
What's New in Python

릴리스 3.14.0a5

A. M. Kuchling

3월 11, 2025

Contents

1	Summary – release highlights	3
2	Incompatible changes	3
3	New features	3
3.1	PEP 649: deferred evaluation of annotations	3
3.2	Improved error messages	4
3.3	PEP 741: Python Configuration C API	5
3.4	A new type of interpreter	5
4	Other language changes	6
5	New modules	7
6	Improved modules	7
6.1	argparse	7
6.2	ast	7
6.3	calendar	7
6.4	concurrent.futures	7
6.5	contextvars	8
6.6	ctypes	8
6.7	datetime	8
6.8	decimal	8
6.9	difflib	8
6.10	dis	8
6.11	errno	9
6.12	fractions	9
6.13	functools	9
6.14	getopt	9
6.15	http	9
6.16	imaplib	9
6.17	inspect	9
6.18	io	9
6.19	json	9
6.20	mimetypes	10
6.21	multiprocessing	10
6.22	operator	11
6.23	os	11
6.24	pathlib	11
6.25	pdb	11
6.26	pickle	12

6.27	platform	12
6.28	pydoc	12
6.29	ssl	12
6.30	symtable	12
6.31	sys	12
6.32	sys.monitoring	12
6.33	threading	13
6.34	tkinter	13
6.35	turtle	13
6.36	unicodedata	13
6.37	unittest	13
6.38	urllib	13
6.39	uuid	13
6.40	zipinfo	13
7	Optimizations	14
7.1	asyncio	14
7.2	base64	14
7.3	io	14
7.4	uuid	14
8	Deprecated	14
8.1	Pending removal in Python 3.15	15
8.2	Pending removal in Python 3.16	16
8.3	Pending removal in future versions	18
9	Removed	20
9.1	argparse	20
9.2	ast	20
9.3	asyncio	20
9.4	collections.abc	22
9.5	email	22
9.6	importlib	22
9.7	itertools	23
9.8	pathlib	23
9.9	pkgutil	23
9.10	pty	23
9.11	sqlite3	23
9.12	typing	23
9.13	urllib	23
9.14	Others	24
10	CPython Bytecode Changes	24
11	Porting to Python 3.14	24
11.1	Changes in the Python API	24
12	Build changes	24
12.1	PEP 761: Discontinuation of PGP signatures	24
13	C API changes	24
13.1	New features	24
13.2	Limited C API changes	26
13.3	Porting to Python 3.14	26
13.4	Deprecated	27
13.5	Removed	31
색인		32

Editor
TBD

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

For full details, see the changelog.

참고

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

- *PEP 649: deferred evaluation of annotations*
- *PEP 741: Python Configuration C API*
- *PEP 761: Discontinuation of PGP signatures*
- *A new type of interpreter*

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.

3 New features

3.1 PEP 649: deferred evaluation of annotations

The annotations on functions, classes, and modules are no longer evaluated eagerly. Instead, annotations are stored in special-purpose `__annotations__` and evaluated only when necessary. 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 usually 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')}
```

(다음 페이지에 계속)

```
>>> 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. 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.

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.

```
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.

3.2 Improved error messages

- When unpacking assignment fails due to 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)
```

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

(다음 페이지에 계속)

```
>>> 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?
```

3.3 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; 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).

(Contributed by Victor Stinner in [gh-107954](#).)

➡ 더 보기

PEP 741.

3.4 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`.

📖 참고

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.

⚠ 주의

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 certain versions of LLVM 19 and 20. At the time of writing, this bug has not yet been fixed in LLVM 19-21. 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.)

4 Other language changes

- 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](#).)
- `\B` in regular expression now matches empty input string. Now it is always the opposite of `\b`. (Contributed by Serhiy Storchaka in [gh-124130](#).)
- iOS and macOS apps can now be configured to redirect `stdout` and `stderr` content to the system log. (Contributed by Russell Keith-Magee in [gh-127592](#).)
- The iOS testbed is now able to stream test output while the test is running. The testbed can also be used to run the test suite of projects other than CPython itself. (Contributed by Russell Keith-Magee in [gh-127592](#).)

5 New modules

- `annotationlib`: For introspecting annotations. See [PEP 749](#) for more details. (Contributed by Jelle Zijlstra in [gh-119180](#).)

6 Improved modules

6.1 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](#).)

6.2 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 B  n  dikt 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 R 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](#).)

6.3 calendar

- By default, today’s date is highlighted in color in `calendar`’s command-line text output. This can be controlled via the `PYTHON_COLORS` environment variable as well as the canonical `NO_COLOR` and `FORCE_COLOR` environment variables. See also [using-on-controlling-color](#). (Contributed by Hugo van Kemenade in [gh-128317](#).)

6.4 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.

(Contributed by Gregory P. Smith in [gh-84559](#).)

- 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](#).)

6.5 contextvars

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

6.6 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](#).)
- 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.dllist()` for listing the shared libraries loaded by the current process. (Contributed by Brian Ward in [gh-119349](#).)

