What's New in Python

發🄵 *3.14.0a1*

A. M. Kuchling

11 月 11, 2024

Python Software Foundation
Email: docs@python.org

Contents

1 Summary -- release highlights **2**

2 New features **2**
2.1 PEP 649: deferred evaluation of annotations . . . 2
2.2 Improved error messages . . . 3

3 Other language changes **4**

4 New modules **4**

5 Improved modules **4**
5.1 argparse . . . 4
5.2 ast . . . 4
5.3 concurrent.futures . . . 5
5.4 ctypes . . . 5
5.5 decimal . . . 5
5.6 datetime . . . 5
5.7 dis . . . 5
5.8 fractions . . . 5
5.9 functools . . . 5
5.10 http . . . 5
5.11 inspect . . . 6
5.12 json . . . 6
5.13 operator . . . 6
5.14 os . . . 6
5.15 pathlib . . . 6
5.16 pdb . . . 6
5.17 pickle . . . 7
5.18 pydoc . . . 7
5.19 symtable . . . 7
5.20 sys . . . 7
5.21 unicodedata . . . 7
5.22 unittest . . . 7

6 Optimizations **7**
6.1 asyncio . . . 7
6.2 io . . . 7

7 Deprecated 7
7.1 Pending removal in Python 3.15 . . . 8
7.2 Pending removal in Python 3.16 . . . 9
7.3 Pending removal in future versions . . . 9

8 Removed 12
8.1 argparse . . . 12
8.2 ast . . . 12
8.3 asyncio . . . 12
8.4 collections.abc . . . 13
8.5 email . . . 13
8.6 importlib . . . 13
8.7 itertools . . . 13
8.8 pathlib . . . 13
8.9 pkgutil . . . 13
8.10 pty . . . 13
8.11 sqlite3 . . . 14
8.12 typing . . . 14
8.13 urllib . . . 14
8.14 Others . . . 14

9 Porting to Python 3.14 14
9.1 Changes in the Python API . . . 14

10 Build changes 14

11 C API changes 14
11.1 New features . . . 14
11.2 Porting to Python 3.14 . . . 16
11.3 Deprecated . . . 16
11.4 Removed . . . 17

索引 18

Editor
TBD

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

For full details, see the changelog.

> **備🄵**
>
> 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

2 New features

2.1 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 annotate functions 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')}
>>> 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.

