What's New in Python

Release 3.13.0b3

A. M. Kuchling

July 18, 2024

Python Software Foundation
Email: docs@python.org

Contents

Editor
Thomas Wouters

This article explains the new features in Python 3.13, compared to 3.12.

For full details, see the changelog.

See also:

PEP 719 – Python 3.13 Release Schedule

Note: Prerelease users should be aware that this document is currently in draft form. It will be updated substantially as Python 3.13 moves towards release, so it's worth checking back even after reading earlier versions.

---

1 Summary – Release Highlights

Python 3.13 beta is the pre-release of the next version of the Python programming language, with a mix of changes to the language, the implementation and the standard library. The biggest changes to the implementation include a new interactive interpreter, and experimental support for dropping the Global Interpreter Lock (**PEP 703**) and a Just-In-Time compiler (**PEP 744**). The library changes contain removal of deprecated APIs and modules, as well as the usual improvements in user-friendliness and correctness.

Interpreter improvements:

- A greatly improved *interactive interpreter* and *improved error messages*.
- Color support in the new *interactive interpreter*, as well as in *tracebacks* and *doctest* output. This can be disabled through the `PYTHON_COLORS` and `NO_COLOR` environment variables.
- **PEP 744**: A basic *JIT compiler* was added. It is currently disabled by default (though we may turn it on later). Performance improvements are modest – we expect to be improving this over the next few releases.
- **PEP 667**: The `locals()` builtin now has *defined semantics* when mutating the returned mapping. Python debuggers and similar tools may now more reliably update local variables in optimized scopes even during concurrent code execution.

New typing features:

- **PEP 696**: Type parameters (`typing.TypeVar`, `typing.ParamSpec`, and `typing.TypeVarTuple`) now support defaults.
- **PEP 702**: Support for marking deprecations in the type system using the new `warnings.deprecated()` decorator.
- **PEP 742**: `typing.TypeIs` was added, providing more intuitive type narrowing behavior.
- **PEP 705**: `typing.ReadOnly` was added, to mark an item of a `typing.TypedDict` as read-only for type checkers.

Free-threading:

- **PEP 703**: CPython 3.13 has experimental support for running with the global interpreter lock disabled when built with `--disable-gil`. See *Free-threaded CPython* for more details.

Platform support:

- **PEP 730**: Apple's iOS is now an officially supported platform. Official Android support (**PEP 738**) is in the works as well.

Removed modules:

- *PEP 594*: The remaining 19 "dead batteries" have been removed from the standard library: `aifc`, `audioop`, `cgi`, `cgitb`, `chunk`, `crypt`, `imghdr`, `mailcap`, `msilib`, `nis`, `nntplib`, `ossaudiodev`, `pipes`, `sndhdr`, `spwd`, `sunau`, `telnetlib`, `uu` and `xdrlib`.
- Also removed were the `tkinter.tix` and `lib2to3` modules, and the `2to3` program.

Release schedule changes:

- **PEP 602** ("Annual Release Cycle for Python") has been updated:
  - Python 3.9 - 3.12 have one and a half years of full support, followed by three and a half years of security fixes.
  - Python 3.13 and later have two years of full support, followed by three years of security fixes.

2 New Features

2.1 A Better Interactive Interpreter

On Unix-like systems like Linux or macOS as well as Windows, Python now uses a new interactive shell. When the user starts the REPL from an interactive terminal the interactive shell now supports the following new features:

- Colorized prompts.
- Multiline editing with history preservation.
- Interactive help browsing using `F1` with a separate command history.
- History browsing using `F2` that skips output as well as the >>> and ... prompts.
- "Paste mode" with `F3` that makes pasting larger blocks of code easier (press `F3` again to return to the regular prompt).
- The ability to issue REPL-specific commands like `help`, `exit`, and `quit` without the need to use call parentheses after the command name.

If the new interactive shell is not desired, it can be disabled via the `PYTHON_BASIC_REPL` environment variable.

The new shell requires `curses` on Unix-like systems.

For more on interactive mode, see tut-interac.

