
What's New in Python

發行 3.14.0rc3

A. M. Kuchling

10 月 01, 2025

Python Software Foundation
Email: docs@python.org

Contents

1	Summary -- release highlights	4
2	Incompatible changes	4
3	New features	5
3.1	PEP 779: Free-threaded Python is officially supported	5
3.2	PEP 734: Multiple interpreters in the stdlib	5
3.3	PEP 750: Template strings	6
3.4	PEP 768: Safe external debugger interface for CPython	7
3.5	PEP 784: Adding Zstandard to the standard library	8
3.6	Remote attaching to a running Python process with PDB	8
3.7	PEP 758 --Allow except and except* expressions without parentheses	9
3.8	PEP 649 and 749: deferred evaluation of annotations	9
3.9	改善錯誤訊息	10
3.10	PEP 741: Python configuration C API	13
3.11	Asyncio introspection capabilities	13
3.12	A new type of interpreter	15
3.13	Free-threaded mode	15
3.14	Syntax highlighting in PyREPL	16
3.15	Binary releases for the experimental just-in-time compiler	16
3.16	Concurrent safe warnings control	16
3.17	Incremental garbage collection	17
4	Platform support	17
5	Other language changes	17
5.1	PEP 765: Disallow return/break/continue that exit a finally block	18
6	New modules	18
7	Improved modules	19
7.1	argparse	19
7.2	ast	19
7.3	asyncio	19
7.4	calendar	19
7.5	concurrent.futures	19
7.6	configparser	20
7.7	contextvars	20

7.8	ctypes	20
7.9	curses	21
7.10	datetime	21
7.11	decimal	21
7.12	difflib	21
7.13	dis	21
7.14	errno	21
7.15	faulthandler	21
7.16	fnmatch	21
7.17	fractions	21
7.18	functools	22
7.19	getopt	22
7.20	getpass	22
7.21	graphlib	22
7.22	heapq	22
7.23	hmac	22
7.24	http	22
7.25	imaplib	23
7.26	inspect	23
7.27	io	23
7.28	json	23
7.29	linecache	23
7.30	logging.handlers	23
7.31	math	23
7.32	mimetypes	23
7.33	multiprocessing	25
7.34	operator	25
7.35	os	25
7.36	os.path	25
7.37	pathlib	26
7.38	pdb	26
7.39	pickle	26
7.40	platform	26
7.41	pydoc	27
7.42	socket	27
7.43	ssl	27
7.44	struct	27
7.45	symtable	27
7.46	sys	27
7.47	sys.monitoring	28
7.48	sysconfig	28
7.49	tarfile	28
7.50	threading	28
7.51	tkinter	28
7.52	turtle	28
7.53	types	28
7.54	typing	28
7.55	unicodedata	29
7.56	unittest	29
7.57	urllib	29
7.58	uuid	30
7.59	webbrowser	30
7.60	zipfile	30
8	最佳化	30
8.1	asyncio	30
8.2	base64	31
8.3	bdb	31

8.4	difflib	31
8.5	gc	31
8.6	io	31
8.7	pathlib	31
8.8	pdb	31
8.9	uuid	31
8.10	zlib	32
9	已移除	32
9.1	argparse	32
9.2	ast	32
9.3	asyncio	32
9.4	email	34
9.5	importlib.abc	34
9.6	itertools	34
9.7	pathlib	34
9.8	pkgutil	34
9.9	pty	35
9.10	sqlite3	35
9.11	urllib	35
10	已⌘用	35
10.1	New deprecations	35
10.2	Python 3.15 中待移除的項目	37
10.3	Python 3.16 中待移除的項目	38
10.4	Python 3.17 中待移除的項目	39
10.5	Python 3.19 中待移除的項目	40
10.6	未來版本中的待移除項目	40
11	CPython 位元組碼變更	42
11.1	Pseudo-instructions	42
12	C API 變更	43
12.1	C API 中的新功能	43
12.2	Limited C API changes	45
12.3	被移除的 C API	45
12.4	已⌘用的 C API	45
13	建置變更	49
13.1	build-details.json	49
13.2	Discontinuation of PGP signatures	49
14	移植至 Python 3.14	49
14.1	Python API 的變更	49
14.2	C API 中的改動	50
	索引	51

編輯者

Hugo van Kemenade

This article explains the new features in Python 3.14, compared to 3.13.

For full details, see the changelog.

 **也參考**



Prerelease users should be aware that this document is currently in draft form. It will be updated substantially as Python 3.14 moves towards release, so it's worth checking back even after reading earlier versions.

1 Summary -- release highlights

Python 3.14 will be the latest stable release of the Python programming language, with a mix of changes to the language, the implementation and the standard library.

The biggest changes to the implementation include template strings ([PEP 750](#)), deferred evaluation of annotations ([PEP 649](#)), and a new type of interpreter that uses tail calls.

The library changes include the addition of a new `annotationlib` module for introspecting and wrapping annotations ([PEP 749](#)), a new `compression.zstd` module for Zstandard support ([PEP 784](#)), plus syntax highlighting in the REPL, as well as the usual deprecations and removals, and improvements in user-friendliness and correctness.

- *PEP 779: Free-threaded Python is officially supported*
- *PEP 649 and 749: deferred evaluation of annotations*
- *PEP 734: Multiple interpreters in the stdlib*
- *PEP 741: Python configuration C API*
- *PEP 750: Template strings*
- *PEP 758: Allow `except` and `except*` expressions without parentheses*
- *PEP 761: Discontinuation of PGP signatures*
- *PEP 765: Disallow `return/break/continue` that exit a `finally` block*
- *Free-threaded mode improvements*
- *PEP 768: Safe external debugger interface for CPython*
- *PEP 784: Adding Zstandard to the standard library*
- *A new type of interpreter*
- *Syntax highlighting in PyREPL, and color output in `unittest`, `argparse`, `json` and `calendar` CLIs*
- *Binary releases for the experimental just-in-time compiler*

2 Incompatible changes

On platforms other than macOS and Windows, the default start method for `multiprocessing` and `ProcessPoolExecutor` switches from `fork` to `forkserver`.

See [\(1\)](#) and [\(2\)](#) for details.

If you encounter `NameErrors` or pickling errors coming out of `multiprocessing` or `concurrent.futures`, see the `forkserver` restrictions.

The interpreter avoids some reference count modifications internally when it's safe to do so. This can lead to different values returned from `sys.getrefcount()` and `Py_REFCNT()` compared to previous versions of Python. See [below](#) for details.

3 New features

3.1 PEP 779: Free-threaded Python is officially supported

The free-threaded build of Python is now supported and no longer experimental. This is the start of phase II where free-threaded Python is officially supported but still optional.

We are confident that the project is on the right path, and we appreciate the continued dedication from everyone working to make free-threading ready for broader adoption across the Python community.

With these recommendations and the acceptance of this PEP, we as the Python developer community should broadly advertise that free-threading is a supported Python build option now and into the future, and that it will not be removed without a proper deprecation schedule.

Any decision to transition to phase III, with free-threading as the default or sole build of Python is still undecided, and dependent on many factors both within CPython itself and the community. This decision is for the future.

也参考

PEP 779 and its acceptance.

3.2 PEP 734: Multiple interpreters in the stdlib

The CPython runtime supports running multiple copies of Python in the same process simultaneously and has done so for over 20 years. Each of these separate copies is called an “interpreter”. However, the feature had been available only through the C-API.

That limitation is removed in the 3.14 release, with the new `concurrent.interpreters` module.

There are at least two notable reasons why using multiple interpreters is worth considering:

- they support a new (to Python), human-friendly concurrency model
- true multi-core parallelism

For some use cases, concurrency in software enables efficiency and can simplify software, at a high level. At the same time, implementing and maintaining all but the simplest concurrency is often a struggle for the human brain. That especially applies to plain threads (for example, `threading`), where all memory is shared between all threads.

With multiple isolated interpreters, you can take advantage of a class of concurrency models, like CSP or the actor model, that have found success in other programming languages, like Smalltalk, Erlang, Haskell, and Go. Think of multiple interpreters like threads but with opt-in sharing.

Regarding multi-core parallelism: as of the 3.12 release, interpreters are now sufficiently isolated from one another to be used in parallel. (See [PEP 684](#).) This unlocks a variety of CPU-intensive use cases for Python that were limited by the GIL.

Using multiple interpreters is similar in many ways to `multiprocessing`, in that they both provide isolated logical “processes” that can run in parallel, with no sharing by default. However, when using multiple interpreters, an application will use fewer system resources and will operate more efficiently (since it stays within the same process). Think of multiple interpreters as having the isolation of processes with the efficiency of threads.

While the feature has been around for decades, multiple interpreters have not been used widely, due to low awareness and the lack of a stdlib module. Consequently, they currently have several notable limitations, which will improve significantly now that the feature is finally going mainstream.

Current limitations:

- starting each interpreter has not been optimized yet
- each interpreter uses more memory than necessary (we will be working next on extensive internal sharing between interpreters)
- there aren’t many options *yet* for truly sharing objects or other data between interpreters (other than `memoryview`)

- many extension modules on PyPI are not compatible with multiple interpreters yet (stdlib extension modules *are* compatible)
- the approach to writing applications that use multiple isolated interpreters is mostly unfamiliar to Python users, for now

The impact of these limitations will depend on future CPython improvements, how interpreters are used, and what the community solves through PyPI packages. Depending on the use case, the limitations may not have much impact, so try it out!

Furthermore, future CPython releases will reduce or eliminate overhead and provide utilities that are less appropriate on PyPI. In the meantime, most of the limitations can also be addressed through extension modules, meaning PyPI packages can fill any gap for 3.14, and even back to 3.12 where interpreters were finally properly isolated and stopped sharing the GIL. Likewise, we expect to slowly see libraries on PyPI for high-level abstractions on top of interpreters.

Regarding extension modules, work is in progress to update some PyPI projects, as well as tools like Cython, pybind11, nanobind, and PyO3. The steps for isolating an extension module are found at [isolating-extensions-howto](#). Isolating a module has a lot of overlap with what is required to support *free-threading*, so the ongoing work in the community in that area will help accelerate support for multiple interpreters.

Also added in 3.14: `concurrent.futures.InterpreterPoolExecutor`.

👉 也參考

PEP 734.

3.3 PEP 750: Template strings

Template string literals (t-strings) are a generalization of f-strings, using a `t` in place of the `f` prefix. Instead of evaluating to `str`, t-strings evaluate to a new `string.templatelib.Template` type:

```
from string.templatelib import Template

name = "World"
template: Template = t"Hello {name}"
```

The template can then be combined with functions that operate on the template's structure to produce a `str` or a string-like result. For example, sanitizing input:

```
evil = "<script>alert('evil')</script>"
template = t"<p>{evil}</p>"
assert html(template) == "<p>&lt;script&gt;alert('evil')&lt;/script&gt;</p>"
```

As another example, generating HTML attributes from data:

```
attributes = {"src": "shrubbery.jpg", "alt": "looks nice"}
template = t"<img {attributes}>"
assert html(template) == ''
```

Compared to using an f-string, the `html` function has access to template attributes containing the original information: static strings, interpolations, and values from the original scope. Unlike existing templating approaches, t-strings build from the well-known f-string syntax and rules. Template systems thus benefit from Python tooling as they are much closer to the Python language, syntax, scoping, and more.

Writing template handlers is straightforward:

```
from string.templatelib import Template, Interpolation

def lower_upper(template: Template) -> str:
    """Render static parts lowercased and interpolations uppercased."""
    parts: list[str] = []
    for item in template:
```

(繼續下一頁)

(繼續上一頁)

```

    if isinstance(item, Interpolation):
        parts.append(str(item.value).upper())
    else:
        parts.append(item.lower())
    return "".join(parts)

name = "world"
assert lower_upper(t"HELLO {name}") == "hello WORLD"

```

With this in place, developers can write template systems to sanitize SQL, make safe shell operations, improve logging, tackle modern ideas in web development (HTML, CSS, and so on), and implement lightweight, custom business DSLs.

(Contributed by Jim Baker, Guido van Rossum, Paul Everitt, Koudai Aono, Lysandros Nikolaou, Dave Peck, Adam Turner, Jelle Zijlstra, Bénédikt Tran, and Pablo Galindo Salgado in [gh-132661](#).)

也參考

PEP 750.

3.4 PEP 768: Safe external debugger interface for CPython

PEP 768 introduces a zero-overhead debugging interface that allows debuggers and profilers to safely attach to running Python processes. This is a significant enhancement to Python’s debugging capabilities allowing debuggers to forego unsafe alternatives. See [below](#) for how this feature is leveraged to implement the new `pdb` module’s remote attaching capabilities.

The new interface provides safe execution points for attaching debugger code without modifying the interpreter's normal execution path or adding runtime overhead. This enables tools to inspect and interact with Python applications in real-time without stopping or restarting them — a crucial capability for high-availability systems and production environments.

For convenience, CPython implements this interface through the `sys` module with a `sys.remote_exec()` function:

```
sys.remote_exec(pid, script_path)
```

This function allows sending Python code to be executed in a target process at the next safe execution point. However, tool authors can also implement the protocol directly as described in the PEP, which details the underlying mechanisms used to safely attach to running processes.

Here's a simple example that inspects object types in a running Python process:

```
import os
import sys
import tempfile

# Create a temporary script
with tempfile.NamedTemporaryFile(mode='w', suffix='.py', delete=False) as f:
    script_path = f.name
    f.write(f"import my_debugger; my_debugger.connect({os.getpid()})")

try:
    # Execute in process with PID 1234
    print("Behold! An offering:")
    sys.remote_exec(1234, script_path)
finally:
    os.unlink(script_path)
```

The debugging interface has been carefully designed with security in mind and includes several mechanisms to control access:

- A PYTHON DISABLE REMOTE DEBUG environment variable.

- A `-X disable-remote-debug` command-line option.
- A `--without-remote-debug` configure flag to completely disable the feature at build time.

A key implementation detail is that the interface piggybacks on the interpreter's existing evaluation loop and safe points, ensuring zero overhead during normal execution while providing a reliable way for external processes to coordinate debugging operations.

(Contributed by Pablo Galindo Salgado, Matt Wozniski, and Ivona Stojanovic in [gh-131591](#).)

🔗 也参考

PEP 768.

3.5 PEP 784: Adding Zstandard to the standard library

The new compression package contains modules `compression.lzma`, `compression.bz2`, `compression.gzip` and `compression.zlib` which re-export the `lzma`, `bz2`, `gzip` and `zlib` modules respectively. The new import names under `compression` are the canonical names for importing these compression modules going forward. However, the existing modules names have not been deprecated. Any deprecation or removal of the existing compression modules will occur no sooner than five years after the release of 3.14.

The new `compression.zstd` module provides compression and decompression APIs for the Zstandard format via bindings to Meta's [zstd library](#). Zstandard is a widely adopted, highly efficient, and fast compression format. In addition to the APIs introduced in `compression.zstd`, support for reading and writing Zstandard compressed archives has been added to the `tarfile`, `zipfile`, and `shutil` modules.

Here's an example of using the new module to compress some data:

```
from compression import zstd
import math

data = str(math.pi).encode() * 20

compressed = zstd.compress(data)

ratio = len(compressed) / len(data)
print(f"Achieved compression ratio of {ratio}")
```

As can be seen, the API is similar to the APIs of the `lzma` and `bz2` modules.

(Contributed by Emma Harper Smith, Adam Turner, Gregory P. Smith, Tomas Roun, Victor Stinner, and Rogdham in [gh-132983](#).)

🔗 也参考

PEP 784.

3.6 Remote attaching to a running Python process with PDB

The `pdb` module now supports remote attaching to a running Python process using a new `-p PID` command-line option:

```
python -m pdb -p 1234
```

This will connect to the Python process with the given PID and allow you to debug it interactively. Notice that due to how the Python interpreter works attaching to a remote process that is blocked in a system call or waiting for I/O will only work once the next bytecode instruction is executed or when the process receives a signal.

This feature uses [PEP 768](#) and the `sys.remote_exec()` function to attach to the remote process and send the PDB commands to it.

(由 Matt Wozniski and Pablo Galindo 於 [gh-131591](#) 貢獻。)

🔗 也參考

PEP 768.