6.7 datetime

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

6.8 decimal

- Add alternative `Decimal` constructor `Decimal.from_number()`. (Contributed by Serhiy Storchaka in [gh-121798](#).)

6.9 difflib

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

6.10 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](#).)

6.11 errno

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

6.12 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](#).)

6.13 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](#).)

6.14 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](#).)

6.15 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](#).)

6.16 imaplib

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

6.17 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](#).)

6.18 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](#).)

6.19 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](#).)

6.20 mimetypes

- 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`

(Contributed by Sahil Prajapati and Hugo van Kemenade in [gh-84852](#).)

- 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)`

(Contributed by Hugo van Kemenade in [gh-89416](#).)

- 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)`

(Contributed by Hugo van Kemenade in [gh-85957](#).)

- 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 9639**: Add FLAC `audio/flac (.flac)`
 - 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)`

(Contributed by Hugo van Kemenade in [gh-129965](#).)

6.21 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.

(Contributed by Gregory P. Smith in [gh-84559](#).)

- `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`.

(Contributed by Roy Hyunjin Han for [gh-103134](#).)

- 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](#).)

6.22 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](#).)

6.23 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](#).)

6.24 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.

(Contributed by Barney Gale in [gh-73991](#).)

- 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](#).)

6.25 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](#).)

6.26 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](#).)

6.27 platform

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

6.28 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](#).)

6.29 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](#).)

6.30 symtable

- Expose the following `symtable.Symbol` methods:

- `is_comp_cell()`
- `is_comp_iter()`
- `is_free_class()`

(Contributed by Bénédict Tran in [gh-120029](#).)

6.31 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'`.

6.32 sys.monitoring

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

6.33 threading

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

6.34 tkinter

- Make tkinter widget methods `after()` and `after_idle()` accept arguments passed by keyword. (Contributed by Zhikang Yan in [gh-126899](#).)

6.35 turtle

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

6.36 unicodedata

- The Unicode database has been updated to Unicode 16.0.0.

6.37 unittest

- `unittest` output is now colored by default. This can be controlled via the `PYTHON_COLORS` environment variable as well as the canonical `NO_COLOR` and `FORCE_COLOR` environment variables. See also `using-on-controlling-color`. (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 string(s).

(Contributed by Serhiy Storchaka in [gh-71339](#).)

6.38 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](#).)

6.39 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](#).)

6.40 zipinfo

- 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 respect `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](#).)

7 Optimizations

7.1 asyncio

- `asyncio` now uses double linked list implementation for native tasks which speeds up execution by 10% on standard pyperformance benchmarks and reduces memory usage. (Contributed by Kumar Aditya in [gh-107803](#).)
- `asyncio` has new utility functions for introspecting and printing the program's call graph: `asyncio.capture_call_graph()` and `asyncio.print_call_graph()`. (Contributed by Yury Selivanov, Pablo Galindo Salgado, and Łukasz Langa in [gh-91048](#).)

7.2 base64

- Improve the performance of `base64.b16decode()` by up to ten times, and reduce the import time of `base64` by up to six times. (Contributed by Bénédict Tran, Chris Markiewicz, and Adam Turner in [gh-118761](#).)

7.3 io

- `io` which provides the built-in `open()` makes less system calls when opening regular files as well as reading whole files. Reading a small operating system cached file in full is up to 15% faster. `pathlib.Path.read_bytes()` has the most optimizations for reading a file's bytes in full. (Contributed by Cody Maloney and Victor Stinner in [gh-120754](#) and [gh-90102](#).)

7.4 uuid

- Improve generation of `UUID` objects via their dedicated functions:
 - `uuid3()` and `uuid5()` are both roughly 40% faster for 16-byte names and 20% faster for 1024-byte names. Performance for longer names remains unchanged.
 - `uuid4()` and `uuid8()` are 30% and 40% faster respectively.(Contributed by Bénédict Tran in [gh-128150](#).)

8 Deprecated

- `argparse`:
 - Passing the undocumented keyword argument `prefix_chars` to `add_argument_group()` is now deprecated. (Contributed by Savannah Ostrowski in [gh-125563](#).)
 - Deprecated the `argparse.FileType` type converter. Anything with resource management should be done downstream after the arguments are parsed. (Contributed by Serhiy Storchaka in [gh-58032](#).)
- `asyncio`:
 - `asyncio.iscoroutinefunction()` is deprecated and will be removed in Python 3.16; use `inspect.iscoroutinefunction()` instead. (Contributed by Jiahao Li and Kumar Aditya in [gh-122875](#).)
 - `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()`