(Contributed by Pablo Galindo Salgado, Łukasz Langa, and Lysandros Nikolaou in gh-111201 based on code from the PyPy project. Windows support contributed by Dino Viehland and Anthony Shaw.)

2.2 Improved Error Messages

- The interpreter now colorizes error messages when displaying tracebacks by default. This feature can be controlled via the new `PYTHON_COLORS` environment variable as well as the canonical `NO_COLOR` and `FORCE_COLOR` environment variables. See also using-on-controlling-color. (Contributed by Pablo Galindo Salgado in gh-112730.)
- A common mistake is to write a script with the same name as a standard library module. When this results in errors, we now display a more helpful error message:

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

  Similarly, if a script has the same name as a third-party module it attempts to import, and this results in errors, we also display a more helpful error message:

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

  (Contributed by Shantanu Jain in gh-95754.)

- When an incorrect keyword argument is passed to a function, the error message now potentially suggests the correct keyword argument. (Contributed by Pablo Galindo Salgado and Shantanu Jain in gh-107944.)

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

- Classes have a new `__static_attributes__` attribute, populated by the compiler, with a tuple of names of attributes of this class which are accessed through `self.X` from any function in its body. (Contributed by Irit Katriel in gh-115775.)

2.3 Defined mutation semantics for `locals()`

Historically, the expected result of mutating the return value of `locals()` has been left to individual Python implementations to define.

Through **PEP 667**, Python 3.13 standardises the historical behaviour of CPython for most code execution scopes, but changes optimized scopes (functions, generators, coroutines, comprehensions, and generator expressions) to explicitly return independent snapshots of the currently assigned local variables, including locally referenced nonlocal variables captured in closures.

This change to the semantics of `locals()` in optimized scopes also affects the default behaviour of code execution functions that implicitly target `locals()` if no explicit namespace is provided (such as `exec()` and `eval()`). In previous versions, whether or not changes could be accessed by calling `locals()` after calling the code execution function was implementation dependent. In CPython specifically, such code would typically appear to work as desired, but could sometimes fail in optimized scopes based on other code (including debuggers and code execution tracing tools) potentially resetting the shared snapshot in that scope. Now, the code will always run against an independent snapshot of the local variables in optimized scopes, and hence the changes will never be visible in subsequent calls to `locals()`. To access the changes made in these cases, an explicit namespace reference must now be passed to the relevant function. Alternatively, it may make sense to update affected code to use a higher level code execution API that returns the resulting code execution namespace (e.g. `runpy.run_path()` when executing Python files from disk).

To ensure debuggers and similar tools can reliably update local variables in scopes affected by this change, `FrameType.f_locals` now returns a write-through proxy to the frame's local and locally referenced nonlocal variables in these scopes, rather than returning an inconsistently updated shared `dict` instance with undefined runtime semantics.

See **PEP 667** for more details, including related C API changes and deprecations. Porting notes are also provided below for the affected *Python APIs* and *C APIs*.

(PEP and implementation contributed by Mark Shannon and Tian Gao in gh-74929. Documentation updates provided by Guido van Rossum and Alyssa Coghlan.)

2.4 Incremental Garbage Collection

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

2.5 Support For Mobile Platforms

- iOS is now a **PEP 11** supported platform. `arm64-apple-ios` (iPhone and iPad devices released after 2013) and `arm64-apple-ios-simulator` (Xcode iOS simulator running on Apple Silicon hardware) are now tier 3 platforms.

  `x86_64-apple-ios-simulator` (Xcode iOS simulator running on older x86_64 hardware) is not a tier 3 supported platform, but will be supported on a best-effort basis.

  See **PEP 730**: for more details.

  (PEP written and implementation contributed by Russell Keith-Magee in gh-114099.)

3 Experimental JIT Compiler

When CPython is configured using the `--enable-experimental-jit` option, a just-in-time compiler is added which may speed up some Python programs.

The internal architecture is roughly as follows.

