, 13
PYTHON_CPU_COUNT, 16
PYTHON_FROZEN_MODULES, 10
PYTHON_GIL, 7
PYTHON_HISTORY, 10
PYTHON_PERF_JIT_SUPPORT, 10
PYTHONHOME, 39, 41
PYTHONLEGACYWINDOWSFSENCODING, 26

R

RFC

RFC 5280, 18