- `load_module()` method: use `exec_module()` instead.
- `locale`:
 - The `getdefaultlocale()` function has been deprecated since Python 3.11. Its removal was originally planned for Python 3.13 ([gh-90817](#)), but has been postponed to Python 3.15. Use `getlocale()`, `setlocale()`, and `getencoding()` instead. (Contributed by Hugo van Kemenade in [gh-111187](#).)
- `pathlib`:
 - `PurePath.is_reserved()` has been deprecated since Python 3.13. Use `os.path.isreserved()` to detect reserved paths on Windows.
- `platform`:
 - `java_ver()` has been deprecated since Python 3.13. This function is only useful for Jython support, has a confusing API, and is largely untested.
- `sysconfig`:
 - The `check_home` argument of `sysconfig.is_python_build()` has been deprecated since Python 3.12.
- `threading`:
 - `RLock()` will take no arguments in Python 3.15. Passing any arguments has been deprecated since Python 3.14, as the Python version does not permit any arguments, but the C version allows any number of positional or keyword arguments, ignoring every argument.
- `types`:
 - `types.CodeType`: Accessing `co_lnotab` was deprecated in [PEP 626](#) since 3.10 and was planned to be removed in 3.12, but it only got a proper `DeprecationWarning` in 3.12. May be removed in 3.15. (Contributed by Nikita Sobolev in [gh-101866](#).)
- `typing`:
 - The undocumented keyword argument syntax for creating `NamedTuple` classes (for example, `Point = NamedTuple("Point", x=int, y=int)`) has been deprecated since Python 3.13. Use the class-based syntax or the functional syntax instead.
 - The `typing.no_type_check_decorator()` decorator function has been deprecated since Python 3.13. After eight years in the `typing` module, it has yet to be supported by any major type checker.
- `wave`:
 - The `getmark()`, `setmark()`, and `getmarkers()` methods of the `Wave_read` and `Wave_write` classes have been deprecated since Python 3.13.
- `zipimport`:
 - `load_module()` has been deprecated since Python 3.10. Use `exec_module()` instead. (Contributed by Jiahao Li in [gh-125746](#).)

8.2 Pending removal in Python 3.16

- The import system:
 - Setting `__loader__` on a module while failing to set `__spec__.loader` is deprecated. In Python 3.16, `__loader__` will cease to be set or taken into consideration by the import system or the standard library.
- `array`:
 - The `'u'` format code (`wchar_t`) has been deprecated in documentation since Python 3.3 and at runtime since Python 3.13. Use the `'w'` format code (`Py_UCS4`) for Unicode characters instead.
- `asyncio`:

- `asyncio.iscoroutinefunction()` is deprecated and will be removed in Python 3.16; use `inspect.iscoroutinefunction()` instead. (Contributed by Jiahao Li and Kumar Aditya in [gh-122875](#).)
- `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()
* asyncio.set_event_loop()
```

Users should use `asyncio.run()` or `asyncio.Runner` with *loop_factory* 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)
```

(Contributed by Kumar Aditya in [gh-127949](#).)

- `builtins`:
 - Bitwise inversion on boolean types, `~True` or `~False` has been deprecated since Python 3.12, as it produces surprising and unintuitive results (`-2` and `-1`). Use `not x` instead for the logical negation of a Boolean. In the rare case that you need the bitwise inversion of the underlying integer, convert to `int` explicitly (`~int(x)`).
- `functools`:
 - Calling the Python implementation of `functools.reduce()` with *function* or *sequence* as keyword arguments has been deprecated since Python 3.14.
- `shutil`:
 - The `ExecError` exception has been deprecated since Python 3.14. It has not been used by any function in `shutil` since Python 3.4, and is now an alias of `RuntimeError`.
- `symtable`:
 - The `Class.get_methods` method has been deprecated since Python 3.14.
- `sys`:
 - The `_enablelegacywindowsfsencoding()` function has been deprecated since Python 3.13. Use the `PYTHONLEGACYWINDOWSFSENCODING` environment variable instead.
- `sysconfig`:
 - The `sysconfig.expand_makefile_vars()` function has been deprecated since Python 3.14. Use the `vars` argument of `sysconfig.get_paths()` instead.
- `tarfile`:
 - The undocumented and unused `TarFile.tarfile` attribute has been deprecated since Python 3.13.

8.3 Pending removal in future versions

The following APIs will be removed in the future, although there is currently no date scheduled for their removal.