3.7 PEP 758 –Allow except and except* expressions without parentheses

The `except` and `except*` expressions now allow parentheses to be omitted when there are multiple exception types and the `as` clause is not used. For example the following expressions are now valid:

```
try:
    connect_to_server()
except TimeoutError, ConnectionRefusedError:
    print("Network issue encountered.")

# The same applies to except* (for exception groups):

try:
    connect_to_server()
except* TimeoutError, ConnectionRefusedError:
    print("Network issue encountered.")
```

Check [PEP 758](#) for more details.

(由 Pablo Galindo 和 Brett Cannon 於 [gh-131831](#) 貢獻。)

🔗 也參考

PEP 758.

3.8 PEP 649 and 749: deferred evaluation of annotations

The annotations on functions, classes, and modules are no longer evaluated eagerly. Instead, annotations are stored in special-purpose `__annotations__` functions and evaluated only when necessary (except if `from __future__ import annotations` is used). This is specified in [PEP 649](#) and [PEP 749](#).

This change is designed to make annotations in Python more performant and more usable in most circumstances. The runtime cost for defining annotations is minimized, but it remains possible to introspect annotations at runtime. It is no longer necessary to enclose annotations in strings if they contain forward references.

The new `annotationlib` module provides tools for inspecting deferred annotations. Annotations may be evaluated in the `VALUE` format (which evaluates annotations to runtime values, similar to the behavior in earlier Python versions), the `FORWARDREF` format (which replaces undefined names with special markers), and the `STRING` format (which returns annotations as strings).

This example shows how these formats behave:

```
>>> from annotationlib import get_annotations, Format
>>> def func(arg: Undefined):
...     pass
>>> get_annotations(func, format=Format.VALUE)
Traceback (most recent call last):
...
NameError: name 'Undefined' is not defined
>>> get_annotations(func, format=Format.FORWARDREF)
{'arg': ForwardRef('Undefined', owner=<function func at 0x...>)}
>>> get_annotations(func, format=Format.STRING)
{'arg': 'Undefined'}
```

Implications for annotated code

If you define annotations in your code (for example, for use with a static type checker), then this change probably does not affect you: you can keep writing annotations the same way you did with previous versions of Python.

You will likely be able to remove quoted strings in annotations, which are frequently used for forward references. Similarly, if you use `from __future__ import annotations` to avoid having to write strings in annotations, you may well be able to remove that import once you support only Python 3.14 and newer. However, if you rely on third-party libraries that read annotations, those libraries may need changes to support unquoted annotations before they work as expected.

Implications for readers of `__annotations__`

If your code reads the `__annotations__` attribute on objects, you may want to make changes in order to support code that relies on deferred evaluation of annotations. For example, you may want to use `annotationlib.get_annotations()` with the `FORWARDREF` format, as the `dataclasses` module now does.

The external `typing_extensions` package provides partial backports of some of the functionality of the `annotationlib` module, such as the `Format` enum and the `get_annotations()` function. These can be used to write cross-version code that takes advantage of the new behavior in Python 3.14.

Related changes

The changes in Python 3.14 are designed to rework how `__annotations__` works at runtime while minimizing breakage to code that contains annotations in source code and to code that reads `__annotations__`. However, if you rely on undocumented details of the annotation behavior or on private functions in the standard library, there are many ways in which your code may not work in Python 3.14. To safeguard your code against future changes, use only the documented functionality of the `annotationlib` module.

In particular, do not read annotations directly from the namespace dictionary attribute of type objects. Use `annotationlib.get_annotate_from_class_namespace()` during class construction and `annotationlib.get_annotations()` afterwards.

In previous releases, it was sometimes possible to access class annotations from an instance of an annotated class. This behavior was undocumented and accidental, and will no longer work in Python 3.14.

`from __future__ import annotations`

In Python 3.7, [PEP 563](#) introduced the `from __future__ import annotations` directive, which turns all annotations into strings. This directive is now considered deprecated and it is expected to be removed in a future version of Python. However, this removal will not happen until after Python 3.13, the last version of Python without deferred evaluation of annotations, reaches its end of life in 2029. In Python 3.14, the behavior of code using `from __future__ import annotations` is unchanged.

(Contributed by Jelle Zijlstra in [gh-119180](#); [PEP 649](#) was written by Larry Hastings.)

也參考

[PEP 649](#) and [PEP 749](#).

3.9 改善錯誤訊息

- The interpreter now provides helpful suggestions when it detects typos in Python keywords. When a word that closely resembles a Python keyword is encountered, the interpreter will suggest the correct keyword in the error message. This feature helps programmers quickly identify and fix common typing mistakes. For example:

```
>>> while True:
...     pass
Traceback (most recent call last):
  File "<stdin>", line 1
    while True:
```

(繼續下一頁)

(繼續上一頁)

```
^^^^^^
SyntaxError: invalid syntax. Did you mean 'while'?

>>> async def fetch_data():
...     pass
Traceback (most recent call last):
  File "<stdin>", line 1
    async def fetch_data():
    ^^^^^^
SyntaxError: invalid syntax. Did you mean 'async'?

>>> async def foo():
...     await fetch_data()
Traceback (most recent call last):
  File "<stdin>", line 2
    await fetch_data()
    ^^^^^
SyntaxError: invalid syntax. Did you mean 'await'?

>>> raisee ValueError("Error")
Traceback (most recent call last):
  File "<stdin>", line 1
    raisee ValueError("Error")
    ^^^^^^
SyntaxError: invalid syntax. Did you mean 'raise'?
```

While the feature focuses on the most common cases, some variations of misspellings may still result in regular syntax errors. (Contributed by Pablo Galindo in [gh-132449](#).)

- When an unpacking assignment fails due to an incorrect number of variables, the error message prints the received number of values in more cases than before. (Contributed by Tushar Sadhwani in [gh-122239](#).)

```
>>> x, y, z = 1, 2, 3, 4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    x, y, z = 1, 2, 3, 4
    ^^^^^^^
ValueError: too many values to unpack (expected 3, got 4)
```

- `elif` statements that follow an `else` block now have a specific error message. (Contributed by Steele Farnsworth in [gh-129902](#).)

```
>>> if who == "me":
...     print("It's me!")
... else:
...     print("It's not me!")
... elif who is None:
...     print("Who is it?")
File "<stdin>", line 5
    elif who is None:
    ^^^^
SyntaxError: 'elif' block follows an 'else' block
```

- If a statement (`pass`, `del`, `return`, `yield`, `raise`, `break`, `continue`, `assert`, `import`, `from`) is passed to the `if_expr` after `else`, or one of `pass`, `break`, or `continue` is passed before `if`, then the error message highlights where the expression is required. (Contributed by Sergey Miryanov in [gh-129515](#).)

```
>>> x = 1 if True else pass
Traceback (most recent call last):
  File "<string>", line 1
    x = 1 if True else pass
```

(繼續下一頁)

(繼續上一頁)

```

^^^^
SyntaxError: expected expression after 'else', but statement is given

>>> x = continue if True else break
Traceback (most recent call last):
  File "<string>", line 1
    x = continue if True else break
    ^^^^^^^^^
SyntaxError: expected expression before 'if', but statement is given

```

- When incorrectly closed strings are detected, the error message suggests that the string may be intended to be part of the string. (Contributed by Pablo Galindo in [gh-88535](#).)

```
>>> "The interesting object "The important object" is very important"
Traceback (most recent call last):
SyntaxError: invalid syntax. Is this intended to be part of the string?
```

- When strings have incompatible prefixes, the error now shows which prefixes are incompatible. (Contributed by Nikita Sobolev in [gh-133197](#).)

```
>>> ub'abc'
File "<python-input-0>", line 1
    ub'abc'
    ^^
SyntaxError: 'u' and 'b' prefixes are incompatible
```

- Improved error messages when using `as` with incompatible targets in:

- Imports: `import ... as ...`
- From imports: `from ... import ... as ...`
- Except handlers: `except ... as ...`
- Pattern-match cases: `case ... as ...`

(Contributed by Nikita Sobolev in [gh-123539](#), [gh-123562](#), and [gh-123440](#).)

```
>>> import ast as arr[0]
File "<python-input-1>", line 1
    import ast as arr[0]
                        ^^^^^^
SyntaxError: cannot use subscript as import target
```

- Improved error message when trying to add an instance of an unhashable type to a `dict` or `set`. (Contributed by CF Bolz-Tereick and Victor Stinner in [gh-132828](#).)

```
>>> s = set()
>>> s.add({'pages': 12, 'grade': 'A'})
Traceback (most recent call last):
  File "<python-input-1>", line 1, in <module>
    s.add({'pages': 12, 'grade': 'A'})
    ~~~~~^
TypeError: cannot use 'dict' as a set element (unhashable type: 'dict')

>>> d = {}
>>> l = [1, 2, 3]
>>> d[l] = 12
Traceback (most recent call last):
  File "<python-input-4>", line 1, in <module>
    d[l] = 12
    ~^^^
TypeError: cannot use 'list' as a dict key (unhashable type: 'list')
```

3.10 PEP 741: Python configuration C API

Add a `PyInitConfig` C API to configure the Python initialization without relying on C structures and the ability to make ABI-compatible changes in the future.

Complete the [PEP 587](#) `PyConfig` C API by adding `PyInitConfig_AddModule()` which can be used to add a built-in extension module; a feature previously referred to as the “`inittab`”.

Add `PyConfig_Get()` and `PyConfig_Set()` functions to get and set the current runtime configuration.

PEP 587 “Python Initialization Configuration” unified all the ways to configure the Python initialization. This PEP unifies also the configuration of the Python preinitialization and the Python initialization in a single API. Moreover, this PEP only provides a single choice to embed Python, instead of having two “Python” and “Isolated” choices (PEP 587), to simplify the API further.

The lower level PEP 587 `PyConfig` API remains available for use cases with an intentionally higher level of coupling to CPython implementation details (such as emulating the full functionality of CPython’s CLI, including its configuration mechanisms).

(由 Victor Stinner 於 [gh-107954](#) 貢獻。)

➡ 也參考

[PEP 741](#).

3.11 Asyncio introspection capabilities

Added a new command-line interface to inspect running Python processes using asynchronous tasks, available via:

```
python -m asyncio ps PID
```

This tool inspects the given process ID (PID) and displays information about currently running asyncio tasks. It outputs a task table: a flat listing of all tasks, their names, their coroutine stacks, and which tasks are awaiting them.

```
python -m asyncio pstree PID
```

This tool fetches the same information, but renders a visual async call tree, showing coroutine relationships in a hierarchical format. This command is particularly useful for debugging long-running or stuck asynchronous programs. It can help developers quickly identify where a program is blocked, what tasks are pending, and how coroutines are chained together.

For example given this code:

```
import asyncio

async def play(track):
    await asyncio.sleep(5)
    print(f"🎵 Finished: {track}")

async def album(name, tracks):
    async with asyncio.TaskGroup() as tg:
        for track in tracks:
            tg.create_task(play(track), name=track)

async def main():
    async with asyncio.TaskGroup() as tg:
        tg.create_task(
            album("Sundowning", ["TNDNBTG", "Levitate"]), name="Sundowning")
        tg.create_task(
            album("TMBTE", ["DYWTYLM", "Aqua Regia"]), name="TMBTE")

if __name__ == "__main__":
    asyncio.run(main())
```

Executing the new tool on the running process will yield a table like this:

```
python -m asyncio ps 12345
```

tid	task id	task name	coroutine stack	
	awaiter chain		awaiter name	awaiter id

1935500	0x7fc930c18050	Task-1	TaskGroup._aexit -> TaskGroup.__aexit__	
↪-> main				0x0
1935500	0x7fc930c18230	Sundowning	TaskGroup._aexit -> TaskGroup.__aexit__	
↪-> album	TaskGroup._aexit -> TaskGroup.__aexit__	-> main	Task-1	
↪0x7fc930c18050				
1935500	0x7fc93173fa50	TMBTE	TaskGroup._aexit -> TaskGroup.__aexit__	
↪-> album	TaskGroup._aexit -> TaskGroup.__aexit__	-> main	Task-1	
↪0x7fc930c18050				
1935500	0x7fc93173fdf0	TNDNB TG	sleep -> play	
↪	TaskGroup._aexit -> TaskGroup.__aexit__	-> album	Sundowning	
↪0x7fc930c18230				
1935500	0x7fc930d32510	Levitate	sleep -> play	
↪	TaskGroup._aexit -> TaskGroup.__aexit__	-> album	Sundowning	
↪0x7fc930c18230				
1935500	0x7fc930d32890	DYWTYLM	sleep -> play	
↪	TaskGroup._aexit -> TaskGroup.__aexit__	-> album	TMBTE	
↪0x7fc93173fa50				
1935500	0x7fc93161ec30	Aqua Regia	sleep -> play	
↪	TaskGroup._aexit -> TaskGroup.__aexit__	-> album	TMBTE	
↪0x7fc93173fa50				

or a tree like this:

```
python -m asyncio pstree 12345
```

```

├─ (T) Task-1
│   └─ main example.py:13
│       └─ TaskGroup.__aexit__ Lib/asyncio/taskgroups.py:72
│           └─ TaskGroup._aexit Lib/asyncio/taskgroups.py:121
│               └─ (T) Sundowning
│                   └─ album example.py:8
│                       └─ TaskGroup.__aexit__ Lib/asyncio/taskgroups.py:72
│                           └─ TaskGroup._aexit Lib/asyncio/taskgroups.py:121
│                               └─ (T) TNDNB TG
│                                   └─ play example.py:4
│                                       └─ sleep Lib/asyncio/tasks.py:702
│                                           └─ (T) Levitate
│                                               └─ play example.py:4
│                                                   └─ sleep Lib/asyncio/tasks.py:702
│                                                       └─ (T) TMBTE
│                                                           └─ album example.py:8
│                                                               └─ TaskGroup.__aexit__ Lib/asyncio/taskgroups.py:72
│                                                                   └─ TaskGroup._aexit Lib/asyncio/taskgroups.py:121
│                                                                       └─ (T) DYWTYLM
│                                                                           └─ play example.py:4
│                                                                               └─ sleep Lib/asyncio/tasks.py:702
│                                                                                   └─ (T) Aqua Regia
│                                                                                       └─ play example.py:4
│                                                                                           └─ sleep Lib/asyncio/tasks.py:702

```

If a cycle is detected in the async await graph (which could indicate a programming issue), the tool raises an error and lists the cycle paths that prevent tree construction:

```
python -m asyncio pstree 12345
```

(繼續下一頁)

```
ERROR: await-graph contains cycles - cannot print a tree!

cycle: Task-2 → Task-3 → Task-2
```

(Contributed by Pablo Galindo, Łukasz Langa, Yury Selivanov, and Marta Gomez Macias in [gh-91048](#).)

3.12 A new type of interpreter

A new type of interpreter has been added to CPython. It uses tail calls between small C functions that implement individual Python opcodes, rather than one large C case statement. For certain newer compilers, this interpreter provides significantly better performance. Preliminary numbers on our machines suggest anywhere up to 30% faster Python code, and a geometric mean of 3-5% faster on `pyperformance` depending on platform and architecture. The baseline is Python 3.14 built with Clang 19 without this new interpreter.

This interpreter currently only works with Clang 19 and newer on x86-64 and AArch64 architectures. However, we expect that a future release of GCC will support this as well.

This feature is opt-in for now. We highly recommend enabling profile-guided optimization with the new interpreter as it is the only configuration we have tested and can validate its improved performance. For further information on how to build Python, see `--with-tail-call-interp`.

備註

This is not to be confused with [tail call optimization](#) of Python functions, which is currently not implemented in CPython.

This new interpreter type is an internal implementation detail of the CPython interpreter. It doesn't change the visible behavior of Python programs at all. It can improve their performance, but doesn't change anything else.

注意

This section previously reported a 9-15% geometric mean speedup. This number has since been cautiously revised down to 3-5%. While we expect performance results to be better than what we report, our estimates are more conservative due to a [compiler bug](#) found in Clang/LLVM 19, which causes the normal interpreter to be slower. We were unaware of this bug, resulting in inaccurate results. We sincerely apologize for communicating results that were only accurate for LLVM v19.1.x and v20.1.0. In the meantime, the bug has been fixed in LLVM v20.1.1 and for the upcoming v21.1, but it will remain unfixed for LLVM v19.1.x and v20.1.0. Thus any benchmarks with those versions of LLVM may produce inaccurate numbers. (Thanks to Nelson Elhage for bringing this to light.)