- We start with specialized *Tier 1 bytecode*. See What's new in 3.11 for details.
- When the Tier 1 bytecode gets hot enough, it gets translated to a new, purely internal *Tier 2 IR*, a.k.a. micro-ops ("uops").
- The Tier 2 IR uses the same stack-based VM as Tier 1, but the instruction format is better suited to translation to machine code.
- We have several optimization passes for Tier 2 IR, which are applied before it is interpreted or translated to machine code.
- There is a Tier 2 interpreter, but it is mostly intended for debugging the earlier stages of the optimization pipeline. The Tier 2 interpreter can be enabled by configuring Python with `--enable-experimental-jit=interpreter`.
- When the JIT is enabled, the optimized Tier 2 IR is translated to machine code, which is then executed.
- The machine code translation process uses a technique called *copy-and-patch*. It has no runtime dependencies, but there is a new build-time dependency on LLVM.

The `--enable-experimental-jit` flag has the following optional values:

- `no` (default) – Disable the entire Tier 2 and JIT pipeline.
- `yes` (default if the flag is present without optional value) – Enable the JIT. To disable the JIT at runtime, pass the environment variable `PYTHON_JIT=0`.
- `yes-off` – Build the JIT but disable it by default. To enable the JIT at runtime, pass the environment variable `PYTHON_JIT=1`.
- `interpreter` – Enable the Tier 2 interpreter but disable the JIT. The interpreter can be disabled by running with `PYTHON_JIT=0`.

(On Windows, use `PCbuild/build.bat --experimental-jit` to enable the JIT or `--experimental-jit-interpreter` to enable the Tier 2 interpreter.)

See **PEP 744** for more details.

(JIT by Brandt Bucher, inspired by a paper by Haoran Xu and Fredrik Kjolstad. Tier 2 IR by Mark Shannon and Guido van Rossum. Tier 2 optimizer by Ken Jin.)

4 Free-threaded CPython

CPython will run with the global interpreter lock (GIL) disabled when configured using the `--disable-gil` option at build time. This is an experimental feature and therefore isn't used by default. Users need to either compile their own interpreter, or install one of the experimental builds that are marked as *free-threaded*. See **PEP 703** "Making the Global Interpreter Lock Optional in CPython" for more detail.

Free-threaded execution allows for full utilization of the available processing power by running threads in parallel on available CPU cores. While not all software will benefit from this automatically, programs designed with threading in mind will run faster on multicore hardware.

Work is still ongoing: expect some bugs and a substantial single-threaded performance hit.

The free-threaded build still supports optionally running with the GIL enabled at runtime using the environment variable `PYTHON_GIL` or the command line option `-X gil`.

To check if the current interpreter is configured with `--disable-gil`, use `sysconfig.get_config_var("Py_GIL_DISABLED")`. To check if the GIL is actually disabled in the running process, the `sys._is_gil_enabled()` function can be used.

C-API extension modules need to be built specifically for the free-threaded build. Extensions that support running with the GIL disabled should use the `Py_mod_gil` slot. Extensions using single-phase init should use `PyUnstable_Module_SetGIL()` to indicate whether they support running with the GIL disabled. Importing C extensions that don't use these mechanisms will cause the GIL to be enabled, unless the GIL was explicitly disabled with the `PYTHON_GIL` environment variable or the `-X gil=0` option.

pip 24.1b1 or newer is required to install packages with C extensions in the free-threaded build.