- `argparse`:
 - Nesting argument groups and nesting mutually exclusive groups are deprecated.
 - Passing the undocumented keyword argument `prefix_chars` to `add_argument_group()` is now deprecated.
 - The `argparse.FileType` type converter is deprecated.
- `array`'s `'u'` format code ([gh-57281](#))
- `builtins`:
 - `bool(NotImplemented)`.
 - **Generators:** `throw(type, exc, tb)` and `athrow(type, exc, tb)` signature is deprecated: use `throw(exc)` and `athrow(exc)` instead, the single argument signature.
 - Currently Python accepts numeric literals immediately followed by keywords, for example `0in x, 1or x, 0if 1else 2`. It allows confusing and ambiguous expressions like `[0x1for x in y]` (which can be interpreted as `[0x1 for x in y]` or `[0x1f or x in y]`). A syntax warning is raised if the numeric literal is immediately followed by one of keywords `and`, `else`, `for`, `if`, `in`, `is` and `or`. In a future release it will be changed to a syntax error. ([gh-87999](#))
 - Support for `__index__()` and `__int__()` method returning non-int type: these methods will be required to return an instance of a strict subclass of `int`.
 - Support for `__float__()` method returning a strict subclass of `float`: these methods will be required to return an instance of `float`.
 - Support for `__complex__()` method returning a strict subclass of `complex`: these methods will be required to return an instance of `complex`.
 - Delegation of `int()` to `__trunc__()` method.
 - Passing a complex number as the *real* or *imag* argument in the `complex()` constructor is now deprecated; it should only be passed as a single positional argument. (Contributed by Serhiy Storchaka in [gh-109218](#).)
- `calendar`: `calendar.January` and `calendar.February` constants are deprecated and replaced by `calendar.JANUARY` and `calendar.FEBRUARY`. (Contributed by Prince Roshan in [gh-103636](#).)
- `codeobject.co_notab`: use the `codeobject.co_lines()` method instead.
- `datetime`:
 - `utcnow()`: use `datetime.datetime.now(tz=datetime.UTC)`.
 - `utcfromtimestamp()`: use `datetime.datetime.fromtimestamp(timestamp, tz=datetime.UTC)`.
- `gettext`: Plural value must be an integer.
- `importlib`:
 - `cache_from_source()` `debug_override` parameter is deprecated: use the `optimization` parameter instead.
- `importlib.metadata`:
 - `EntryPoints` tuple interface.
 - Implicit `None` on return values.
- `logging`: the `warn()` method has been deprecated since Python 3.3, use `warning()` instead.
- `mailbox`: Use of `StringIO` input and text mode is deprecated, use `BytesIO` and binary mode instead.
- `os`: Calling `os.register_at_fork()` in multi-threaded process.

- `pydoc.ErrorDuringImport`: A tuple value for *exc_info* parameter is deprecated, use an exception instance.
- `re`: More strict rules are now applied for numerical group references and group names in regular expressions. Only sequence of ASCII digits is now accepted as a numerical reference. The group name in bytes patterns and replacement strings can now only contain ASCII letters and digits and underscore. (Contributed by Serhiy Storchaka in [gh-91760](#).)
- `sre_compile`, `sre_constants` and `sre_parse` modules.
- `shutil.rmtree()`'s *onerror* parameter is deprecated in Python 3.12; use the *onexc* parameter instead.
- `ssl` options and protocols:
 - `ssl.SSLContext` without protocol argument is deprecated.
 - `ssl.SSLContext.set_npn_protocols()` and `selected_npn_protocol()` are deprecated: use ALPN instead.
 - `ssl.OP_NO_SSL*` options
 - `ssl.OP_NO_TLS*` options
 - `ssl.PROTOCOL_SSLv3`
 - `ssl.PROTOCOL_TLS`
 - `ssl.PROTOCOL_TLSv1`
 - `ssl.PROTOCOL_TLSv1_1`
 - `ssl.PROTOCOL_TLSv1_2`
 - `ssl.TLSVersion.SSLv3`
 - `ssl.TLSVersion.TLSv1`
 - `ssl.TLSVersion.TLSv1_1`
- `sysconfig.is_python_build()` *check_home* parameter is deprecated and ignored.
- `threading` methods:
 - `threading.Condition.notifyAll()`: use `notify_all()`.
 - `threading.Event.isSet()`: use `is_set()`.
 - `threading.Thread.isDaemon()`, `threading.Thread.setDaemon()`: use `threading.Thread.daemon` attribute.
 - `threading.Thread.getName()`, `threading.Thread.setName()`: use `threading.Thread.name` attribute.
 - `threading.currentThread()`: use `threading.current_thread()`.
 - `threading.activeCount()`: use `threading.active_count()`.
- `typing.Text` ([gh-92332](#)).
- The internal class `typing._UnionGenericAlias` is no longer used to implement `typing.Union`. To preserve compatibility with users using this private class, a compatibility shim will be provided until at least Python 3.17. (Contributed by Jelle Zijlstra in [gh-105499](#).)
- `unittest.IsolatedAsyncioTestCase`: it is deprecated to return a value that is not `None` from a test case.
- `urllib.parse` deprecated functions: `urlparse()` instead
 - `splitattr()`
 - `splithost()`
 - `splitnport()`
 - `splitpasswd()`
 - `splitport()`

- `splitquery()`
 - `splittag()`
 - `splitttype()`
 - `splituser()`
 - `splitvalue()`
 - `to_bytes()`
- `wsgiref: SimpleHandler.stdout.write()` should not do partial writes.
 - `xml.etree.ElementTree: Testing the truth value of an Element is deprecated. In a future release it will always return True. Prefer explicit len(elem) or elem is not None tests instead.`

9 Removed

9.1 argparse

- Remove the *type*, *choices*, and *metavar* parameters of `argparse.BooleanOptionalAction`. They were deprecated since 3.12.
- Calling `add_argument_group()` on an argument group, and calling `add_argument_group()` or `add_mutually_exclusive_group()` on a mutually exclusive group now raise exceptions. 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. They were all deprecated since Python 3.8, and have emitted deprecation warnings since Python 3.12:

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

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

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

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

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

9.3 asyncio

- Remove the following classes and functions. They were all deprecated and emitted deprecation warnings since Python 3.12:

```
- asyncio.get_child_watcher()
- asyncio.set_child_watcher()
- asyncio.AbstractEventLoopPolicy.get_child_watcher()
```

- `asyncio.AbstractEventLoopPolicy.set_child_watcher()`
- `asyncio.AbstractChildWatcher`
- `asyncio.FastChildWatcher`
- `asyncio.MultiLoopChildWatcher`
- `asyncio.PidfdChildWatcher`
- `asyncio.SafeChildWatcher`
- `asyncio.ThreadedChildWatcher`

(Contributed by Kumar Aditya in [gh-120804](#).)