(Contributed by Ken Jin in [gh-128563](#), with ideas on how to implement this in CPython by Mark Shannon, Garrett Gu, Haoran Xu, and Josh Haberman.)

3.13 Free-threaded mode

Free-threaded mode ([PEP 703](#)), initially added in 3.13, has been significantly improved. The implementation described in PEP 703 was finished, including C API changes, and temporary workarounds in the interpreter were replaced with more permanent solutions. The specializing adaptive interpreter ([PEP 659](#)) is now enabled in free-threaded mode, which along with many other optimizations greatly improves its performance. The performance penalty on single-threaded code in free-threaded mode is now roughly 5-10%, depending on platform and C compiler used.

This work was done by many contributors: Sam Gross, Matt Page, Neil Schemenauer, Thomas Wouters, Donghee Na, Kirill Podoprigora, Ken Jin, Itamar Oren, Brett Simmers, Dino Viehland, Nathan Goldbaum, Ralf Gommers, Lysandros Nikolaou, Kumar Aditya, Edgar Margffoy, and many others.

Some of these contributors are employed by Meta, which has continued to provide significant engineering resources to support this project.

From 3.14, when compiling extension modules for the free-threaded build of CPython on Windows, the preprocessor variable `PY_GIL_DISABLED` now needs to be specified by the build backend, as it will no longer be determined automatically by the C compiler. For a running interpreter, the setting that was used at compile time can be found using `sysconfig.get_config_var()`.

A new flag has been added, `context_aware_warnings`. This flag defaults to true for the free-threaded build and false for the GIL-enabled build. If the flag is true then the `warnings.catch_warnings` context manager uses a context variable for warning filters. This makes the context manager behave predictably when used with multiple threads or asynchronous tasks.

A new flag has been added, `thread_inherit_context`. This flag defaults to true for the free-threaded build and false for the GIL-enabled build. If the flag is true then threads created with `threading.Thread` start with a copy of the `Context()` of the caller of `start()`. Most significantly, this makes the warning filtering context established by `catch_warnings` be “inherited” by threads (or asyncio tasks) started within that context. It also affects other modules that use context variables, such as the `decimal` context manager.

3.14 Syntax highlighting in PyREPL

The default interactive shell now highlights Python syntax as you type. The feature is enabled by default unless the `PYTHON_BASIC_REPL` environment is set or any color-disabling environment variables are used. See [using-on-controlling-color](#) for details.

The default color theme for syntax highlighting strives for good contrast and uses exclusively the 4-bit VGA standard ANSI color codes for maximum compatibility. The theme can be customized using an experimental API `_colorize.set_theme()`. This can be called interactively, as well as in the `PYTHONSTARTUP` script.

(由 Łukasz Langa 於 [gh-131507](#) 貢獻。)

3.15 Binary releases for the experimental just-in-time compiler

The official macOS and Windows release binaries now include an *experimental* just-in-time (JIT) compiler. Although it is **not** recommended for production use, it can be tested by setting `PYTHON_JIT=1` as an environment variable. Downstream source builds and redistributors can use the `--enable-experimental-jit=yes-off` configuration option for similar behavior.

The JIT is at an early stage and still in active development. As such, the typical performance impact of enabling it can range from 10% slower to 20% faster, depending on workload. To aid in testing and evaluation, a set of introspection functions has been provided in the `sys._jit` namespace. `sys._jit.is_available()` can be used to determine if the current executable supports JIT compilation, while `sys._jit.is_enabled()` can be used to tell if JIT compilation has been enabled for the current process.

Currently, the most significant missing functionality is that native debuggers and profilers like `gdb` and `perf` are unable to unwind through JIT frames (Python debuggers and profilers, like `pdb` or `profile`, continue to work without modification). Free-threaded builds do not support JIT compilation.

Please report any bugs or major performance regressions that you encounter!

🔗 也参考

PEP 744

3.16 Concurrent safe warnings control

The `warnings.catch_warnings` context manager will now optionally use a context variable for warning filters. This is enabled by setting the `context_aware_warnings` flag, either with the `-X` command-line option or an environment variable. This gives predictable warnings control when using `catch_warnings` combined with multiple threads or asynchronous tasks. The flag defaults to true for the free-threaded build and false for the GIL-enabled build.

(由 Neil Schemenauer 和 Kumar Aditya 於 [gh-130010](#) 貢獻。)

3.17 Incremental garbage collection

The cycle garbage collector is now incremental. This means that maximum pause times are reduced by an order of magnitude or more for larger heaps.

There are now only two generations: young and old. When `gc.collect()` is not called directly, the GC is invoked a little less frequently. When invoked, it collects the young generation and an increment of the old generation, instead of collecting one or more generations.

The behavior of `gc.collect()` changes slightly:

- `gc.collect(1)`: Performs an increment of garbage collection, rather than collecting generation 1.
- Other calls to `gc.collect()` are unchanged.

(由 Mark Shannon 於 [gh-108362](#) 貢獻。)

4 Platform support

- **PEP 776**: Emscripten is now an officially supported platform at **tier 3**. As a part of this effort, more than 25 bugs in [Emscripten libc](#) were fixed. Emscripten now includes support for `ctypes`, `termios`, and `fcntl`, as well as experimental support for PyREPL.

(Contributed by R. Hood Chatham in [gh-127146](#), [gh-127683](#), and [gh-136931](#).)

5 Other language changes

- The default interactive shell now supports import autocompletion. This means that typing `import foo` and pressing `<tab>` will suggest modules starting with `foo`. Similarly, typing `from foo import b` will suggest submodules of `foo` starting with `b`. Note that autocompletion of module attributes is not currently supported. (Contributed by Tomas Roun in [gh-69605](#).)
- The `map()` built-in now has an optional keyword-only `strict` flag like `zip()` to check that all the iterables are of equal length. (Contributed by Wannes Boeykens in [gh-119793](#).)
- Incorrect usage of `await` and asynchronous comprehensions is now detected even if the code is optimized away by the `-O` command-line option. For example, `python -O -c 'assert await 1'` now produces a `SyntaxError`. (Contributed by Jelle Zijlstra in [gh-121637](#).)
- Writes to `__debug__` are now detected even if the code is optimized away by the `-O` command-line option. For example, `python -O -c 'assert (__debug__ := 1)'` now produces a `SyntaxError`. (Contributed by Irit Katriel in [gh-122245](#).)
- Add class methods `float.from_number()` and `complex.from_number()` to convert a number to `float` or `complex` type correspondingly. They raise an error if the argument is a string. (Contributed by Serhiy Storchaka in [gh-84978](#).)
- Implement mixed-mode arithmetic rules combining real and complex numbers as specified by C standards since C99. (Contributed by Sergey B Kirpichev in [gh-69639](#).)
- All Windows code pages are now supported as "cpXXX" codecs on Windows. (Contributed by Serhiy Storchaka in [gh-123803](#).)
- `super` objects are now `pickleable` and `copyable`. (Contributed by Serhiy Storchaka in [gh-125767](#).)
- The `memoryview` type now supports subscription, making it a generic type. (Contributed by Brian Schubert in [gh-126012](#).)
- Support underscore and comma as thousands separators in the fractional part for floating-point presentation types of the new-style string formatting (with `format()` or f-strings). (Contributed by Sergey B Kirpichev in [gh-87790](#).)

- ## 5.1 PEP 765: Disallow `return/break/continue` that exit a `finally` block

6 New modules

- 18

7 Improved modules

7.1 argparse

- The default value of the program name for `argparse.ArgumentParser` now reflects the way the Python interpreter was instructed to find the `__main__` module code. (Contributed by Serhiy Storchaka and Alyssa Coghlan in [gh-66436](#).)
- Introduced the optional `suggest_on_error` parameter to `argparse.ArgumentParser`, enabling suggestions for argument choices and subparser names if mistyped by the user. (Contributed by Savannah Ostrowski in [gh-124456](#).)
- Enable color for help text, which can be disabled with the optional `color` parameter to `argparse.ArgumentParser`. This can also be controlled by environment variables. (Contributed by Hugo van Kemenade in [gh-130645](#).)

7.2 ast

- Add `ast.compare()` for comparing two ASTs. (Contributed by Batuhan Taskaya and Jeremy Hylton in [gh-60191](#).)
- Add support for `copy.replace()` for AST nodes. (Contributed by Bénédict Tran in [gh-121141](#).)
- Docstrings are now removed from an optimized AST in optimization level 2. (Contributed by Irit Katriel in [gh-123958](#).)
- The `repr()` output for AST nodes now includes more information. (Contributed by Tomas Roun in [gh-116022](#).)
- `ast.parse()`, when called with an AST as input, now always verifies that the root node type is appropriate. (Contributed by Irit Katriel in [gh-130139](#).)
- Add new `--feature-version`, `--optimize`, `--show-empty` options to the command-line interface. (Contributed by Semyon Moroz in [gh-133367](#).)

7.3 asincio

- The function and methods named `create_task()` now take an arbitrary list of keyword arguments. All keyword arguments are passed to the `Task` constructor or the custom task factory. (See `set_task_factory()` for details.) The `name` and `context` keyword arguments are no longer special; the name should now be set using the `name` keyword argument of the factory, and `context` may be `None`.

This affects the following function and methods: `asyncio.create_task()`, `asyncio.loop.create_task()`, `asyncio.TaskGroup.create_task()`. (Contributed by Thomas Grainger in [gh-128307](#).)

- There are two new utility functions for introspecting and printing a program's call graph: `capture_call_graph()` and `print_call_graph()`. (Contributed by Yury Selivanov, Pablo Galindo Salgado, and Łukasz Langa in [gh-91048](#).)

7.4 calendar

- By default, today's date is highlighted in color in `calendar`'s command-line text output. This can be controlled by environment variables. (Contributed by Hugo van Kemenade in [gh-128317](#).)

7.5 concurrent.futures

- Add `InterpreterPoolExecutor`, which exposes “subinterpreters” (multiple Python interpreters in the same process) to Python code. This is separate from the proposed API in [PEP 734](#). (Contributed by Eric Snow in [gh-124548](#).)
- The default `ProcessPoolExecutor` start method changed from `fork` to `forkserver` on platforms other than macOS and Windows where it was already `spawn`.

If the threading incompatible *fork* method is required, you must explicitly request it by supplying a multiprocessing context *mp_context* to `ProcessPoolExecutor`.

See `forkserver` restrictions for information and differences with the *fork* method and how this change may affect existing code with mutable global shared variables and/or shared objects that can not be automatically pickled.

(由 Gregory P. Smith 於 [gh-84559](#) 貢獻。)

- Add `concurrent.futures.ProcessPoolExecutor.terminate_workers()` and `concurrent.futures.ProcessPoolExecutor.kill_workers()` as ways to terminate or kill all living worker processes in the given pool. (Contributed by Charles Machalow in [gh-130849](#).)
- Add the optional `buffer_size` parameter to `concurrent.futures.Executor.map()` to limit the number of submitted tasks whose results have not yet been yielded. If the buffer is full, iteration over the *iterables* pauses until a result is yielded from the buffer. (Contributed by Enzo Bonnal and Josh Rosenberg in [gh-74028](#).)

7.6 configparser

- Security fix: will no longer write config files it cannot read. Attempting to `configparser.ConfigParser.write()` keys containing delimiters or beginning with the section header pattern will raise a `configparser.InvalidWriteError`. (Contributed by Jacob Lincoln in [gh-129270](#).)

7.7 contextvars

- Support context manager protocol by `contextvars.Token`. (Contributed by Andrew Svetlov in [gh-129889](#).)

7.8 ctypes

- The layout of bit fields in `Structure` and `Union` now matches platform defaults (GCC/Clang or MSVC) more closely. In particular, fields no longer overlap. (Contributed by Matthias Görgens in [gh-97702](#).)
- The `Structure._layout_class` attribute can now be set to help match a non-default ABI. (Contributed by Petr Viktorin in [gh-97702](#).)
- The class of `Structure/Union` field descriptors is now available as `CField`, and has new attributes to aid debugging and introspection. (Contributed by Petr Viktorin in [gh-128715](#).)
- On Windows, the `COMError` exception is now public. (Contributed by Jun Komoda in [gh-126686](#).)
- On Windows, the `CopyComPointer()` function is now public. (Contributed by Jun Komoda in [gh-127275](#).)
- `ctypes.memoryview_at()` now exists to create a `memoryview` object that refers to the supplied pointer and length. This works like `ctypes.string_at()` except it avoids a buffer copy, and is typically useful when implementing pure Python callback functions that are passed dynamically-sized buffers. (Contributed by Rian Hunter in [gh-112018](#).)
- Complex types, `c_float_complex`, `c_double_complex` and `c_longdouble_complex`, are now available if both the compiler and the `libffi` library support complex C types. (Contributed by Sergey B Kirpichev in [gh-61103](#).)
- Add `ctypes.util.dllopen()` for listing the shared libraries loaded by the current process. (Contributed by Brian Ward in [gh-119349](#).)
- Move `ctypes.POINTER()` types cache from a global internal cache (`_pointer_type_cache`) to the `ctypes._CData.__pointer_type__` attribute of the corresponding `ctypes` types. This will stop the cache from growing without limits in some situations. (Contributed by Sergey Miryanov in [gh-100926](#).)
- The `ctypes.py_object` type now supports subscription, making it a generic type. (Contributed by Brian Schubert in [gh-132168](#).)
- `ctypes` now supports free-threading builds. (Contributed by Kumar Aditya and Peter Bierma in [gh-127945](#).)

7.9 curses

- Add the `assume_default_colors()` function, a refinement of the `use_default_colors()` function which allows to change the color pair 0. (Contributed by Serhiy Storchaka in [gh-133139](#).)

7.10 datetime

- Add `datetime.time.strptime()` and `datetime.date.strptime()`. (Contributed by Wannes Boeykens in [gh-41431](#).)

7.11 decimal

- Add alternative `Decimal` constructor `Decimal.from_number()`. (Contributed by Serhiy Storchaka in [gh-121798](#).)
- Expose `decimal.IEEEContext()` to support creation of contexts corresponding to the IEEE 754 (2008) decimal interchange formats. (Contributed by Sergey B Kirpichev in [gh-53032](#).)

7.12 difflib

- Comparison pages with highlighted changes generated by the `difflib.HtmlDiff` class now support dark mode. (Contributed by Jiahao Li in [gh-129939](#).)

7.13 dis

- Add support for rendering full source location information of `instructions`, rather than only the line number. This feature is added to the following interfaces via the `show_positions` keyword argument:

```
- dis.Bytecode
- dis.dis()
- dis.distb()
- dis.disassemble()
```

This feature is also exposed via `dis --show-positions`. (Contributed by Bénédict Tran in [gh-123165](#).)

- Add the `dis --specialized` command-line option to show specialized bytecode. (Contributed by Bénédict Tran in [gh-127413](#).)

7.14 errno

- Add `errno.EHWPOISON` error code. (Contributed by James Roy in [gh-126585](#).)

7.15 faulthandler

- Add support for printing the C stack trace on systems that support it via `faulthandler.dump_c_stack()` or via the `c_stack` argument in `faulthandler.enable()`. (Contributed by Peter Bierma in [gh-127604](#).)

7.16 fnmatch

- Added `fnmatch.filterfalse()` for excluding names matching a pattern. (Contributed by Bénédict Tran in [gh-74598](#).)

7.17 fractions

- Add support for converting any objects that have the `as_integer_ratio()` method to a `Fraction`. (Contributed by Serhiy Storchaka in [gh-82017](#).)
- Add alternative `Fraction` constructor `Fraction.from_number()`. (Contributed by Serhiy Storchaka in [gh-121797](#).)

7.18 functools

- Add support to `functools.partial()` and `functools.partialmethod()` for `functools.Placeholder` sentinels to reserve a place for positional arguments. (Contributed by Dominykas Grigonis in [gh-119127](#).)
- Allow the *initial* parameter of `functools.reduce()` to be passed as a keyword argument. (Contributed by Sayandip Dutta in [gh-125916](#).)

7.19 getopt

- Add support for options with optional arguments. (Contributed by Serhiy Storchaka in [gh-126374](#).)
- Add support for returning intermixed options and non-option arguments in order. (Contributed by Serhiy Storchaka in [gh-126390](#).)