2.2 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)
```

3 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.)
- `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.)

4 New modules

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

5 Improved modules

5.1 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)

5.2 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 Bénédikt 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.)

5.3 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.)

5.4 ctypes

- The layout of bit fields in `Structure` and `Union` now matches platform defaults (GCC/Clang or MVSC) 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.)

5.5 decimal

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

5.6 datetime

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

5.7 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 Bénédikt Tran in gh-123165.)

5.8 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.)

5.9 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.)

5.10 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.)

5.11 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.)

5.12 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.)

5.13 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.)

5.14 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.)

5.15 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.scandir()` to scan a directory and return an iterator of `os.DirEntry` objects. This is exactly equivalent to calling `os.scandir()` on a path object.

  (Contributed by Barney Gale in gh-125413.)

5.16 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.)

5.17 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.)

5.18 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.)

5.19 symtable

- Expose the following `symtable.Symbol` methods:
  - `is_comp_cell()`
  - `is_comp_iter()`
  - `is_free_class()`

  (Contributed by Bénédikt Tran in gh-120029.)

5.20 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.

5.21 unicodedata

- The Unicode database has been updated to Unicode 16.0.0.

5.22 unittest

- unittest discovery supports namespace package as start directory again. It was removed in Python 3.11. (Contributed by Jacob Walls in gh-80958.)

6 Optimizations

6.1 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.)

6.2 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 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.)
- `builtins`: 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.)
- `multiprocessing` and `concurrent.futures`: The default start method (see multiprocessing-start-methods) changed away from *fork* to *forkserver* on platforms where it was not already *spawn* (Windows & macOS). If you require the threading incompatible *fork* start method you must explicitly specify it when using `multiprocessing` or `concurrent.futures` APIs. (Contributed by Gregory P. Smith in gh-84559.)
- `os`: Soft deprecate `os.popen()` and `os.spawn*` functions. They should no longer be used to write new code. The `subprocess` module is recommended instead. (Contributed by Victor Stinner in gh-120743.)
- `symtable`: Deprecate `symtable.Class.get_methods()` due to the lack of interest. (Contributed by Bénédikt Tran in gh-119698.)

7.1 Pending removal in Python 3.15

- The import system:
  - Setting `__cached__` on a module while failing to set `__spec__.cached` is deprecated. In Python 3.15, `__cached__` will cease to be set or take into consideration by the import system or standard library. (gh-97879)
  - Setting `__package__` on a module while failing to set `__spec__.parent` is deprecated. In Python 3.15, `__package__` will cease to be set or take into consideration by the import system or standard library. (gh-97879)
- `ctypes`:
  - The undocumented `ctypes.SetPointerType()` function has been deprecated since Python 3.13.
- `http.server`:
  - The obsolete and rarely used `CGIHTTPRequestHandler` has been deprecated since Python 3.13. No direct replacement exists. *Anything* is better than CGI to interface a web server with a request handler.
  - The `--cgi` flag to the **`python -m http.server`** command-line interface has been deprecated since Python 3.13.
- `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.
- `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.

7.2 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.)

- `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)`).

- `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.

- `tarfile`:
  - The undocumented and unused `TarFile.tarfile` attribute has been deprecated since Python 3.13.

7.3 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_lnotab`: 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`:
  - `load_module()` method: use `exec_module()` instead.
  - `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).
- `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()`
  - `splittype()`
  - `splituser()`

  - `splitvalue()`
  - `to_bytes()`

- `urllib.request`: `URLopener` and `FancyURLopener` style of invoking requests is deprecated. Use newer `urlopen()` functions and methods.
- `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.
- `zipimport.zipimporter.load_module()` is deprecated: use `exec_module()` instead.

8 Removed

8.1 argparse

- Remove the *type*, *choices*, and *metavar* parameters of `argparse.BooleanOptionalAction`. They were deprecated since 3.12.

8.2 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.)

8.3 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.)

8.4 collections.abc

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

8.5 email

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

8.6 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.)

8.7 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.)

8.8 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*.

8.9 pkgutil

- Remove deprecated `pkgutil.get_loader()` and `pkgutil.find_loader()`. These had previously raised a `DeprecationWarning` since Python 3.12. (Contributed by Bénédikt Tran in gh-97850.)

8.10 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.)

8.11 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.)

8.12 typing

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

8.13 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.)

8.14 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.)

9 Porting to Python 3.14

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

9.1 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.)

10 Build changes

11 C API changes

11.1 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.)
- `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.)
- 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 `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.)

11.2 Porting to Python 3.14

- 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.)

11.3 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.)

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.
- `Py_UNICODE` type and the `Py_UNICODE_WIDE` macro: Use `wchar_t` instead.
- Python initialization functions:
  - `PySys_ResetWarnOptions()`: Clear `sys.warnoptions` and `warnings.filters` instead.
  - `Py_GetExecPrefix()`: Get `sys.base_exec_prefix` and `sys.exec_prefix` instead.
  - `Py_GetPath()`: Get `sys.path` instead.
  - `Py_GetPrefix()`: Get `sys.base_prefix` and `sys.prefix` instead.
  - `Py_GetProgramFullPath()`: Get `sys.executable` instead.
  - `Py_GetProgramName()`: Get `sys.executable` instead.
  - `Py_GetPythonHome()`: Get `PyConfig.home` or the `PYTHONHOME` environment variable instead.

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.
- `PyDictObject.ma_version_tag` member.
- 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.

11.4 Removed

- Creating `immutable types` with mutable bases was deprecated since 3.12 and now raises a `TypeError`.

索引

非依字母順序

環境變數
    PYTHONHOME, 16
    PYTHONLEGACYWINDOWSFSENCODING, 9

P

Python Enhancement Proposals
    PEP 563, 3
    PEP 626, 9
    PEP 630, 16
    PEP 649, 2
    PEP 734, 5
    PEP 741, 15
    PEP 749, 2, 4
PYTHONHOME, 16
PYTHONLEGACYWINDOWSFSENCODING, 9