- Removed implicit creation of event loop by `asyncio.get_event_loop()`. It now raises a `RuntimeError` if there is no current event loop. (Contributed by Kumar Aditya in [gh-126353](#).)

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, e.g. 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():
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
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 collections.abc

- Remove `collections.abc.ByteString`. It had previously raised a `DeprecationWarning` since Python 3.12.

9.5 email

- Remove the `isdst` parameter from `email.utils.localtime()`. (Contributed by Hugo van Kemenade in [gh-118798](#).)

9.6 importlib

- Remove deprecated `importlib.abc` classes:
 - `importlib.abc.ResourceReader`
 - `importlib.abc.Traversable`

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

Use `importlib.resources.abc` classes instead:

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

(Contributed by Jason R. Coombs and Hugo van Kemenade in [gh-93963](#).)

9.7 itertools

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

9.8 pathlib

- Remove support for passing additional keyword arguments to `pathlib.Path`. In previous versions, any such arguments are ignored.
- Remove support for passing additional positional arguments to `pathlib.PurePath.relative_to()` and `is_relative_to()`. In previous versions, any such arguments are joined onto *other*.

9.9 pkgutil

- Remove deprecated `pkgutil.get_loader()` and `pkgutil.find_loader()`. These had previously raised a `DeprecationWarning` since Python 3.12. (Contributed by B  n  dikt Tran in [gh-97850](#).)

9.10 pty

- Remove deprecated `pty.master_open()` and `pty.slave_open()`. They had previously raised a `DeprecationWarning` since Python 3.12. Use `pty.openpty()` instead. (Contributed by Nikita Sobolev in [gh-118824](#).)

9.11 sqlite3

- Remove `version` and `version_info` from `sqlite3`. (Contributed by Hugo van Kemenade in [gh-118924](#).)
- Disallow using a sequence of parameters with named placeholders. This had previously raised a `DeprecationWarning` since Python 3.12; it will now raise a `sqlite3.ProgrammingError`. (Contributed by Erlend E. Aasland in [gh-118928](#) and [gh-101693](#).)

9.12 typing

- **Remove `typing.ByteString`.** It had previously raised a `DeprecationWarning` since Python 3.12.
- `typing.TypeAliasType` now supports star unpacking.

9.13 urllib

- Remove deprecated `Quoter` class from `urllib.parse`. It had previously raised a `DeprecationWarning` since Python 3.11. (Contributed by Nikita Sobolev in [gh-118827](#).)
- Remove deprecated `URLopener` and `FancyURLopener` classes from `urllib.request`. They had previously raised a `DeprecationWarning` since Python 3.3.

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

9.14 Others

- Using `NotImplemented` in a boolean context will now raise a `TypeError`. It had previously raised a `DeprecationWarning` since Python 3.9. (Contributed by Jelle Zijlstra in [gh-118767](#).)
- The `int()` built-in no longer delegates to `__trunc__()`. Classes that want to support conversion to integer must implement either `__int__()` or `__index__()`. (Contributed by Mark Dickinson in [gh-119743](#).)

10 CPython Bytecode Changes

- Replaced the opcode `BINARY_SUBSCR` by `BINARY_OP` with oparg `NB_SUBSCR`. (Contributed by Irit Katriel in [gh-100239](#).)

11 Porting to Python 3.14

This section lists previously described changes and other bugfixes that may require changes to your code.

11.1 Changes in the Python API

- `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 `locale.nl_langinfo()` function now sets temporarily the `LC_CTYPE` locale in some cases. This temporary change affects other threads. (Contributed by Serhiy Storchaka in [gh-69998](#).)

12 Build changes

- GNU Autoconf 2.72 is now required to generate `configure`. (Contributed by Erlend Aasland in [gh-115765](#).)
- `#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](#).)

12.1 PEP 761: Discontinuation of PGP signatures

PGP signatures will not be available for CPython 3.14 and onwards. Users verifying artifacts must use [Sigstore verification materials](#) for verifying CPython artifacts. This change in release process is specified in [PEP 761](#).

13 C API changes

13.1 New features

- Add `PyLong_GetSign()` function to get the sign of `int` objects. (Contributed by Sergey B Kirpichev in [gh-116560](#).)
- Add a new `PyUnicodeWriter` API to create a Python `str` object:
 - `PyUnicodeWriter_Create()`
 - `PyUnicodeWriter_DecodeUTF8Stateful()`
 - `PyUnicodeWriter_Discard()`
 - `PyUnicodeWriter_Finish()`
 - `PyUnicodeWriter_Format()`
 - `PyUnicodeWriter_WriteChar()`
 - `PyUnicodeWriter_WriteRepr()`
 - `PyUnicodeWriter_WriteStr()`

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

(Contributed by Victor Stinner in [gh-119182](#).)