7.20 getpass

- Support keyboard feedback by `getpass.getpass()` via the keyword-only optional argument `echo_char`. Placeholder characters are rendered whenever a character is entered, and removed when a character is deleted. (Contributed by Semyon Moroz in [gh-77065](#).)

7.21 graphlib

- Allow `graphlib.TopologicalSorter.prepare()` to be called more than once as long as sorting has not started. (Contributed by Daniel Pope in [gh-130914](#).)

7.22 heapq

- Add functions for working with max-heaps:
 - `heapq.heapify_max()`,
 - `heapq.heappush_max()`,
 - `heapq.heappop_max()`,
 - `heapq.heapreplace_max()`
 - `heapq.heappushpop_max()`

7.23 hmac

- Add a built-in implementation for HMAC (**RFC 2104**) using formally verified code from the **HACL*** project. (Contributed by Bénédict Tran in [gh-99108](#).)

7.24 http

- Directory lists and error pages generated by the `http.server` module allow the browser to apply its default dark mode. (Contributed by Yorik Hansen in [gh-123430](#).)
- The `http.server` module now supports serving over HTTPS using the `http.server.HTTPSServer` class. This functionality is exposed by the command-line interface (`python -m http.server`) through the following options:
 - `--tls-cert <path>`: Path to the TLS certificate file.
 - `--tls-key <path>`: Optional path to the private key file.
 - `--tls-password-file <path>`: Optional path to the password file for the private key.(由 Semyon Moroz 於 [gh-85162](#) 貢獻。)

7.25 imaplib

- Add `IMAP4.idle()`, implementing the IMAP4 IDLE command as defined in [RFC 2177](#). (Contributed by Forest in [gh-55454](#).)

7.26 inspect

- `inspect.signature()` takes a new argument *annotation_format* to control the `annotationlib.Format` used for representing annotations. (Contributed by Jelle Zijlstra in [gh-101552](#).)
- `inspect.Signature.format()` takes a new argument *unquote_annotations*. If true, string annotations are displayed without surrounding quotes. (Contributed by Jelle Zijlstra in [gh-101552](#).)
- Add function `inspect.ispackage()` to determine whether an object is a package or not. (Contributed by Zhikang Yan in [gh-125634](#).)

7.27 io

- Reading text from a non-blocking stream with `read` may now raise a `BlockingIOError` if the operation cannot immediately return bytes. (Contributed by Giovanni Siragusa in [gh-109523](#).)
- Add protocols `io.Reader` and `io.Writer` as simpler alternatives to the pseudo-protocols `typing.IO`, `typing.TextIO`, and `typing.BinaryIO`. (Contributed by Sebastian Rittau in [gh-127648](#).)

7.28 json

- Add notes for JSON serialization errors that allow to identify the source of the error. (Contributed by Serhiy Storchaka in [gh-122163](#).)
- Enable the `json` module to work as a script using the `-m` switch: `python -m json`. See the JSON command-line interface documentation. (Contributed by Trey Hunner in [gh-122873](#).)
- By default, the output of the JSON command-line interface is highlighted in color. This can be controlled by environment variables. (Contributed by Tomas Roun in [gh-131952](#).)

7.29 linecache

- `linecache.getline()` can retrieve source code for frozen modules. (Contributed by Tian Gao in [gh-131638](#).)

7.30 logging.handlers

- `logging.handlers.QueueListener` now implements the context manager protocol, allowing it to be used in a `with` statement. (Contributed by Charles Machalow in [gh-132106](#).)
- `QueueListener.start` now raises a `RuntimeError` if the listener is already started. (Contributed by Charles Machalow in [gh-132106](#).)

7.31 math

- Added more detailed error messages for domain errors in the module. (Contributed by Charlie Zhao and Sergey B Kirpichev in [gh-101410](#).)

7.32 mimetypes

- Document the command-line for `mimetypes`. It now exits with 1 on failure instead of 0 and 2 on incorrect command-line parameters instead of 1. Also, errors are printed to `stderr` instead of `stdout` and their text is made tighter. (Contributed by Oleg Iarygin and Hugo van Kemenade in [gh-93096](#).)
- Add MS and [RFC 8081](#) MIME types for fonts:
 - Embedded OpenType: `application/vnd.ms-fontobject`
 - OpenType Layout (OTF) `font/otf`

- TrueType: `font/ttf`
- WOFF 1.0 `font/woff`
- WOFF 2.0 `font/woff2`

(由 Sahil Prajapati and Hugo van Kemenade 於 [gh-84852](#) 貢獻。)

- Add **RFC 9559** MIME types for Matroska audiovisual data container structures, containing:

- audio with no video: `audio/matroska (.mka)`
- video: `video/matroska (.mkv)`
- stereoscopic video: `video/matroska-3d (.mk3d)`

(由 Hugo van Kemenade 於 [gh-89416](#) 貢獻。)

- Add MIME types for images with RFCs:

- **RFC 1494**: CCITT Group 3 (`.g3`)
- **RFC 3362**: Real-time Facsimile, T.38 (`.t38`)
- **RFC 3745**: JPEG 2000 (`.jp2`), extension (`.jpx`) and compound (`.jpm`)
- **RFC 3950**: Tag Image File Format Fax eXtended, TIFF-FX (`.tfx`)
- **RFC 4047**: Flexible Image Transport System (`.fits`)
- **RFC 7903**: Enhanced Metafile (`.emf`) and Windows Metafile (`.wmf`)

(由 Hugo van Kemenade 於 [gh-85957](#) 貢獻。)

- More MIME type changes:

- **RFC 2361**: Change type for `.avi` to `video/vnd.avi` and for `.wav` to `audio/vnd.wave`
- **RFC 4337**: Add MPEG-4 `audio/mp4 (.m4a)`
- **RFC 5334**: Add Ogg media (`.oga`, `.ogg` and `.ogx`)
- **RFC 6713**: Add `gzip application/gzip (.gz)`
- **RFC 9639**: Add `FLAC audio/flac (.flac)`
- Add `7z application/x-7z-compressed (.7z)`
- Add `Android Package application/vnd.android.package-archive (.apk)` when not strict
- Add `deb application/x-debian-package (.deb)`
- Add `glTF binary model/gltf-binary (.glb)`
- Add `glTF JSON/ASCII model/gltf+json (.gltf)`
- Add `M4V video/x-m4v (.m4v)`
- Add `PHP application/x-httpd-php (.php)`
- Add `RAR application/vnd.rar (.rar)`
- Add `RPM application/x-rpm (.rpm)`
- Add `STL model/stl (.stl)`
- Add `Windows Media Video video/x-ms-wmv (.wmv)`
- De facto: Add `WebM audio/webm (.weba)`
- **ECMA-376**: Add `.docx`, `.pptx` and `.xlsx` types
- **OASIS**: Add `OpenDocument .odg`, `.odp`, `.ods` and `.odt` types
- **W3C**: Add `EPUB application/epub+zip (.epub)`

(由 Hugo van Kemenade 於 [gh-129965](#) 貢獻。)

- Add [RFC 9512](#) `application/yaml` MIME type for YAML files (`.yaml` and `.yml`). (Contributed by Sasha "Nelie" Chernykh and Hugo van Kemenade in [gh-132056](#).)

7.33 multiprocessing

- The default start method changed from `fork` to `forkserver` on platforms other than macOS and Windows where it was already spawn.

If the threading incompatible `fork` method is required, you must explicitly request it via a context from `multiprocessing.get_context()` (preferred) or change the default via `multiprocessing.set_start_method()`.

See `forkserver` restrictions for information and differences with the `fork` method and how this change may affect existing code with mutable global shared variables and/or shared objects that can not be automatically pickled.

(由 Gregory P. Smith 於 [gh-84559](#) 貢獻。)

- `multiprocessing`'s `"forkserver"` start method now authenticates its control socket to avoid solely relying on filesystem permissions to restrict what other processes could cause the `forkserver` to spawn workers and run code. (Contributed by Gregory P. Smith for [gh-97514](#).)
- The multiprocessing proxy objects for `list` and `dict` types gain previously overlooked missing methods:
 - `clear()` and `copy()` for proxies of `list`
 - `fromkeys()`, `reversed(d)`, `d | {}`, `{}` | `d`, `d |= {'b': 2}` for proxies of `dict`

(由 Roy Hyunjin Han 於 [gh-103134](#) 貢獻。)

- Add support for shared `set` objects via `SyncManager.set()`. The `set()` in `multiprocessing.Manager()` method is now available. (Contributed by Mingyu Park in [gh-129949](#).)
- Add `multiprocessing.Process.interrupt()` which terminates the child process by sending `SIGINT`. This enables `finally` clauses to print a stack trace for the terminated process. (Contributed by Artem Pulkin in [gh-131913](#).)

7.34 operator

- Two new functions `operator.is_none()` and `operator.is_not_none()` have been added, such that `operator.is_none(obj)` is equivalent to `obj is None` and `operator.is_not_none(obj)` is equivalent to `obj is not None`. (Contributed by Raymond Hettinger and Nico Mexis in [gh-115808](#).)

7.35 os

- Add the `os.reload_environ()` function to update `os.environ` and `os.environb` with changes to the environment made by `os.putenv()`, by `os.unsetenv()`, or made outside Python in the same process. (Contributed by Victor Stinner in [gh-120057](#).)
- Add the `SCHED_DEADLINE` and `SCHED_NORMAL` constants to the `os` module. (Contributed by James Roy in [gh-127688](#).)
- Add the `os.readinto()` function to read into a buffer object from a file descriptor. (Contributed by Cody Maloney in [gh-129205](#).)

7.36 os.path

- The `strict` parameter to `os.path.realpath()` accepts a new value, `os.path.ALLOW_MISSING`. If used, errors other than `FileNotFoundError` will be re-raised; the resulting path can be missing but it will be free of symlinks. (Contributed by Petr Viktorin for [CVE 2025-4517](#).)

7.37 pathlib

- Add methods to `pathlib.Path` to recursively copy or move files and directories:
 - `copy()` copies a file or directory tree to a destination.
 - `copy_into()` copies *into* a destination directory.
 - `move()` moves a file or directory tree to a destination.
 - `move_into()` moves *into* a destination directory.(由 Barney Gale 於 [gh-73991](#) 貢獻。)
- Add `pathlib.Path.info` attribute, which stores an object implementing the `pathlib.types.PathInfo` protocol (also new). The object supports querying the file type and internally caching `stat()` results. Path objects generated by `iterdir()` are initialized with file type information gleaned from scanning the parent directory. (Contributed by Barney Gale in [gh-125413](#).)

7.38 pdb

- Hardcoded breakpoints (`breakpoint()` and `pdb.set_trace()`) now reuse the most recent `Pdb` instance that calls `set_trace()`, instead of creating a new one each time. As a result, all the instance specific data like `display` and `commands` are preserved across hardcoded breakpoints. (Contributed by Tian Gao in [gh-121450](#).)
- Add a new argument `mode` to `pdb.Pdb`. Disable the `restart` command when `pdb` is in `inline` mode. (Contributed by Tian Gao in [gh-123757](#).)
- A confirmation prompt will be shown when the user tries to quit `pdb` in `inline` mode. `y`, `Y`, `<Enter>` or `EOF` will confirm the quit and call `sys.exit()`, instead of raising `bdb.BdbQuit`. (Contributed by Tian Gao in [gh-124704](#).)
- Inline breakpoints like `breakpoint()` or `pdb.set_trace()` will always stop the program at calling frame, ignoring the `skip` pattern (if any). (Contributed by Tian Gao in [gh-130493](#).)
- `<tab>` at the beginning of the line in `pdb` multi-line input will fill in a 4-space indentation now, instead of inserting a `\t` character. (Contributed by Tian Gao in [gh-130471](#).)
- Auto-indent is introduced in `pdb` multi-line input. It will either keep the indentation of the last line or insert a 4-space indentation when it detects a new code block. (Contributed by Tian Gao in [gh-133350](#).)
- `$_asynctask` is added to access the current `asyncio` task if applicable. (Contributed by Tian Gao in [gh-124367](#).)
- `pdb.set_trace_async()` is added to support debugging `asyncio` coroutines. `await` statements are supported with this function. (Contributed by Tian Gao in [gh-132576](#).)
- Source code displayed in `pdb` will be syntax-highlighted. This feature can be controlled using the same methods as `PyREPL`, in addition to the newly added `colorize` argument of `pdb.Pdb`. (Contributed by Tian Gao and Łukasz Langa in [gh-133355](#).)

7.39 pickle

- Set the default protocol version on the `pickle` module to 5. For more details, see `pickle` protocols.
- Add notes for `pickle` serialization errors that allow to identify the source of the error. (Contributed by Serhiy Storchaka in [gh-122213](#).)

7.40 platform

- Add `platform.invalidate_caches()` to invalidate the cached results. (Contributed by Bénédict Tran in [gh-122549](#).)

7.41 pydoc

- Annotations in help output are now usually displayed in a format closer to that in the original source. (Contributed by Jelle Zijlstra in [gh-101552](#).)

7.42 socket

- Improve and fix support for Bluetooth sockets.
 - Fix support of Bluetooth sockets on NetBSD and DragonFly BSD. (Contributed by Serhiy Storchaka in [gh-132429](#).)
 - Fix support for `BTPROTO_HCI` on FreeBSD. (Contributed by Victor Stinner in [gh-111178](#).)
 - Add support for `BTPROTO_SCO` on FreeBSD. (Contributed by Serhiy Storchaka in [gh-85302](#).)
 - Add support for *cid* and *bdaddr_type* in the address for `BTPROTO_L2CAP` on FreeBSD. (Contributed by Serhiy Storchaka in [gh-132429](#).)
 - Add support for *channel* in the address for `BTPROTO_HCI` on Linux. (Contributed by Serhiy Storchaka in [gh-70145](#).)
 - Accept an integer as the address for `BTPROTO_HCI` on Linux. (Contributed by Serhiy Storchaka in [gh-132099](#).)
 - Return *cid* in `getsockname()` for `BTPROTO_L2CAP`. (Contributed by Serhiy Storchaka in [gh-132429](#).)
 - Add many new constants. (Contributed by Serhiy Storchaka in [gh-132734](#).)

7.43 ssl

- Indicate through `ssl.HAS_PHA` whether the `ssl` module supports TLSv1.3 post-handshake client authentication (PHA). (Contributed by Will Childs-Klein in [gh-128036](#).)

7.44 struct

- Support the `float complex` and `double complex` C types in the `struct` module (formatting characters 'F' and 'D' respectively). (Contributed by Sergey B Kirpichev in [gh-121249](#).)

7.45 symtable

- Expose the following `symtable.Symbol` methods:

```
- is_comp_cell()
```

```
- is_comp_iter()
```

```
- is_free_class()
```

(由 Bénédict Tran 於 gh-120029 貢獻。)

7.46 sys

- The previously undocumented special function `sys.getobjects()`, which only exists in specialized builds of Python, may now return objects from other interpreters than the one it's called in.
- Add `sys._is_immortal()` for determining if an object is immortal. (Contributed by Peter Bierma in [gh-128509](#).)
- On FreeBSD, `sys.platform` doesn't contain the major version anymore. It is always `'freebsd'`, instead of `'freebsd13'` or `'freebsd14'`.
- Raise `DeprecationWarning` for `sys._clear_type_cache()`. This function was deprecated in Python 3.13 but it didn't raise a runtime warning.

7.47 sys.monitoring

- Two new events are added: `BRANCH_LEFT` and `BRANCH_RIGHT`. The `BRANCH` event is deprecated.

7.48 sysconfig

- Add `ABIFLAGS` key to `sysconfig.get_config_vars()` on Windows. (Contributed by Xuehai Pan in [gh-131799](#).)

7.49 tarfile

- `data_filter()` now normalizes symbolic link targets in order to avoid path traversal attacks. (Contributed by Petr Viktorin in [gh-127987](#) and [CVE 2025-4138](#).)
- `extractall()` now skips fixing up directory attributes when a directory was removed or replaced by another kind of file. (Contributed by Petr Viktorin in [gh-127987](#) and [CVE 2024-12718](#).)
- `extract()` and `extractall()` now (re-)apply the extraction filter when substituting a link (hard or symbolic) with a copy of another archive member, and when fixing up directory attributes. The former raises a new exception, `LinkFallbackError`. (Contributed by Petr Viktorin for [CVE 2025-4330](#) and [CVE 2024-12718](#).)
- `extract()` and `extractall()` no longer extract rejected members when `errorlevel()` is zero. (Contributed by Matt Prodan and Petr Viktorin in [gh-112887](#) and [CVE 2025-4435](#).)