5 Other Language Changes

- The `exec()` and `eval()` built-ins now accept their `globals` and `locals` namespace arguments as keywords. (Contibuted by Raphael Gaschignard in gh-105879)
- Allow the *count* argument of `str.replace()` to be a keyword. (Contributed by Hugo van Kemenade in gh-106487.)
- Compiler now strip indents from docstrings. This will reduce the size of bytecode cache (e.g. `.pyc` file). For example, cache file size for `sqlalchemy.orm.session` in SQLAlchemy 2.0 is reduced by about 5%. This change will affect tools using docstrings, like `doctest`. (Contributed by Inada Naoki in gh-81283.)
- The `compile()` built-in can now accept a new flag, `ast.PyCF_OPTIMIZED_AST`, which is similar to `ast.PyCF_ONLY_AST` except that the returned `AST` is optimized according to the value of the `optimize` argument. (Contributed by Irit Katriel in gh-108113).
- `multiprocessing`, `concurrent.futures`, `compileall`: Replace `os.cpu_count()` with `os.process_cpu_count()` to select the default number of worker threads and processes. Get the CPU affinity if supported. (Contributed by Victor Stinner in gh-109649.)
- `os.path.realpath()` now resolves MS-DOS style file names even if the file is not accessible. (Contributed by Moonsik Park in gh-82367.)
- Fixed a bug where a `global` declaration in an `except` block is rejected when the global is used in the `else` block. (Contributed by Irit Katriel in gh-111123.)
- Many functions now emit a warning if a boolean value is passed as a file descriptor argument. This can help catch some errors earlier. (Contributed by Serhiy Storchaka in gh-82626.)
- Added a new environment variable `PYTHON_FROZEN_MODULES`. It determines whether or not frozen modules are ignored by the import machinery, equivalent of the `-X frozen_modules` command-line option. (Contributed by Yilei Yang in gh-111374.)
- Add support for the perf profiler working without frame pointers through the new environment variable `PYTHON_PERF_JIT_SUPPORT` and command-line option `-X perf_jit` (Contributed by Pablo Galindo in gh-118518.)
- The new `PYTHON_HISTORY` environment variable can be used to change the location of a `.python_history` file. (Contributed by Levi Sabah, Zackery Spytz and Hugo van Kemenade in gh-73965.)
- Add `PythonFinalizationError` exception. This exception derived from `RuntimeError` is raised when an operation is blocked during the Python finalization.

  The following functions now raise PythonFinalizationError, instead of `RuntimeError`:

- `_thread.start_new_thread()`.
- `subprocess.Popen`.
- `os.fork()`.
- `os.forkpty()`.

(Contributed by Victor Stinner in gh-114570.)

- Added `name` and `mode` attributes for compressed and archived file-like objects in modules `bz2`, `lzma`, `tarfile` and `zipfile`. (Contributed by Serhiy Storchaka in gh-115961.)
- Allow controlling Expat >=2.6.0 reparse deferral (CVE-2023-52425) by adding five new methods:
  - `xml.etree.ElementTree.XMLParser.flush()`
  - `xml.etree.ElementTree.XMLPullParser.flush()`
  - `xml.parsers.expat.xmlparser.GetReparseDeferralEnabled()`
  - `xml.parsers.expat.xmlparser.SetReparseDeferralEnabled()`
  - `xml.sax.expatreader.ExpatParser.flush()`

  (Contributed by Sebastian Pipping in gh-115623.)
- The `ssl.create_default_context()` API now includes `ssl.VERIFY_X509_PARTIAL_CHAIN` and `ssl.VERIFY_X509_STRICT` in its default flags.

Note: `ssl.VERIFY_X509_STRICT` may reject pre-**RFC 5280** or malformed certificates that the underlying OpenSSL implementation otherwise would accept. While disabling this is not recommended, you can do so using:

```
ctx = ssl.create_default_context()
ctx.verify_flags &= ~ssl.VERIFY_X509_STRICT
```

(Contributed by William Woodruff in gh-112389.)

- The `configparser.ConfigParser` now accepts unnamed sections before named ones if configured to do so. (Contributed by Pedro Sousa Lacerda in gh-66449.)
- annotation scope within class scopes can now contain lambdas and comprehensions. Comprehensions that are located within class scopes are not inlined into their parent scope. (Contributed by Jelle Zijlstra in gh-109118 and gh-118160.)
- Classes have a new `__firstlineno__` attribute, populated by the compiler, with the line number of the first line of the class definition. (Contributed by Serhiy Storchaka in gh-118465.)
- `from __future__ import ...` statements are now just normal relative imports if dots are present before the module name. (Contributed by Jeremiah Gabriel Pascual in gh-118216.)

6 New Modules

- None.