- 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_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 from/to Python `int`:

- `PyLong_AsInt32()`
- `PyLong_AsInt64()`
- `PyLong_AsUInt32()`
- `PyLong_AsUInt64()`
- `PyLong_FromInt32()`
- `PyLong_FromInt64()`
- `PyLong_FromUInt32()`
- `PyLong_FromUInt64()`

(Contributed by Victor Stinner in [gh-120389](#).)

- Add `PyBytes_Join(sep, iterable)` function, similar to `sep.join(iterable)` in Python. (Contributed by Victor Stinner in [gh-121645](#).)
- Add `Py_HashBuffer()` to compute and return the hash value of a buffer. (Contributed by Antoine Pitrou and Victor Stinner in [gh-122854](#).)
- Add functions to get and set the current runtime Python configuration (**PEP 741**):

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

(Contributed by Victor Stinner in [gh-107954](#).)

- Add functions to configure the Python initialization (**PEP 741**):

- `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()

(Contributed by Victor Stinner in [gh-107954](#).)

- 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().

(Contributed by Sergey B Kirpichev and Victor Stinner in [gh-102471](#).)

- Add `PyType_GetBaseByToken()` and `Py_tp_token` slot for easier superclass identification, which attempts to resolve the [type checking issue](#) mentioned in **PEP 630** ([gh-124153](#)).
- Add `PyUnicode_Equal()` function to the limited C API: test if two strings are equal. (Contributed by Victor Stinner in [gh-124502](#).)
- Add `PyType_Freeze()` function to make a type immutable. (Contributed by Victor Stinner in [gh-121654](#).)
- Add `PyUnstable_Object_EnableDeferredRefcount()` for enabling deferred reference counting, as outlined in **PEP 703**.
- Add `PyMonitoring_FireBranchLeftEvent()` and `PyMonitoring_FireBranchRightEvent()` for generating `BRANCH_LEFT` and `BRANCH_RIGHT` events, respectively.
- Add `Py_fopen()` function to open a file. Similar to the `fopen()` function, but the `path` parameter is a Python object and an exception is set on error. Add also `Py_fclose()` function to close a file. (Contributed by Victor Stinner in [gh-127350](#).)
- Add macros `Py_PACK_VERSION()` and `Py_PACK_FULL_VERSION()` for bit-packing Python version numbers. (Contributed by Petr Viktorin in [gh-128629](#).)
- Add `PyUnstable_IsImmortal()` for determining whether an object is immortal, for debugging purposes.
- 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 support for a new `p` format unit in `Py_BuildValue()` that allows to take a C integer and produce a Python `bool` object. (Contributed by Pablo Galindo in [bpo-45325](#).)

13.2 Limited C API changes

- In the limited C API 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 `PySequence_Fast()` from the limited C API, since this function has to be used with `PySequence_Fast_GET_ITEM` which never worked in the limited C API. (Contributed by Victor Stinner in [gh-91417](#).)

13.3 Porting to Python 3.14

- `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               in [gh-127691](#).)
- 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()`.

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

13.4 Deprecated

- The `Py_HUGE_VAL` macro is soft deprecated, use `Py_INFINITY` instead. (Contributed by Sergey B Kirpichev in [gh-120026](#).)
- Macros `Py_IS_NAN`, `Py_IS_INFINITY` and `Py_IS_FINITE` are soft deprecated, use instead `isnan`, `isinf` and `isfinite` available from `math.h` since C99. (Contributed by Sergey B Kirpichev in [gh-119613](#).)
- The previously undocumented function `PySequence_In()` is soft deprecated. Use `PySequence_Contains()` instead. (Contributed by Yuki Kobayashi in [gh-127896](#).)
- The `PyMonitoring_FireBranchEvent` function is deprecated and should be replaced with calls to `PyMonitoring_FireBranchLeftEvent()` and `PyMonitoring_FireBranchRightEvent()`.
- The following private functions are deprecated and planned for removal in Python 3.18:
 - `_PyBytes_Join()`: **use** `PyBytes_Join()`.
 - `_PyDict_GetItemStringWithError()`: **use** `PyDict_GetItemStringRef()`.
 - `_PyDict_Pop()`: **use** `PyDict_Pop()`.
 - `_PyLong_Sign()`: **use** `PyLong_GetSign()`.
 - `_PyLong_FromDigits()` **and** `_PyLong_New()`: **use** `PyLongWriter_Create()`.
 - `_PyThreadState_UncheckedGet()`: **use** `PyThreadState_GetUnchecked()`.