7.50 threading

- `threading.Thread.start()` now sets the operating system thread name to `threading.Thread.name`. (Contributed by Victor Stinner in [gh-59705](#).)

7.51 tkinter

- Make `tkinter` widget methods `after()` and `after_idle()` accept arguments passed by keyword. (Contributed by Zhikang Yan in [gh-126899](#).)
- Add ability to specify name for `tkinter.OptionMenu` and `tkinter.ttk.OptionMenu`. (Contributed by Zhikang Yan in [gh-130482](#).)

7.52 turtle

- Add context managers for `turtle.fill()`, `turtle.poly()` and `turtle.no_animation()`. (Contributed by Marie Roald and Yngve Mardal Moe in [gh-126350](#).)

7.53 types

- `types.UnionType` is now an alias for `typing.Union`. See [below](#) for more details. (Contributed by Jelle Zijlstra in [gh-105499](#).)

7.54 typing

- `types.UnionType` and `typing.Union` are now aliases for each other, meaning that both old-style unions (created with `Union[int, str]`) and new-style unions (`int | str`) now create instances of the same runtime type. This unifies the behavior between the two syntaxes, but leads to some differences in behavior that may affect users who introspect types at runtime:
 - Both syntaxes for creating a union now produce the same string representation in `repr()`. For example, `repr(Union[int, str])` is now `"int | str"` instead of `"typing.Union[int, str]"`.
 - Unions created using the old syntax are no longer cached. Previously, running `Union[int, str]` multiple times would return the same object (`Union[int, str] is Union[int, str]` would be `True`), but now it will return two different objects. Users should use `==` to compare unions for equality, not `is`. New-style unions have never been cached this way. This change could increase memory usage

for some programs that use a large number of unions created by subscripting `typing.Union`. However, several factors offset this cost: unions used in annotations are no longer evaluated by default in Python 3.14 because of [PEP 649](#); an instance of `types.UnionType` is itself much smaller than the object returned by `Union[]` was on prior Python versions; and removing the cache also saves some space. It is therefore unlikely that this change will cause a significant increase in memory usage for most users.

- Previously, old-style unions were implemented using the private class `typing._UnionGenericAlias`. This class is no longer needed for the implementation, but it has been retained for backward compatibility, with removal scheduled for Python 3.17. Users should use documented introspection helpers like `typing.get_origin()` and `typing.get_args()` instead of relying on private implementation details.
- It is now possible to use `typing.Union` itself in `isinstance()` checks. For example, `isinstance(int | str, typing.Union)` will return `True`; previously this raised `TypeError`.
- The `__args__` attribute of `typing.Union` objects is no longer writable.
- It is no longer possible to set any attributes on `typing.Union` objects. This only ever worked for dunder attributes on previous versions, was never documented to work, and was subtly broken in many cases.

(由 Jelle Zijlstra 於 [gh-105499](#) 貢獻。)

- `typing.TypeAliasType` now supports star unpacking.

7.55 unicodedata

- The Unicode database has been updated to Unicode 16.0.0.

7.56 unittest

- `unittest` output is now colored by default. This can be controlled by environment variables. (Contributed by Hugo van Kemenade in [gh-127221](#).)
- `unittest` discovery supports namespace package as start directory again. It was removed in Python 3.11. (Contributed by Jacob Walls in [gh-80958](#).)
- A number of new methods were added in the `TestCase` class that provide more specialized tests.
 - `assertHasAttr()` and `assertNotHasAttr()` check whether the object has a particular attribute.
 - `assertIsSubclass()` and `assertNotIsSubclass()` check whether the object is a subclass of a particular class, or of one of a tuple of classes.
 - `assertStartsWith()`, `assertNotStartsWith()`, `assertEndsWith()` and `assertNotEndsWith()` check whether the Unicode or byte string starts or ends with particular strings.

(由 Serhiy Storchaka 於 [gh-71339](#) 貢獻。)

7.57 urllib

- Upgrade HTTP digest authentication algorithm for `urllib.request` by supporting SHA-256 digest authentication as specified in [RFC 7616](#). (Contributed by Calvin Bui in [gh-128193](#).)
- Improve ergonomics and standards compliance when parsing and emitting `file:` URLs.

In `urllib.request.url2pathname()`:

- Accept a complete URL when the new `require_scheme` argument is set to `true`.
- Discard URL authority if it matches the local hostname.
- Discard URL authority if it resolves to a local IP address when the new `resolve_host` argument is set to `true`.
- Discard URL query and fragment components.

- Raise `URLError` if a URL authority isn't local, except on Windows where we return a UNC path as before.

In `urllib.request.pathname2url()`:

- Return a complete URL when the new `add_scheme` argument is set to true.
- Include an empty URL authority when a path begins with a slash. For example, the path `/etc/hosts` is converted to the URL `///etc/hosts`.

On Windows, drive letters are no longer converted to uppercase, and `:` characters not following a drive letter no longer cause an `OSError` exception to be raised.

(由 Barney Gale 於 [gh-125866](#) 貢獻。)

7.58 uuid

- Add support for UUID versions 6, 7, and 8 via `uuid.uuid6()`, `uuid.uuid7()`, and `uuid.uuid8()` respectively, as specified in [RFC 9562](#). (Contributed by Bénédict Tran in [gh-89083](#).)
- `uuid.NIL` and `uuid.MAX` are now available to represent the Nil and Max UUID formats as defined by [RFC 9562](#). (Contributed by Nick Pope in [gh-128427](#).)
- Allow to generate multiple UUIDs at once via `python -m uuid --count`. (Contributed by Simon Legner in [gh-131236](#).)

7.59 webbrowser

- Names in the `BROWSER` environment variable can now refer to already registered browsers for the `webbrowser` module, instead of always generating a new browser command.

This makes it possible to set `BROWSER` to the value of one of the supported browsers on macOS.

7.60 zipfile

- Added `ZipInfo._for_archive` to resolve suitable defaults for a `ZipInfo` object as used by `ZipFile.writestr`. (Contributed by Bénédict Tran in [gh-123424](#).)
- `zipfile.ZipFile.writestr()` now respects `SOURCE_DATE_EPOCH` that distributions can set centrally and have build tools consume this in order to produce reproducible output. (Contributed by Jiahao Li in [gh-91279](#).)

8 最佳化

- The import time for several standard library modules has been improved, including `annotationlib`, `ast`, `asyncio`, `base64`, `cmd`, `csv`, `gettext`, `importlib.util`, `locale`, `mimetypes`, `optparse`, `pickle`, `pprint`, `pstats`, `shlex`, `socket`, `string`, `subprocess`, `threading`, `tomllib`, `types`, and `zipfile`. (Contributed by Adam Turner, Bénédict Tran, Chris Markiewicz, Eli Schwartz, Hugo van Kemenade, Jelle Zijlstra, and others in [gh-118761](#).)

8.1 asyncio

- Standard benchmark results have improved by 10-20%, following the implementation of a new per-thread double linked list for native tasks, also reducing memory usage. This enables external introspection tools such as `python -m asyncio pstree` to introspect the call graph of asyncio tasks running in all threads. (Contributed by Kumar Aditya in [gh-107803](#).)
- The module now has first class support for free-threading builds. This enables parallel execution of multiple event loops across different threads, scaling linearly with the number of threads. (Contributed by Kumar Aditya in [gh-128002](#).)

8.10 zlib

- On Windows, `zlib-ng` is now used as the implementation of the `zlib` module in the default binaries. There are no known incompatibilities between `zlib-ng` and the previously-used `zlib` implementation. This should result in better performance at all compression levels.

It is worth noting that `zlib.Z_BEST_SPEED(1)` may result in significantly less compression than the previous implementation, whilst also significantly reducing the time taken to compress.

(由 Steve Dower 於 [gh-91349](#) 貢獻。)

9 已移除

9.1 argparse

- Remove the `type`, `choices`, and `metavar` parameters of `BooleanOptionalAction`. These have been deprecated since Python 3.12. (Contributed by Nikita Sobolev in [gh-118805](#).)
- Calling `add_argument_group()` on an argument group now raises a `ValueError`. Similarly, `add_argument_group()` or `add_mutually_exclusive_group()` on a mutually exclusive group now both raise `ValueErrors`. This 'nesting' was never supported, often failed to work correctly, and was unintentionally exposed through inheritance. This functionality has been deprecated since Python 3.11. (Contributed by Savannah Ostrowski in [gh-127186](#).)

9.2 ast

- Remove the following classes, which have been deprecated aliases of `Constant` since Python 3.8 and have emitted deprecation warnings since Python 3.12:

- `Bytes`
- `Ellipsis`
- `NameConstant`
- `Num`
- `Str`

As a consequence of these removals, user-defined `visit_Num`, `visit_Str`, `visit_Bytes`, `visit_NameConstant` and `visit_Ellipsis` methods on custom `NodeVisitor` subclasses will no longer be called when the `NodeVisitor` subclass is visiting an AST. Define a `visit_Constant` method instead.

(由 Alex Waygood 於 [gh-119562](#) 貢獻。)

- Remove the following deprecated properties on `ast.Constant`, which were present for compatibility with the now-removed AST classes:

- `Constant.n`
- `Constant.s`

Use `Constant.value` instead. (Contributed by Alex Waygood in [gh-119562](#).)

9.3 asyncio

- Remove the following classes, methods, and functions, which have been deprecated since Python 3.12:

- `AbstractChildWatcher`
- `FastChildWatcher`
- `MultiLoopChildWatcher`
- `PidfdChildWatcher`
- `SafeChildWatcher`

- ThreadedChildWatcher
- AbstractEventLoopPolicy.get_child_watcher()
- AbstractEventLoopPolicy.set_child_watcher()
- get_child_watcher()
- set_child_watcher()

(由 Kumar Aditya 於 [gh-120804](#) 貢獻。)

- `asyncio.get_event_loop()` now raises a `RuntimeError` if there is no current event loop, and no longer implicitly creates an event loop.

(由 Kumar Aditya 於 [gh-126353](#) 貢獻。)

There's a few patterns that use `asyncio.get_event_loop()`, most of them can be replaced with `asyncio.run()`.

If you're running an async function, simply use `asyncio.run()`.

Before:

```
async def main():
    ...

loop = asyncio.get_event_loop()
try:
    loop.run_until_complete(main())
finally:
    loop.close()
```

After:

```
async def main():
    ...

asyncio.run(main())
```

If you need to start something, for example, a server listening on a socket and then run forever, use `asyncio.run()` and an `asyncio.Event`.

Before:

```
def start_server(loop): ...

loop = asyncio.get_event_loop()
try:
    start_server(loop)
    loop.run_forever()
finally:
    loop.close()
```

After:

```
def start_server(loop): ...

async def main():
    start_server(asyncio.get_running_loop())
    await asyncio.Event().wait()

asyncio.run(main())
```

If you need to run something in an event loop, then run some blocking code around it, use `asyncio.Runner`.

Before:

```
async def operation_one(): ...
def blocking_code(): ...
async def operation_two(): ...

loop = asyncio.get_event_loop()
try:
    loop.run_until_complete(operation_one())
    blocking_code()
    loop.run_until_complete(operation_two())
finally:
    loop.close()
```

After:

```
async def operation_one(): ...
def blocking_code(): ...
async def operation_two(): ...

with asyncio.Runner() as runner:
    runner.run(operation_one())
    blocking_code()
    runner.run(operation_two())
```

9.4 email

- Remove `email.utils.localtime()`'s `isdst` parameter, which was deprecated in and has been ignored since Python 3.12. (Contributed by Hugo van Kemenade in [gh-118798](#).)

9.5 importlib.abc

- Remove deprecated `importlib.abc` classes:
 - `ResourceReader` (use `TraversableResources`)
 - `Traversable` (use `Traversable`)
 - `TraversableResources` (use `TraversableResources`)(由 Jason R. Coombs 和 Hugo van Kemenade 貢獻於 [gh-93963](#).)

9.6 itertools

- Remove support for copy, deepcopy, and pickle operations from `itertools` iterators. These have emitted a `DeprecationWarning` since Python 3.12. (Contributed by Raymond Hettinger in [gh-101588](#).)

9.7 pathlib

- Remove support for passing additional keyword arguments to `Path`. In previous versions, any such arguments are ignored. (Contributed by Barney Gale in [gh-74033](#).)
- Remove support for passing additional positional arguments to `PurePath.relative_to()` and `is_relative_to()`. In previous versions, any such arguments are joined onto `other`. (Contributed by Barney Gale in [gh-78707](#).)

9.8 pkgutil

- Remove the `get_loader()` and `find_loader()` functions, which have been deprecated since Python 3.12. (Contributed by Bénédict Tran in [gh-97850](#).)

9.9 `pty`

- Remove the `master_open()` and `slave_open()` functions, which have been deprecated since Python 3.12. Use `pty.openpty()` instead. (Contributed by Nikita Sobolev in [gh-118824](#).)

9.10 `sqlite3`

- Remove `version` and `version_info` from the `sqlite3` module; use `sqlite_version` and `sqlite_version_info` for the actual version number of the runtime SQLite library. (Contributed by Hugo van Kemenade in [gh-118924](#).)
- Using a sequence of parameters with named placeholders now raises a `ProgrammingError`, having been deprecated since Python 3.12. (Contributed by Erlend E. Aasland in [gh-118928](#) and [gh-101693](#).)

9.11 `urllib`

- Remove the `Quoter` class from `urllib.parse`, which has been deprecated since Python 3.11. (Contributed by Nikita Sobolev in [gh-118827](#).)
- Remove the `URLOpener` and `FancyURLOpener` classes from `urllib.request`, which have been deprecated since Python 3.3.

`myopener.open()` can be replaced with `urlopen()`. `myopener.retrieve()` can be replaced with `urlretrieve()`. Customisations to the opener classes can be replaced by passing customized handlers to `build_opener()`. (Contributed by Barney Gale in [gh-84850](#).)

10 已用

10.1 New deprecations

- Passing a complex number as the *real* or *imag* argument in the `complex()` constructor is now deprecated; complex numbers should only be passed as a single positional argument. (Contributed by Serhiy Storchaka in [gh-109218](#).)
- `argparse`:
 - Passing the undocumented keyword argument `prefix_chars` to the `add_argument_group()` method is now deprecated. (Contributed by Savannah Ostrowski in [gh-125563](#).)
 - Deprecated the `argparse.FileType` type converter. Anything relating to resource management should be handled downstream, after the arguments have been parsed. (Contributed by Serhiy Storchaka in [gh-58032](#).)
- `asyncio`:
 - The `asyncio.iscoroutinefunction()` is now deprecated and will be removed in Python 3.16; use `inspect.iscoroutinefunction()` instead. (Contributed by Jiahao Li and Kumar Aditya in [gh-122875](#).)
 - The `asyncio` policy system is deprecated and will be removed in Python 3.16. In particular, the following classes and functions are deprecated:
 - * `asyncio.AbstractEventLoopPolicy`
 - * `asyncio.DefaultEventLoopPolicy`
 - * `asyncio.WindowsSelectorEventLoopPolicy`
 - * `asyncio.WindowsProactorEventLoopPolicy`
 - * `asyncio.get_event_loop_policy()`
 - * `asyncio.set_event_loop_policy()`

Users should use `asyncio.run()` or `asyncio.Runner` with the `loop_factory` argument to use the desired event loop implementation.