7 Improved Modules

7.1 argparse

- Add parameter *deprecated* in methods `add_argument()` and `add_parser()` which allows to deprecate command-line options, positional arguments and subcommands. (Contributed by Serhiy Storchaka in gh-83648.)

7.2 array

- Add `'w'` type code (`Py_UCS4`) that can be used for Unicode strings. It can be used instead of `'u'` type code, which is deprecated. (Contributed by Inada Naoki in gh-80480.)
- Add `clear()` method in order to implement `MutableSequence`. (Contributed by Mike Zimin in gh-114894.)

7.3 ast

- The constructors of node types in the `ast` module are now stricter in the arguments they accept, and have more intuitive behaviour when arguments are omitted.

  If an optional field on an AST node is not included as an argument when constructing an instance, the field will now be set to `None`. Similarly, if a list field is omitted, that field will now be set to an empty list, and if a `ast.expr_context` field is omitted, it defaults to `Load()`. (Previously, in all cases, the attribute would be missing on the newly constructed AST node instance.)

  If other arguments are omitted, a `DeprecationWarning` is emitted. This will cause an exception in Python 3.15. Similarly, passing a keyword argument that does not map to a field on the AST node is now deprecated, and will raise an exception in Python 3.15.

  These changes do not apply to user-defined subclasses of `ast.AST`, unless the class opts in to the new behavior by setting the attribute `ast.AST._field_types`.

  (Contributed by Jelle Zijlstra in gh-105858, gh-117486, and gh-118851.)
- `ast.parse()` now accepts an optional argument *optimize* which is passed on to the `compile()` built-in. This makes it possible to obtain an optimized AST. (Contributed by Irit Katriel in gh-108113.)

7.4 asyncio

- `asyncio.loop.create_unix_server()` will now automatically remove the Unix socket when the server is closed. (Contributed by Pierre Ossman in gh-111246.)
- `asyncio.DatagramTransport.sendto()` will now send zero-length datagrams if called with an empty bytes object. The transport flow control also now accounts for the datagram header when calculating the buffer size. (Contributed by Jamie Phan in gh-115199.)
- Add `asyncio.Server.close_clients()` and `asyncio.Server.abort_clients()` methods which allow to more forcefully close an asyncio server. (Contributed by Pierre Ossman in gh-113538.)

- `asyncio.as_completed()` now returns an object that is both an asynchronous iterator and a plain iterator of awaitables. The awaitables yielded by asynchronous iteration include original task or future objects that were passed in, making it easier to associate results with the tasks being completed. (Contributed by Justin Arthur in gh-77714.)
- When `asyncio.TaskGroup.create_task()` is called on an inactive `asyncio.TaskGroup`, the given coroutine will be closed (which prevents a `RuntimeWarning` about the given coroutine being never awaited). (Contributed by Arthur Tacca and Jason Zhang in gh-115957.)
- Improved behavior of `asyncio.TaskGroup` when an external cancellation collides with an internal cancellation. For example, when two task groups are nested and both experience an exception in a child task simultaneously, it was possible that the outer task group would hang, because its internal cancellation was swallowed by the inner task group.

  In the case where a task group is cancelled externally and also must raise an `ExceptionGroup`, it will now call the parent task's `cancel()` method. This ensures that a `asyncio.CancelledError` will be raised at the next `await`, so the cancellation is not lost.

  An added benefit of these changes is that task groups now preserve the cancellation count (`asyncio.Task.cancelling()`).

  In order to handle some corner cases, `asyncio.Task.uncancel()` may now reset the undocumented `_must_cancel` flag when the cancellation count reaches zero.

  (Inspired by an issue reported by Arthur Tacca in gh-116720.)
- Add `asyncio.Queue.shutdown()` (along with `asyncio.QueueShutDown`) for queue termination. (Contributed by Laurie Opperman and Yves Duprat in gh-104228.)
- Accept a tuple of separators in `asyncio.StreamReader.readuntil()`, stopping when one of them is encountered. (Contributed by Bruce Merry in gh-81322.)

7.5 base64

- Add `base64.z85encode()` and `base64.z85decode()` functions which allow encoding and decoding Z85 data. See Z85 specification for more information. (Contributed by Matan Perelman in gh-75299.)