- `_PyUnicode_AsString()`: use `PyUnicode_AsUTF8()`.
- `_PyUnicodeWriter_Init()`: replace `_PyUnicodeWriter_Init(&writer)` with `writer = PyUnicodeWriter_Create(0)`.
- `_PyUnicodeWriter_Finish()`: replace `_PyUnicodeWriter_Finish(&writer)` with `PyUnicodeWriter_Finish(writer)`.
- `_PyUnicodeWriter_Dealloc()`: replace `_PyUnicodeWriter_Dealloc(&writer)` with `PyUnicodeWriter_Discard(writer)`.
- `_PyUnicodeWriter_WriteChar()`: replace `_PyUnicodeWriter_WriteChar(&writer, ch)` with `PyUnicodeWriter_WriteChar(writer, ch)`.
- `_PyUnicodeWriter_WriteStr()`: replace `_PyUnicodeWriter_WriteStr(&writer, str)` with `PyUnicodeWriter_WriteStr(writer, str)`.
- `_PyUnicodeWriter_WriteSubstring()`: replace `_PyUnicodeWriter_WriteSubstring(&writer, str, start, end)` with `PyUnicodeWriter_WriteSubstring(writer, str, start, end)`.
- `_PyUnicodeWriter_WriteASCIIString()`: replace `_PyUnicodeWriter_WriteASCIIString(&writer, str)` with `PyUnicodeWriter_WriteUTF8(writer, str)`.
- `_PyUnicodeWriter_WriteLatin1String()`: replace `_PyUnicodeWriter_WriteLatin1String(&writer, str)` with `PyUnicodeWriter_WriteUTF8(writer, str)`.
- `_Py_HashPointer()`: use `Py_HashPointer()`.
- `_Py_fopen_obj()`: use `Py_fopen()`.

The [pythoncapi-compat](#) project can be used to get these new public functions on Python 3.13 and older. (Contributed by Victor Stinner in [gh-128863](#).)

Pending removal in Python 3.15

- The bundled copy of `libmpdecimal`.
- The `PyImport_ImportModuleNoBlock()`: Use `PyImport_ImportModule()` instead.
- `PyWeakref_GetObject()` and `PyWeakref_GET_OBJECT()`: Use `PyWeakref_GetRef()` instead. The [pythoncapi-compat](#) project can be used to get `PyWeakref_GetRef()` on Python 3.12 and older.
- `Py_UNICODE` type and the `Py_UNICODE_WIDE` macro: Use `wchar_t` instead.
- Python initialization functions, deprecated in Python 3.13:
 - `Py_GetPath()`: Use `PyConfig_Get("module_search_paths")` (`sys.path`) instead.
 - `Py_GetPrefix()`: Use `PyConfig_Get("base_prefix")` (`sys.base_prefix`) instead. Use `PyConfig_Get("prefix")` (`sys.prefix`) if virtual environments need to be handled.
 - `Py_GetExecPrefix()`: Use `PyConfig_Get("base_exec_prefix")` (`sys.base_exec_prefix`) instead. Use `PyConfig_Get("exec_prefix")` (`sys.exec_prefix`) if virtual environments need to be handled.
 - `Py_GetProgramFullPath()`: Use `PyConfig_Get("executable")` (`sys.executable`) instead.
 - `Py_GetProgramName()`: Use `PyConfig_Get("executable")` (`sys.executable`) instead.
 - `Py_GetPythonHome()`: Use `PyConfig_Get("home")` or the `PYTHONHOME` environment variable instead.

The [pythoncapi-compat](#) project can be used to get `PyConfig_Get()` on Python 3.13 and older.

- Functions to configure Python's initialization, deprecated in Python 3.11:
 - `PySys_SetArgvEx()`: Set `PyConfig.argv` instead.
 - `PySys_SetArgv()`: Set `PyConfig.argv` instead.

- `Py_SetProgramName()`: **Set** `PyConfig.program_name` **instead**.
- `Py_SetPythonHome()`: **Set** `PyConfig.home` **instead**.
- `PySys_ResetWarnOptions()`: **Clear** `sys.warnoptions` **and** `warnings.filters` **instead**.

The `Py_InitializeFromConfig()` API should be used with `PyConfig` **instead**.

- **Global configuration variables:**

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

The `Py_InitializeFromConfig()` API should be used with `PyConfig` to set these options. `PyConfig_Get()` can be used to get these options at runtime.