For example, to use `asyncio.SelectorEventLoop` on Windows:

```
import asyncio

async def main():
    ...

asyncio.run(main(), loop_factory=asyncio.SelectorEventLoop)
```

(由 Kumar Aditya 於 [gh-127949](#) 貢獻。)

- **codecs:** The `codecs.open()` function is now deprecated, and will be removed in a future version of Python. Use `open()` instead. (Contributed by Inada Naoki in [gh-133036](#).)
- **ctypes:**
 - On non-Windows platforms, setting `Structure._pack_` to use a MSVC-compatible default memory layout is now deprecated in favor of setting `Structure._layout_` to 'ms', and will be removed in Python 3.19. (Contributed by Petr Viktorin in [gh-131747](#).)
 - Calling `ctypes.POINTER()` on a string is now deprecated. Use incomplete types for self-referential structures. Also, the internal `ctypes._pointer_type_cache` is deprecated. See `ctypes.POINTER()` for updated implementation details. (Contributed by Sergey Myrianov in [gh-100926](#).)
- **functools:** Calling the Python implementation of `functools.reduce()` with *function* or *sequence* as keyword arguments is now deprecated; the parameters will be made positional-only in Python 3.16. (Contributed by Kirill Podoprigora in [gh-121676](#).)
- **logging:** Support for custom logging handlers with the *strm* argument is now deprecated and scheduled for removal in Python 3.16. Define handlers with the *stream* argument instead. (Contributed by Mariusz Felisiak in [gh-115032](#).)
- **mimetypes:** Valid extensions are either empty or must start with '.' for `mimetypes.MimeTypes.add_type()`. Undotted extensions are deprecated and will raise a `ValueError` in Python 3.16. (Contributed by Hugo van Kemenade in [gh-75223](#).)
- **nturl2path:** This module is now deprecated. Call `urllib.request.url2pathname()` and `pathname2url()` instead. (Contributed by Barney Gale in [gh-125866](#).)
- **os:** The `os.popen()` and `os.spawn*` functions are now soft deprecated. They should no longer be used to write new code. The `subprocess` module is recommended instead. (Contributed by Victor Stinner in [gh-120743](#).)
- **pathlib:** `pathlib.PurePath.as_uri()` is now deprecated and scheduled for removal in Python 3.19. Use `pathlib.Path.as_uri()` instead. (Contributed by Barney Gale in [gh-123599](#).)
- **pdb:** The undocumented `pdb.Pdb.curframe_locals` attribute is now a deprecated read-only property, which will be removed in a future version of Python. The low overhead dynamic frame locals access added in Python 3.13 by [PEP 667](#) means the frame locals cache reference previously stored in this attribute is no longer needed. Derived debuggers should access `pdb.Pdb.curframe.f_locals` directly in Python 3.13 and later versions. (Contributed by Tian Gao in [gh-124369](#) and [gh-125951](#).)
- **symtable:** Deprecate `symtable.Class.get_methods()` due to the lack of interest, scheduled for removal in Python 3.16. (Contributed by Bénédict Tran in [gh-119698](#).)
- **tkinter:** The `tkinter.Variable` methods `trace_variable()`, `trace_vdelete()` and `trace_vinfo()` are now deprecated. Use `trace_add()`, `trace_remove()` and `trace_info()` instead. (Contributed by Serhiy Storchaka in [gh-120220](#).)
- **urllib.parse:** Accepting objects with false values (like 0 and []) except empty strings, bytes-like objects and None in `parse_qs1()` and `parse_qs()` is now deprecated. (Contributed by Serhiy Storchaka in [gh-116897](#).)

10.2 Python 3.15 中待移除的項目

- 引入系統 (import system):
 - 在模組上設定 `__cached__` 而沒有設定 `__spec__.cached` 的做法已被廢用。在 Python 3.15 中，引入系統或標準函式庫將不再設定或考慮 `__cached__`。([gh-97879](#))
 - 在模組上設定 `__package__` 而沒有設定 `__spec__.parent` 的做法已被廢用。在 Python 3.15 中，引入系統或標準函式庫將不再設定或考慮 `__package__`。([gh-97879](#))
- ctypes:
 - 自 Python 3.13 起，未記入的 `ctypes.SetPointerType()` 函式已被廢用。
- http.server:
 - 過時且很少使用的 `CGIHTTPRequestHandler` 自 Python 3.13 起已被廢用。不存在直接的替代。任何東西都比 CGI 更好地將 Web 伺服器與請求處理程序介接起來。
 - 自 Python 3.13 起，`python -m http.server` 命令列界面的 `--cgi` 旗標已被廢用。
- importlib:
 - `load_module()` method: 請改用 `exec_module()`。
- locale:
 - `getdefaultlocale()` 已在 Python 3.11 中被廢用，原本計劃在 Python 3.13 中移除 ([gh-90817](#))，但被延後至 Python 3.15。請改用 `getlocale()`、`setlocale()` 和 `getencoding()`。(由 Hugo van Kemenade 於 [gh-111187](#) 貢獻。)
- pathlib:
 - `PurePath.is_reserved()` 已自 Python 3.13 被廢用。請用 `os.path.isreserved()` 來偵測 Windows 上的保留路徑。
- platform:
 - 自 Python 3.13 起，`java_ver()` 已被廢用。此函式僅對 Jython 支援有用，具有令人困惑的 API，基本上未經測試。
- sysconfig:
 - `sysconfig.is_python_build()` 的 `check_home` 引數自 Python 3.12 起已被廢用。
- threading:
 - `RLock()` 在 Python 3.15 中將不接受任何引數。自 Python 3.14 起，傳遞任何引數的用法已被廢用，因為 Python 版本不允許任何引數，但 C 版本允許任意數量的位置或關鍵字引數，廢忽略每個引數。
- types:
 - `types.CodeType`: 自 3.10 起，存取 `co_notab` 已在 [PEP 626](#) 中被廢用，計劃在 3.12 中移除，但只在 3.12 中於適當時發出 `DeprecationWarning`。可能在 3.15 中移除。(由 Nikita Sobolev 於 [gh-101866](#) 貢獻。)
- typing:
 - 用於建立 `NamedTuple` 類型的未以文件記入之關鍵字引數語法 (`Point = NamedTuple("Point", x=int, y=int)`) 已自 Python 3.13 廢用。請改用基於類型的語法或函式語法 (functional syntax)。
 - 當使用 `TypedDict` 的函式語法時，未傳遞值給 `fields` 參數 (`TD = TypedDict("TD")`) 或傳遞 `None` (`TD = TypedDict("TD", None)`) 的做法自 Python 3.13 起已被廢用。請使用 `class TD(TypedDict): pass` 或 `TD = TypedDict("TD", {})` 來建立具有零個欄位的 `TypedDict`。
 - 自 Python 3.13 起，`typing.no_type_check_decorator()` 裝飾器函式已被廢用。在 `typing` 模組中使用了八年之後，它尚未得到任何主要型別檢查器的支援。
- wave:

- 已☑用 `Wave_read` 和 `Wave_write` 類☑的 `getmark()`、`setmark()` 和 `getmarkers()` 方法自 Python 3.13 被☑用。
- `zipimport`:
 - `load_module()` 自 Python 3.10 被☑用。請改用 `exec_module()`。(由 Jiahao Li 於 [gh-125746](#) 貢獻。)

10.3 Python 3.16 中待移除的項目

- 引入系統 (import system):
 - 在模組上設定 `__loader__` 而☑有設定 `__spec__.loader` 的做法將於 Python 3.16 被☑用。在 Python 3.16 中，引入系統或標準函式庫將不再設定或考慮 `__loader__`。
- `array`:
 - 自 Python 3.3 起，'u' 格式碼 (`wchar_t`) 在文件中已被☑用，自 Python 3.13 起在 runtime 已被☑用。請使用 'w' 格式碼 (`Py_UCS4`) 來取代 Unicode 字元。
- `asyncio`:
 - `asyncio.iscoroutinefunction()` 已被☑用☑將在 Python 3.16 中移除；請改用 `inspect.iscoroutinefunction()`。(由 Jiahao Li 和 Kumar Aditya 於 [gh-122875](#) 貢獻。)
 - `asyncio` 策略系統已被☑用☑將在 Python 3.16 中移除。特☑是以下類☑和函式已被☑用：

```
* asyncio.AbstractEventLoopPolicy
* asyncio.DefaultEventLoopPolicy
* asyncio.WindowsSelectorEventLoopPolicy
* asyncio.WindowsProactorEventLoopPolicy
* asyncio.get_event_loop_policy()
* asyncio.set_event_loop_policy()
```

使用者應該使用 `asyncio.run()` 或 `asyncio.Runner` 搭配 `loop_factory` 來使用所需的事件☑圈實作。

例如在 Windows 上使用 `asyncio.SelectorEventLoop`:

```
import asyncio

async def main():
    ...

asyncio.run(main(), loop_factory=asyncio.SelectorEventLoop)
```

(由 Kumar Aditya 於 [gh-127949](#) 貢獻。)

- `builtins`:
 - 自 Python 3.12 起，布林型☑的位元反轉 `~True` 或 `~False` 已被☑用，因☑它會☑生不預期且不直觀的結果 (`-2` 和 `-1`)。使用 `not x` 代替布林值的邏輯否定。在極少數情☑下，你需要對底層的整數進行位元反轉，請明確轉☑☑ `~int(x)` (`~int(x)`)。
- `functools`:
 - 自 Python 3.14 起，使用 *function* 或 *sequence* 關鍵字引數呼叫 `functools.reduce()` 的 Python 實作已被☑用。
- `logging`:
 - 對具有 *strm* 引數的自訂日☑記☑處理函式的支援已被☑用，☑計劃在 Python 3.16 中移除。請改用 *stream* 引數。(由 Mariusz Felisiak 於 [gh-115032](#) 貢獻。)
- `mimetypes`:

- 有效的副檔名以'.' 開頭或對 `mimetypes.MimeTypes.add_type()` 的空字串。未加點的副檔名已被禁用，將在 Python 3.16 中引發 `ValueError`。(由 Hugo van Kemenade 於 [gh-75223](#) 貢獻。)
- `shutil`:
 - 自 Python 3.14 起，`ExecError` 例外已被禁用。自 Python 3.4 以來，它尚未被 `shutil` 中的任何函式使用，現在是 `RuntimeError` 的別名。
- `symtable`:
 - 自 Python 3.14 起，`Class.get_methods` 方法已被禁用。
- `sys`:
 - 自 Python 3.13 起，`_enablelegacywindowsfsencoding()` 函式已被禁用。請改用 `PYTHONLEGACYWINDOWSFSENCODING` 環境變數。
- `sysconfig`:
 - 自 Python 3.14 起，`sysconfig.expand_makefile_vars()` 函式已被禁用。請改用 `sysconfig.get_paths()` 的 `vars` 引數。
- `tarfile`:
 - 自 Python 3.13 起，未以文件記錄和未被使用的 `TarFile.tarfile` 屬性已被禁用。

10.4 Python 3.17 中待移除的項目

- `collections.abc`:
 - `collections.abc.ByteString` is scheduled for removal in Python 3.17.
 Use `isinstance(obj, collections.abc.Buffer)` to test if `obj` implements the buffer protocol at runtime. For use in type annotations, either use `Buffer` or a union that explicitly specifies the types your code supports (e.g., `bytes | bytearray | memoryview`).
`ByteString` was originally intended to be an abstract class that would serve as a supertype of both `bytes` and `bytearray`. However, since the ABC never had any methods, knowing that an object was an instance of `ByteString` never actually told you anything useful about the object. Other common buffer types such as `memoryview` were also never understood as subtypes of `ByteString` (either at runtime or by static type checkers).
 See [PEP 688](#) for more details. (Contributed by Shantanu Jain in [gh-91896](#).)
- `typing`:
 - 在 Python 3.14 之前，舊式聯集是使用私有類 `typing._UnionGenericAlias` 實作的。這個類不再被需要，但為了向後相容性而保留，它計劃將在 Python 3.17 中移除。使用者應該改用文件中記錄的省輔助函式，例如 `typing.get_origin()` 和 `typing.get_args()`，或者依賴私有實作細節。
 - `typing.ByteString`, deprecated since Python 3.9, is scheduled for removal in Python 3.17.
 Use `isinstance(obj, collections.abc.Buffer)` to test if `obj` implements the buffer protocol at runtime. For use in type annotations, either use `Buffer` or a union that explicitly specifies the types your code supports (e.g., `bytes | bytearray | memoryview`).
`ByteString` was originally intended to be an abstract class that would serve as a supertype of both `bytes` and `bytearray`. However, since the ABC never had any methods, knowing that an object was an instance of `ByteString` never actually told you anything useful about the object. Other common buffer types such as `memoryview` were also never understood as subtypes of `ByteString` (either at runtime or by static type checkers).
 See [PEP 688](#) for more details. (Contributed by Shantanu Jain in [gh-91896](#).)

10.5 Python 3.19 中待移除的項目

- `ctypes`:
 - 在非 Windows 平台上，透過設定 `_pack_` 而沒有設定 `_layout_` 來隱式地切到與 MSVC 相容的結構布局。

10.6 未來版本中的待移除項目

以下 API 將在未來被移除，雖然目前尚未安排移除日期。

- `argparse`:
 - 巢狀引數群組和巢狀互斥群組已被廢用。
 - 將未以文件記的關鍵字引數 `prefix_chars` 傳遞給 `add_argument_group()` 的做法現在已被廢用。
 - `argparse.FileType` 型轉器已被廢用。
- `builtins`:
 - `bool(NotImplemented)`。
 - 生成器：`throw(type, exc, tb)` 和 `athrow(type, exc, tb)` 簽名已被廢用：請改用 `throw(exc)` 和 `athrow(exc)`，單引數簽名。
 - 目前 Python 接受數值字面值後面立即接關鍵字，例如 `0 in x`、`1 or x`、`0 if 1 else 2`。它讓表達式模糊且容易混淆，如 `[0x1 for x in y]`（可以解釋為 `[0x1 for x in y]` 或 `[0x1f or x in y]`）。如果數值字面值後面立即接 `and`、`else`、`for`、`if`、`in`、`is` 和 `or` 之一的關鍵字，則會引發語法警告。在未來版本中，它將被更改為語法錯誤。[\(gh-87999\)](#)
 - `__index__()` 和 `__int__()` 方法回傳非 `int` 型別的支援：這些方法將需要回傳 `int` 的嚴格子類實例。
 - 回傳 `float` 嚴格子類 `__float__()` 方法的支援：這些方法將需要回傳 `float` 的實例。
 - 回傳 `complex` 嚴格子類 `__complex__()` 方法的支援：這些方法將需要回傳 `complex` 的實例。
 - 將 `int()` 委派給 `__trunc__()` 方法。
 - 在 `complex()` 建構子中將數作 `real` 或 `imag` 引數傳遞現在已被廢用；它應該只作單個位置引數傳遞。（由 Serhiy Storchaka 於 [gh-109218](#) 貢獻。）
- `calendar`: `calendar.January` 和 `calendar.February` 常數已被廢用被 `calendar.JANUARY` 和 `calendar.FEBRUARY` 取代。（由 Prince Roshan 於 [gh-103636](#) 貢獻。）
- `codecs`: 請改用 `open()` 而非 `codecs.open()`。[\(gh-133038\)](#)
- `codeobject.co_nnotab`: 請改用 `codeobject.co_lines()` 方法。
- `datetime`:
 - `utcnow()`: 請改用 `datetime.datetime.now(tz=datetime.UTC)`。
 - `utcfromtimestamp()`: 請改用 `datetime.datetime.fromtimestamp(timestamp, tz=datetime.UTC)`。
- `gettext`: 數值必須是整數。
- `importlib`:
 - `cache_from_source()` `debug_override` 參數已被廢用：請改用 `optimization` 參數。
- `importlib.metadata`:
 - `EntryPoint`s 元組介面。
 - 回傳值上的隱式 `None`。
- `logging`: 自 Python 3.3 起，`warn()` 方法已被廢用，請改用 `warning()`。

- `mailbox`: 已改用 `StringIO` 輸入和文本模式，請改用 `BytesIO` 和二進位模式。
- `os`: 在多執行緒行程中呼叫 `os.register_at_fork()`。
- `pydoc.ErrorDuringImport`: `exc_info` 參數的元組值已被改用，請用例外實例。
- `re`: 現在對正規表示式中的數值群組參照和群組名稱用了更嚴格的規則。現在只有 ASCII 數碼序列被接受作數值參照。位元組模式和替字串中的群組名稱現在只能包含 ASCII 字母、數碼和底。 (由 Serhiy Storchaka 於 [gh-91760](#) 貢獻。)
- `sre_compile`、`sre_constants` 和 `sre_parse` 模組。
- `shutil`: `rmtree()` 的 `onerror` 參數在 Python 3.12 中已被改用；請改用 `onexc` 參數。
- `ssl` 選項和協定：
 - 不帶協定引數的 `ssl.SSLContext` 已被改用。
 - `ssl.SSLContext`: `set_npn_protocols()` 和 `selected_npn_protocol()` 已被改用：請改用 `ALPN`。
 - `ssl.OP_NO_SSL*` 選項
 - `ssl.OP_NO_TLS*` 選項
 - `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`
- `threading` 方法：
 - `threading.Condition.notifyAll()`: 請用 `notify_all()`。
 - `threading.Event.isSet()`: 請用 `is_set()`。
 - `threading.Thread.isDaemon()`、`threading.Thread.setDaemon()`: 請用 `threading.Thread.daemon` 屬性。
 - `threading.Thread.getName()`、`threading.Thread.setName()`: 請用 `threading.Thread.name` 屬性。
 - `threading.currentThread()`: 請用 `threading.current_thread()`。
 - `threading.activeCount()`: 請用 `threading.active_count()`。
- `typing.Text` ([gh-92332](#))。
- 部類 `typing._UnionGenericAlias` 不再用於實作 `typing.Union`。為了保持與此私有類使用者的相容性，直到至少 Python 3.17 都將提供一個相容性 shim。 (由 Jelle Zijlstra 於 [gh-105499](#) 貢獻。)
- `unittest.IsolatedAsyncioTestCase`: 從測試案例中回傳非 `None` 的值已被改用。
- `urllib.parse` 已改用函式：請改用 `urlparse()`。
 - `splitattr()`
 - `splithost()`
 - `splitnport()`
 - `splitpasswd()`

- `splitport()`
- `splitquery()`
- `splittag()`
- `splitttype()`
- `splituser()`
- `splitvalue()`
- `to_bytes()`
- `wsgiref: SimpleHandler.stdout.write()` 不應該進行部分寫入。
- `xml.etree.ElementTree: 已☑用對 Element 的真值測試`。在未來版本中，它將始終回傳 `True`。請改用明確的 `len(elem)` 或 `elem is not None` 測試。
- `sys._clear_type_cache()` 已被☑用：請改用 `sys._clear_internal_caches()`。

11 CPython 位元組碼變更

- Replaced the opcode `BINARY_SUBSCR` by the `BINARY_OP` opcode with the `NB_SUBSCR` oparg. (Contributed by Irit Katriel in [gh-100239](#).)
- Add the `BUILD_INTERPOLATION` and `BUILD_TEMPLATE` opcodes to construct new `Interpolation` and `Template` instances, respectively. (Contributed by Lysandros Nikolaou and others in [gh-132661](#); see also *PEP 750: Template strings*).
- Remove the `BUILD_CONST_KEY_MAP` opcode. Use `BUILD_MAP` instead. (Contributed by Mark Shannon in [gh-122160](#).)
- Replace the `LOAD_ASSERTION_ERROR` opcode with `LOAD_COMMON_CONSTANT` and add support for loading `NotImplementedError`.
- Add the `LOAD_FAST_BORROW` and `LOAD_FAST_BORROW_LOAD_FAST_BORROW` opcodes to reduce reference counting overhead when the interpreter can prove that the reference in the frame outlives the reference loaded onto the stack. (Contributed by Matt Page in [gh-130704](#).)
- Add the `LOAD_SMALL_INT` opcode, which pushes a small integer equal to the oparg to the stack. The `RETURN_CONST` opcode is removed as it is no longer used. (Contributed by Mark Shannon in [gh-125837](#).)
- Add the new `LOAD_SPECIAL` instruction. Generate code for `with` and `async with` statements using the new instruction. Removed the `BEFORE_WITH` and `BEFORE_ASYNC_WITH` instructions. (Contributed by Mark Shannon in [gh-120507](#).)
- Add the `POP_ITER` opcode to support 'virtual' iterators. (Contributed by Mark Shannon in [gh-132554](#).)

11.1 Pseudo-instructions

- Add the `ANNOTATIONS_PLACEHOLDER` pseudo instruction to support partially executed module-level annotations with *deferred evaluation of annotations*. (Contributed by Jelle Zijlstra in [gh-130907](#).)
- Add the `BINARY_OP_EXTEND` pseudo instruction, which executes a pair of functions (guard and specialization functions) accessed from the inline cache. (Contributed by Irit Katriel in [gh-100239](#).)
- Add three specializations for `CALL_KW`; `CALL_KW_PY` for calls to Python functions, `CALL_KW_BOUND_METHOD` for calls to bound methods, and `CALL_KW_NON_PY` for all other calls. (Contributed by Mark Shannon in [gh-118093](#).)
- Add the `JUMP_IF_TRUE` and `JUMP_IF_FALSE` pseudo instructions, conditional jumps which do not impact the stack. Replaced by the sequence `COPY 1, TO_BOOL, POP_JUMP_IF_TRUE/FALSE`. (Contributed by Irit Katriel in [gh-124285](#).)
- Add the `LOAD_CONST_MORTAL` pseudo instruction. (Contributed by Mark Shannon in [gh-128685](#).)

- Add the `LOAD_CONST_IMMORTAL` pseudo instruction, which does the same as `LOAD_CONST`, but is more efficient for immortal objects. (Contributed by Mark Shannon in [gh-125837](#).)
- Add the `NOT_TAKEN` pseudo instruction, used by `sys.monitoring` to record branch events (such as `BRANCH_LEFT`). (Contributed by Mark Shannon in [gh-122548](#).)

12 C API 變更

12.1 C API 中的新功能

- Add `Py_PACK_VERSION()` and `Py_PACK_FULL_VERSION()`, two new macros for bit-packing Python version numbers. This is useful for comparisons with `Py_Version` or `PY_VERSION_HEX`. (Contributed by Petr Viktorin in [gh-128629](#).)
- Add `PyBytes_Join(sep, iterable)` function, similar to `sep.join(iterable)` in Python. (Contributed by Victor Stinner in [gh-121645](#).)
- Add functions to manipulate the configuration of the current runtime Python interpreter (*PEP 741: Python configuration C API*):

```
- PyConfig_Get()
- PyConfig_GetInt()
- PyConfig_Set()
- PyConfig_Names()
```

(由 Victor Stinner 於 [gh-107954](#) 貢獻。)

- 新增用於配置 Python 初始化的函式 (*PEP 741: Python 配置 C API*):

```
- Py_InitializeFromInitConfig()
- PyInitConfig_AddModule()
- PyInitConfig_Create()
- PyInitConfig_Free()
- PyInitConfig_FreeStrList()
- PyInitConfig_GetError()
- PyInitConfig_GetExitCode()
- PyInitConfig_GetInt()
- PyInitConfig_GetStr()
- PyInitConfig_GetStrList()
- PyInitConfig_HasOption()
- PyInitConfig_SetInt()
- PyInitConfig_SetStr()
- PyInitConfig_SetStrList()
```

(由 Victor Stinner 於 [gh-107954](#) 貢獻。)

- Add `Py_fopen()` function to open a file. This works similarly to the standard C `fopen()` function, instead accepting a Python object for the *path* parameter and setting an exception on error. The corresponding new `Py_fclose()` function should be used to close a file. (Contributed by Victor Stinner in [gh-127350](#).)
- Add `Py_HashBuffer()` to compute and return the hash value of a buffer. (Contributed by Antoine Pitrou and Victor Stinner in [gh-122854](#).)
- Add `PyImport_ImportModuleAttr()` and `PyImport_ImportModuleAttrString()` helper functions to import a module and get an attribute of the module. (Contributed by Victor Stinner in [gh-128911](#).)

- Add `PyIter_NextItem()` to replace `PyIter_Next()`, which has an ambiguous return value. (Contributed by Irit Katriel and Erlend Aasland in [gh-105201](#).)
- Add `PyLong_GetSign()` function to get the sign of `int` objects. (Contributed by Sergey B Kirpichev in [gh-116560](#).)
- Add `PyLong_IsPositive()`, `PyLong_IsNegative()` and `PyLong_IsZero()` for checking if `PyLongObject` is positive, negative, or zero, respectively. (Contributed by James Roy and Sergey B Kirpichev in [gh-126061](#).)
- Add new functions to convert C `<stdint.h>` numbers to/from Python `int` objects:
 - `PyLong_AsInt32()`
 - `PyLong_AsInt64()`
 - `PyLong_AsUInt32()`
 - `PyLong_AsUInt64()`
 - `PyLong_FromInt32()`
 - `PyLong_FromInt64()`
 - `PyLong_FromUInt32()`
 - `PyLong_FromUInt64()`
 (由 Victor Stinner 於 [gh-120389](#) 貢獻。)
- Add a new import and export API for Python `int` objects (**PEP 757**):
 - `PyLong_GetNativeLayout()`
 - `PyLong_Export()`
 - `PyLong_FreeExport()`
 - `PyLongWriter_Create()`
 - `PyLongWriter_Finish()`
 - `PyLongWriter_Discard()`
 (由 Sergey B Kirpichev 和 Victor Stinner 於 [gh-102471](#) 貢獻。)
- Add `PyMonitoring_FireBranchLeftEvent()` and `PyMonitoring_FireBranchRightEvent()` for generating `BRANCH_LEFT` and `BRANCH_RIGHT` events, respectively. (Contributed by Mark Shannon in [gh-122548](#).)
- Add `PyType_Freeze()` function to make a type immutable. (Contributed by Victor Stinner in [gh-121654](#).)
- Add `PyType_GetBaseByToken()` and `Py_tp_token` slot for easier superclass identification, which attempts to resolve the type checking issue mentioned in **PEP 630**. (Contributed in [gh-124153](#).)
- Add a new `PyUnicode_Equal()` function to test if two strings are equal. The function is also added to the Limited C API. (Contributed by Victor Stinner in [gh-124502](#).)
- Add a new `PyUnicodeWriter` API to create a Python `str` object, with the following functions:
 - `PyUnicodeWriter_Create()`
 - `PyUnicodeWriter_DecodeUTF8Stateful()`
 - `PyUnicodeWriter_Discard()`
 - `PyUnicodeWriter_Finish()`
 - `PyUnicodeWriter_Format()`
 - `PyUnicodeWriter_WriteASCII()`
 - `PyUnicodeWriter_WriteChar()`
 - `PyUnicodeWriter_WriteRepr()`

- PyUnicodeWriter_WriteStr()
- PyUnicodeWriter_WriteSubstring()
- PyUnicodeWriter_WriteUCS4()
- PyUnicodeWriter_WriteUTF8()
- PyUnicodeWriter_WriteWideChar()

(由 Victor Stinner 於 [gh-119182](#) 貢獻。)

- The `k` and `K` formats in `PyArg_ParseTuple()` and similar functions now use `__index__()` if available, like all other integer formats. (Contributed by Serhiy Storchaka in [gh-112068](#).)
- Add support for a new `p` format unit in `Py_BuildValue()` that produces a Python `bool` object from a C integer. (Contributed by Pablo Galindo in [bpo-45325](#).)
- Add `PyUnstable_IsImmortal()` for determining if an object is immortal, for debugging purposes. (Contributed by Peter Bierma in [gh-128509](#).)
- Add `PyUnstable_Object_EnableDeferredRefCount()` for enabling deferred reference counting, as outlined in [PEP 703](#).
- Add `PyUnstable_Object_IsUniquelyReferenced()` as a replacement for `Py_REFCNT(op) == 1` on free threaded builds. (Contributed by Peter Bierma in [gh-133140](#).)
- Add `PyUnstable_Object_IsUniqueReferencedTemporary()` to determine if an object is a unique temporary object on the interpreter's operand stack. This can be used in some cases as a replacement for checking if `Py_REFCNT()` is 1 for Python objects passed as arguments to C API functions. (Contributed by Sam Gross in [gh-133164](#).)

12.2 Limited C API changes

- In the limited C API version 3.14 and newer, `Py_TYPE()` and `Py_REFCNT()` are now implemented as an opaque function call to hide implementation details. (Contributed by Victor Stinner in [gh-120600](#) and [gh-124127](#).)
- Remove the `PySequence_Fast_GET_SIZE`, `PySequence_Fast_GET_ITEM`, and `PySequence_Fast_ITEMS` macros from the limited C API, since they have always been broken in the limited C API. (Contributed by Victor Stinner in [gh-91417](#).)

12.3 被移除的 C API

- Creating immutable types with mutable bases was deprecated in Python 3.12, and now raises a `TypeError`. (Contributed by Nikita Sobolev in [gh-119775](#).)
- Remove `PyDictObject.ma_version_tag` member, which was deprecated in Python 3.12. Use the `PyDict_AddWatcher()` API instead. (Contributed by Sam Gross in [gh-124296](#).)
- Remove the private `_Py_InitializeMain()` function. It was a provisional API added to Python 3.8 by [PEP 587](#). (Contributed by Victor Stinner in [gh-129033](#).)
- Remove the undocumented APIs `Py_C_RECURSION_LIMIT` and `PyThreadState.c_recursion_remaining`. These were added in 3.13 and have been removed without deprecation. Use `Py_EnterRecursiveCall()` to guard against runaway recursion in C code. (Removed by Petr Viktorin in [gh-133079](#), see also [gh-130396](#).)

12.4 已弃用的 C API

- The `Py_HUGE_VAL` macro is now soft deprecated. Use `Py_INFINITY` instead. (Contributed by Sergey B Kirpichev in [gh-120026](#).)
- The `Py_IS_NAN`, `Py_IS_INFINITY`, and `Py_IS_FINITE` macros are now soft deprecated. Use `isnan`, `isinf` and `isfinite` instead, available from `math.h` since C99. (Contributed by Sergey B Kirpichev in [gh-119613](#).)

- Non-tuple sequences are now deprecated as argument for the `(items)` format unit in `PyArg_ParseTuple()` and other argument parsing functions if `items` contains format units which store a borrowed buffer or a borrowed reference. (Contributed by Serhiy Storchaka in [gh-50333](#).)
- The `_PyMonitoring_FireBranchEvent` function is now deprecated and should be replaced with calls to `PyMonitoring_FireBranchLeftEvent()` and `PyMonitoring_FireBranchRightEvent()`.
- The previously undocumented function `PySequence_In()` is now soft deprecated. Use `PySequence_Contains()` instead. (Contributed by Yuki Kobayashi in [gh-127896](#).)