Pending removal in Python 3.18

- **Deprecated private functions ([gh-128863](#)):**

- `_PyBytes_Join()`: use `PyBytes_Join()`.
- `_PyDict_GetItemStringWithError()`: use `PyDict_GetItemStringRef()`.
- `_PyDict_Pop()`: `PyDict_Pop()`.
- `_PyLong_Sign()`: use `PyLong_GetSign()`.
- `_PyLong_FromDigits()` and `_PyLong_New()`: use `PyLongWriter_Create()`.
- `_PyThreadState_UncheckedGet()`: use `PyThreadState_GetUnchecked()`.
- `_PyUnicode_AsString()`: use `PyUnicode_AsUTF8()`.
- `_PyUnicodeWriter_Init()`: replace `_PyUnicodeWriter_Init(&writer)` with `writer = PyUnicodeWriter_Create(0)`.
- `_PyUnicodeWriter_Finish()`: replace `_PyUnicodeWriter_Finish(&writer)` with `PyUnicodeWriter_Finish(writer)`.
- `_PyUnicodeWriter_Dealloc()`: replace `_PyUnicodeWriter_Dealloc(&writer)` with `PyUnicodeWriter_Discard(writer)`.
- `_PyUnicodeWriter_WriteChar()`: replace `_PyUnicodeWriter_WriteChar(&writer, ch)` with `PyUnicodeWriter_WriteChar(writer, ch)`.
- `_PyUnicodeWriter_WriteStr()`: replace `_PyUnicodeWriter_WriteStr(&writer, str)` with `PyUnicodeWriter_WriteStr(writer, str)`.
- `_PyUnicodeWriter_WriteSubstring()`: replace `_PyUnicodeWriter_WriteSubstring(&writer, str, start, end)` with `PyUnicodeWriter_WriteSubstring(writer, str, start, end)`.
- `_PyUnicodeWriter_WriteASCIIString()`: replace `_PyUnicodeWriter_WriteASCIIString(&writer, str)` with `PyUnicodeWriter_WriteUTF8(writer, str)`.
- `_PyUnicodeWriter_WriteLatin1String()`: replace `_PyUnicodeWriter_WriteLatin1String(&writer, str)` with `PyUnicodeWriter_WriteUTF8(writer, str)`.
- `_PyUnicodeWriter_Prepare()`: (no replacement).
- `_PyUnicodeWriter_PrepareKind()`: (no replacement).
- `_Py_HashPointer()`: use `Py_HashPointer()`.
- `_Py_fopen_obj()`: use `Py_fopen()`.

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

Pending removal in future versions

The following APIs are deprecated and will be removed, although there is currently no date scheduled for their removal.

- `Py_TPFLAGS_HAVE_FINALIZE`: Unneeded since Python 3.8.
- `PyErr_Fetch()`: Use `PyErr_GetRaisedException()` instead.
- `PyErr_NormalizeException()`: Use `PyErr_GetRaisedException()` instead.
- `PyErr_Restore()`: Use `PyErr_SetRaisedException()` instead.
- `PyModule_GetFilename()`: Use `PyModule_GetFilenameObject()` instead.
- `PyOS_AfterFork()`: Use `PyOS_AfterFork_Child()` instead.
- `PySlice_GetIndicesEx()`: Use `PySlice_Unpack()` and `PySlice_AdjustIndices()` instead.
- `PyUnicode_AsDecodedObject()`: Use `PyCodec_Decode()` instead.

- `PyUnicode_AsDecodedUnicode()`: Use `PyCodec_Decode()` instead.
- `PyUnicode_AsEncodedObject()`: Use `PyCodec_Encode()` instead.
- `PyUnicode_AsEncodedUnicode()`: Use `PyCodec_Encode()` instead.
- `PyUnicode_READY()`: Unneeded since Python 3.12
- `PyErr_Display()`: Use `PyErr_DisplayException()` instead.
- `_PyErr_ChainExceptions()`: Use `_PyErr_ChainExceptions1()` instead.
- `PyBytesObject.ob_shash` member: call `PyObject_Hash()` instead.
- Thread Local Storage (TLS) API:
 - `PyThread_create_key()`: Use `PyThread_tss_alloc()` instead.
 - `PyThread_delete_key()`: Use `PyThread_tss_free()` instead.
 - `PyThread_set_key_value()`: Use `PyThread_tss_set()` instead.
 - `PyThread_get_key_value()`: Use `PyThread_tss_get()` instead.
 - `PyThread_delete_key_value()`: Use `PyThread_tss_delete()` instead.
 - `PyThread_ReInitTLS()`: Unneeded since Python 3.7.

13.5 Removed

- Creating `immutable` types with mutable bases was deprecated since 3.12 and now raises a `TypeError`.
- Remove `PyDictObject.ma_version_tag` member which was deprecated since 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](#).)

색인

P

Python **항상 제안**

PEP 563, [4](#)
PEP 587, [5](#), [31](#)
PEP 626, [16](#)
PEP 630, [26](#)
PEP 649, [3](#)
PEP 703, [26](#)
PEP 734, [7](#)
PEP 741, [5](#), [25](#)
PEP 749, [3](#), [7](#)
PEP 757, [26](#)
PEP 761, [24](#)

PYTHON_COLORS, [7](#), [13](#)

PYTHONHOME, [28](#)

PYTHONLEGACYWINDOWSFSENCODING, [17](#)

R

RFC

RFC 1494, [10](#)
RFC 2177, [9](#)
RFC 2361, [10](#)
RFC 3362, [10](#)
RFC 3745, [10](#)
RFC 3950, [10](#)
RFC 4047, [10](#)
RFC 4337, [10](#)
RFC 5334, [10](#)
RFC 7616, [13](#)
RFC 7903, [10](#)
RFC 8081, [10](#)
RFC 9559, [10](#)
RFC 9562, [13](#)
RFC 9639, [10](#)

Y

환경 변수

PYTHON_COLORS, [7](#), [13](#)

PYTHONHOME, [28](#)

PYTHONLEGACYWINDOWSFSENCODING, [17](#)