Python 3.15 中待移除的項目

- `PyImport_ImportModuleNoBlock()`: 請改用 `PyImport_ImportModule()`。
- `PyWeakref_GetObject()` 和 `PyWeakref_GET_OBJECT()`: 請改用 `PyWeakref_GetRef()`。可以使用 [pythoncapi-compat](#) 專案來在 Python 3.12 和更早版本取得 `PyWeakref_GetRef()`。
- `Py_UNICODE` 型與 `Py_UNICODE_WIDE` 巨集: 請改用 `wchar_t`。
- `PyUnicode_AsDecodedObject()`: 請改用 `PyCodec_Decode()`。
- `PyUnicode_AsDecodedUnicode()`: 請改用 `PyCodec_Decode()`; 請注意某些編解碼器 (例如 "base64") 可能會回傳非 `str` 的型, 例如 `bytes`。
- `PyUnicode_AsEncodedObject()`: 請改用 `PyCodec_Encode()`。
- `PyUnicode_AsEncodedUnicode()`: 請改用 `PyCodec_Encode()`; 請注意某些編解碼器 (例如 "base64") 可能會回傳非 `bytes` 的型, 例如 `str`。
- Python 初始化函式, 自 Python 3.13 起已被用:
 - `Py_GetPath()`: 請改用 `PyConfig_Get("module_search_paths")` (`sys.path`)。
 - `Py_GetPrefix()`: 請改用 `PyConfig_Get("base_prefix")` (`sys.base_prefix`)。如果需要處理擬環境, 請改用 `PyConfig_Get("prefix")` (`sys.prefix`)。
 - `Py_GetExecPrefix()`: 請改用 `PyConfig_Get("base_exec_prefix")` (`sys.base_exec_prefix`)。如果需要處理擬環境, 請改用 `PyConfig_Get("exec_prefix")` (`sys.exec_prefix`)。
 - `Py_GetProgramFullPath()`: 請改用 `PyConfig_Get("executable")` (`sys.executable`)。
 - `Py_GetProgramName()`: 請改用 `PyConfig_Get("executable")` (`sys.executable`)。
 - `Py_GetPythonHome()`: 請改用 `PyConfig_Get("home")` 或 `PYTHONHOME` 環境變數。

[pythoncapi-compat](#) 專案可以用來在 Python 3.13 和更早版本取得 `PyConfig_Get()`。

- 用於配置 Python 初始化的函式, 自 Python 3.11 起已被用:
 - `PySys_SetArgvEx()`: 請改用 `PyConfig.argv`。
 - `PySys_SetArgv()`: 請改用 `PyConfig.argv`。
 - `Py_SetProgramName()`: 請改用 `PyConfig.program_name`。
 - `Py_SetPythonHome()`: 請改用 `PyConfig.home`。
 - `PySys_ResetWarnOptions()`: 請改清除 `sys.warnoptions` 和 `warnings.filters`。

應改用帶有 `PyConfig` 的 `Py_InitializeFromConfig()` API。

- 全域配置變數:
 - `Py_DebugFlag`: 請改用 `PyConfig.parser_debug` 或 `PyConfig_Get("parser_debug")`。
 - `Py_VerboseFlag`: 請改用 `PyConfig.verbose` 或 `PyConfig_Get("verbose")`。
 - `Py_QuietFlag`: 請改用 `PyConfig.quiet` 或 `PyConfig_Get("quiet")`。
 - `Py_InteractiveFlag`: 請改用 `PyConfig.interactive` 或 `PyConfig_Get("interactive")`。
 - `Py_InspectFlag`: 請改用 `PyConfig.inspect` 或 `PyConfig_Get("inspect")`。

- `Py_OptimizeFlag`: 請改用 `PyConfig.optimization_level` 或 `PyConfig_Get("optimization_level")`。
- `Py_NoSiteFlag`: 請改用 `PyConfig.site_import` 或 `PyConfig_Get("site_import")`。
- `Py_BytesWarningFlag`: 請改用 `PyConfig.bytes_warning` 或 `PyConfig_Get("bytes_warning")`。
- `Py_FrozenFlag`: 請改用 `PyConfig.pathconfig_warnings` 或 `PyConfig_Get("pathconfig_warnings")`。
- `Py_IgnoreEnvironmentFlag`: 請改用 `PyConfig.use_environment` 或 `PyConfig_Get("use_environment")`。
- `Py_DontWriteBytecodeFlag`: 請改用 `PyConfig.write_bytecode` 或 `PyConfig_Get("write_bytecode")`。
- `Py_NoUserSiteDirectory`: 請改用 `PyConfig.user_site_directory` 或 `PyConfig_Get("user_site_directory")`。
- `Py_UnbufferedStdioFlag`: 請改用 `PyConfig.buffered_stdio` 或 `PyConfig_Get("buffered_stdio")`。
- `Py_HashRandomizationFlag`: 請改用 `PyConfig.use_hash_seed` 和 `PyConfig.hash_seed` 或 `PyConfig_Get("hash_seed")`。
- `Py_IsolatedFlag`: 請改用 `PyConfig.isolated` 或 `PyConfig_Get("isolated")`。
- `Py_LegacyWindowsFSEncodingFlag`: 請改用 `PyPreConfig.legacy_windows_fs_encoding` 或 `PyConfig_Get("legacy_windows_fs_encoding")`。
- `Py_LegacyWindowsStdioFlag`: 請改用 `PyConfig.legacy_windows_stdio` 或 `PyConfig_Get("legacy_windows_stdio")`。
- `Py_FileSystemDefaultEncoding`、`Py_HasFileSystemDefaultEncoding`: 請改用 `PyConfig.filesystem_encoding` 或 `PyConfig_Get("filesystem_encoding")`。
- `Py_FileSystemDefaultEncodeErrors`: 請改用 `PyConfig.filesystem_errors` 或 `PyConfig_Get("filesystem_errors")`。
- `Py_UTF8Mode`: 請改用 `PyPreConfig.utf8_mode` 或 `PyConfig_Get("utf8_mode")`。(請參閱 `Py_PreInitialize()`)

應改用帶有 `PyConfig` 的 `Py_InitializeFromConfig()` API 來設定這些選項。或者也可以使用 `PyConfig_Get()` 在執行時取得這些選項。

Python 3.16 中待移除的項目

- `libmpdecimal` 的打包副本 (bundled copy)。

Python 3.18 中待移除的項目

- 以下私有函式已被棄用，計劃在 Python 3.18 中移除：
 - `_PyBytes_Join()`: 請改用 `PyBytes_Join()`。
 - `_PyDict_GetItemStringWithError()`: 請改用 `PyDict_GetItemStringRef()`。
 - `_PyDict_Pop()`: 請改用 `PyDict_Pop()`。
 - `_PyLong_Sign()`: 請改用 `PyLong_GetSign()`。
 - `_PyLong_FromDigits()` 和 `_PyLong_New()`: 請改用 `PyLongWriter_Create()`。
 - `_PyThreadState_UncheckedGet()`: 請改用 `PyThreadState_GetUnchecked()`。
 - `_PyUnicode_AsString()`: 請改用 `PyUnicode_AsUTF8()`。
 - `_PyUnicodeWriter_Init()`: 將 `_PyUnicodeWriter_Init(&writer)` 替換為 `writer = PyUnicodeWriter_Create(0)`。

- `_PyUnicodeWriter_Finish()`: 將 `_PyUnicodeWriter_Finish(&writer)` 替 `PyUnicodeWriter_Finish(writer)`。
- `_PyUnicodeWriter_Dealloc()`: 將 `_PyUnicodeWriter_Dealloc(&writer)` 替 `PyUnicodeWriter_Discard(writer)`。
- `_PyUnicodeWriter_WriteChar()`: 將 `_PyUnicodeWriter_WriteChar(&writer, ch)` 替 `PyUnicodeWriter_WriteChar(writer, ch)`。
- `_PyUnicodeWriter_WriteStr()`: 將 `_PyUnicodeWriter_WriteStr(&writer, str)` 替 `PyUnicodeWriter_WriteStr(writer, str)`。
- `_PyUnicodeWriter_WriteSubstring()`: 將 `_PyUnicodeWriter_WriteSubstring(&writer, str, start, end)` 替 `PyUnicodeWriter_WriteSubstring(writer, str, start, end)`。
- `_PyUnicodeWriter_WriteASCIIString()`: 將 `_PyUnicodeWriter_WriteASCIIString(&writer, str)` 替 `PyUnicodeWriter_WriteASCII(writer, str)`。
- `_PyUnicodeWriter_WriteLatin1String()`: 將 `_PyUnicodeWriter_WriteLatin1String(&writer, str)` 替 `PyUnicodeWriter_WriteUTF8(writer, str)`。
- `_PyUnicodeWriter_Prepare()`: (無替代方案)。
- `_PyUnicodeWriter_PrepareKind()`: (無替代方案)。
- `_Py_HashPointer()`: 請改用 `Py_HashPointer()`。
- `_Py_fopen_obj()`: 請改用 `Py_fopen()`。

可以使用 [pythoncapi-compat project](#) 來取得這些於 Python 3.13 及更早版本的新公開函式。(由 Victor Stinner 在 [gh-128863](#) 貢獻)

未來版本中的待移除項目

下列 API 已被 `DEPRECATED` 將會被移除，不過目前尚未訂定移除日期。

- `Py_TPFLAGS_HAVE_FINALIZE`: 自 Python 3.8 起不再需要
- `PyErr_Fetch()`: 請改用 `PyErr_GetRaisedException()`。
- `PyErr_NormalizeException()`: 請改用 `PyErr_GetRaisedException()`。
- `PyErr_Restore()`: 請改用 `PyErr_SetRaisedException()`。
- `PyModule_GetFilename()`: 請改用 `PyModule_GetFilenameObject()`。
- `PyOS_AfterFork()`: 請改用 `PyOS_AfterFork_Child()`。
- `PySlice_GetIndicesEx()`: 請改用 `PySlice_Unpack()` 和 `PySlice_AdjustIndices()`。
- `PyUnicode_READY()`: 自 Python 3.12 起不再需要
- `PyErr_Display()`: 請改用 `PyErr_DisplayException()`。
- `_PyErr_ChainExceptions()`: 請改用 `_PyErr_ChainExceptions1`。
- `PyBytesObject.ob_shash` 成員: 請改 `PyObject_Hash()`。
- 執行緒局部儲存 (Thread Local Storage, TLS) API:
 - `PyThread_create_key()`: 請改用 `PyThread_tss_alloc()`。
 - `PyThread_delete_key()`: 請改用 `PyThread_tss_free()`。
 - `PyThread_set_key_value()`: 請改用 `PyThread_tss_set()`。
 - `PyThread_get_key_value()`: 請改用 `PyThread_tss_get()`。
 - `PyThread_delete_key_value()`: 請改用 `PyThread_tss_delete()`。
 - `PyThread_ReInitTLS()`: 自 Python 3.7 起不再需要。

13 建置變更

- GNU Autoconf 2.72 is now required to generate `configure`. (Contributed by Erlend Aasland in [gh-115765](#).)
- `wasm32-unknown-emsripten` is now a **PEP 11** tier 3 platform. (Contributed by R. Hood Chatham in [gh-127146](#), [gh-127683](#), and [gh-136931](#).)
- `#pragma-based` linking with `python3*.lib` can now be switched off with `Py_NO_LINK_LIB`. (Contributed by Jean-Christophe Fillion-Robin in [gh-82909](#).)
- CPython now enables a set of recommended compiler options by default for improved security. Use the `--disable-safety` `configure` option to disable them, or the `--enable-slower-safety` option for a larger set of compiler options, albeit with a performance cost.
- The `WITH_FREELISTS` macro and `--without-freelists` `configure` option have been removed.
- The new `configure` option `--with-tail-call-interp` may be used to enable the experimental tail call interpreter. See *A new type of interpreter* for further details.
- To disable the new remote debugging support, use the `--without-remote-debug` `configure` option. This may be useful for security reasons.

13.1 `build-details.json`

Installations of Python now contain a new file, `build-details.json`. This is a static JSON document containing build details for CPython, to allow for introspection without needing to run code. This is helpful for use-cases such as Python launchers, cross-compilation, and so on.

`build-details.json` must be installed in the platform-independent standard library directory. This corresponds to the `'stdlib'` `sysconfig` installation path, which can be found by running `sysconfig.get_path('stdlib')`.

也參考

PEP 739 -- `build-details.json` 1.0 -- a static description file for Python build details

13.2 Discontinuation of PGP signatures

PGP (Pretty Good Privacy) signatures will not be provided for releases of Python 3.14 or future versions. To verify CPython artifacts, users must use [Sigstore verification materials](#). Releases have been signed using [Sigstore](#) since Python 3.11.

This change in release process was specified in [PEP 761](#).

14 移植至 Python 3.14

本節列出了前面描述的更改以及可能需要更改程式碼的其他錯誤修復。

14.1 Python API 的變更

- `functools.partial` is now a method descriptor. Wrap it in `staticmethod()` if you want to preserve the old behavior. (Contributed by Serhiy Storchaka and Dominykas Grigonis in [gh-121027](#).)
- The *garbage collector is now incremental*, which means that the behavior of `gc.collect()` changes slightly:
 - `gc.collect(1)`: Performs an increment of garbage collection, rather than collecting generation 1.
 - Other calls to `gc.collect()` are unchanged.
- The `locale.nl_langinfo()` function now temporarily sets the `LC_CTYPE` locale in some cases. This temporary change affects other threads. (Contributed by Serhiy Storchaka in [gh-69998](#).)
- `types.UnionType` is now an alias for `typing.Union`, causing changes in some behaviors. See *above* for more details. (Contributed by Jelle Zijlstra in [gh-105499](#).)

- The runtime behavior of annotations has changed in various ways; see [above](#) for details. While most code that interacts with annotations should continue to work, some undocumented details may behave differently.

14.2 C API 中的改動

- `Py_Finalize()` now deletes all interned strings. This is backwards incompatible to any C extension that holds onto an interned string after a call to `Py_Finalize()` and is then reused after a call to `Py_Initialize()`. Any issues arising from this behavior will normally result in crashes during the execution of the subsequent call to `Py_Initialize()` from accessing uninitialized memory. To fix, use an address sanitizer to identify any use-after-free coming from an interned string and deallocate it during module shutdown. (Contributed by Eddie Elizondo in [gh-113601](#).)
- The Unicode Exception Objects C API now raises a `TypeError` if its exception argument is not a `UnicodeError` object. (Contributed by B  n  dikt Tran in [gh-127691](#).)
- The interpreter internally avoids some reference count modifications when loading objects onto the operands stack by borrowing references when possible. This can lead to smaller reference count values compared to previous Python versions. C API extensions that checked `Py_REFCNT()` of 1 to determine if a function argument is not referenced by any other code should instead use `PyUnstable_Object_IsUniqueReferencedTemporary()` as a safer replacement.
- Private functions promoted to public C APIs:

```
- _PyBytes_Join(): PyBytes_Join()
- _PyLong_IsNegative(): PyLong_IsNegative()
- _PyLong_IsPositive(): PyLong_IsPositive()
- _PyLong_IsZero(): PyLong_IsZero()
- _PyLong_Sign(): PyLong_GetSign()
- _PyUnicodeWriter_Dealloc(): PyUnicodeWriter_Discard()
- _PyUnicodeWriter_Finish(): PyUnicodeWriter_Finish()
- _PyUnicodeWriter_Init(): use PyUnicodeWriter_Create()
- _PyUnicodeWriter_Prepare(): (no replacement)
- _PyUnicodeWriter_PrepareKind(): (no replacement)
- _PyUnicodeWriter_WriteChar(): PyUnicodeWriter_WriteChar()
- _PyUnicodeWriter_WriteStr(): PyUnicodeWriter_WriteStr()
- _PyUnicodeWriter_WriteSubstring(): PyUnicodeWriter_WriteSubstring()
- _PyUnicode_EQ(): PyUnicode_Equal()
- _PyUnicode_Equal(): PyUnicode_Equal()
- _Py_GetConfig(): PyConfig_Get() and PyConfig_GetInt()
- _Py_HashBytes(): Py_HashBuffer()
- _Py_fopen_obj(): Py_fopen()
- PyMutex_IsLocked(): PyMutex_IsLocked()
```

The [pythoncapi-compat](#) project can be used to get most of these new functions on Python 3.13 and older.

索引

非依字母順序

環境變數

BROWSER, 30
PYTHON_BASIC_REPL, 16
PYTHON_DISABLE_REMOTE_DEBUG, 7
PYTHON_JIT, 16
PYTHONHOME, 46
PYTHONLEGACYWINDOWSFSENCODING, 39
PYTHONSTARTUP, 16

B

BROWSER, 30

C

Common Vulnerabilities and Exposures

CVE 2024-12718, 28
CVE 2025-4138, 28
CVE 2025-4330, 28
CVE 2025-4435, 28
CVE 2025-4517, 25

P

Python Enhancement Proposals

PEP 11, 49
PEP 11#tier-3, 17
PEP 563, 10
PEP 587, 13, 45
PEP 626, 37
PEP 630#type-checking, 44
PEP 649, 4, 9, 10, 29
PEP 659, 15
PEP 667, 36
PEP 684, 5
PEP 688#current-options, 39
PEP 703, 15, 45
PEP 734, 6, 19
PEP 739, 49
PEP 741, 13
PEP 744, 16
PEP 745, 4
PEP 749, 4, 9, 10, 18
PEP 750, 4, 7
PEP 757, 44
PEP 758, 9
PEP 761, 49
PEP 765, 18
PEP 768, 79
PEP 776, 17
PEP 779, 5
PEP 784, 4, 8

PYTHON_BASIC_REPL, 16
PYTHON_DISABLE_REMOTE_DEBUG, 7
PYTHON_JIT, 16
PYTHONHOME, 46

PYTHONLEGACYWINDOWSFSENCODING, 39

PYTHONSTARTUP, 16

R

RFC

RFC 1494, 24
RFC 2104, 18, 22
RFC 2177, 23
RFC 2361, 24
RFC 3362, 24
RFC 3745, 24
RFC 3950, 24
RFC 4047, 24
RFC 4337, 24
RFC 5334, 24
RFC 6713, 24
RFC 7616, 29
RFC 7903, 24
RFC 8081, 23
RFC 9512, 25
RFC 9559, 24
RFC 9562, 30
RFC 9639, 24