7.6 copy

- Add `copy.replace()` function which allows to create a modified copy of an object, which is especially useful for immutable objects. It supports named tuples created with the factory function `collections.namedtuple()`, `dataclass` instances, various `datetime` objects, `Signature` objects, `Parameter` objects, code object, and any user classes which define the `__replace__()` method. (Contributed by Serhiy Storchaka in gh-108751.)

7.7 dbm

- Add `dbm.gnu.gdbm.clear()` and `dbm.ndbm.ndbm.clear()` methods that remove all items from the database. (Contributed by Donghee Na in gh-107122.)
- Add new `dbm.sqlite3` backend, and make it the default `dbm` backend. (Contributed by Raymond Hettinger and Erlend E. Aasland in gh-100414.)

7.8 dis

- Change the output of `dis` module functions to show logical labels for jump targets and exception handlers, rather than offsets. The offsets can be added with the new `-O` command line option or the `show_offsets` parameter. (Contributed by Irit Katriel in gh-112137.)
- `get_instructions()` no longer represents cache entries as separate instructions. Instead, it returns them as part of the `Instruction`, in the new *cache_info* field. The *show_caches* argument to `get_instructions()` is deprecated and no longer has any effect. (Contributed by Irit Katriel in gh-112962.)

7.9 doctest

- Color is added to the output by default. This can be controlled via the new `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-117225.)
- The `doctest.DocTestRunner.run()` method now counts the number of skipped tests. Add `doctest.DocTestRunner.skips` and `doctest.TestResults.skipped` attributes. (Contributed by Victor Stinner in gh-108794.)

7.10 email

- `email.utils.getaddresses()` and `email.utils.parseaddr()` now return `('', '')` 2-tuples in more situations where invalid email addresses are encountered instead of potentially inaccurate values. Add optional *strict* parameter to these two functions: use `strict=False` to get the old behavior, accept malformed inputs. `getattr(email.utils, 'supports_strict_parsing', False)` can be used to check if the *strict* parameter is available. (Contributed by Thomas Dwyer and Victor Stinner for gh-102988 to improve the CVE-2023-27043 fix.)

7.11 fractions

- Formatting for objects of type `fractions.Fraction` now supports the standard format specification mini-language rules for fill, alignment, sign handling, minimum width and grouping. (Contributed by Mark Dickinson in gh-111320.)

7.12 gc

- The cyclic garbage collector is now incremental, which changes the meanings of the results of `gc.get_threshold()` and `gc.set_threshold()` as well as `gc.get_count()` and `gc.get_stats()`.
  - `gc.get_threshold()` returns a three-item tuple for backwards compatibility. The first value is the threshold for young collections, as before; the second value determines the rate at which the old collection is scanned (the default is 10, and higher values mean that the old collection is scanned more slowly). The third value is meaningless and is always zero.
  - `gc.set_threshold()` ignores any items after the second.
  - `gc.get_count()` and `gc.get_stats()` return the same format of results as before. The only difference is that instead of the results referring to the young, aging and old generations, the results refer to the young generation and the aging and collecting spaces of the old generation.

  In summary, code that attempted to manipulate the behavior of the cycle GC may not work exactly as intended, but it is very unlikely to be harmful. All other code will work just fine.

7.13 glob

- Add `glob.translate()` function that converts a path specification with shell-style wildcards to a regular expression. (Contributed by Barney Gale in gh-72904.)

7.14 importlib

- Previously deprecated `importlib.resources` functions are un-deprecated:
  - `is_resource()`
  - `open_binary()`
  - `open_text()`
  - `path()`
  - `read_binary()`
  - `read_text()`

  All now allow for a directory (or tree) of resources, using multiple positional arguments.

  For text-reading functions, the *encoding* and *errors* must now be given as keyword arguments.

  The `contents()` remains deprecated in favor of the full-featured `Traversable` API. However, there is now no plan to remove it.

  (Contributed by Petr Viktorin in gh-106532.)

7.15 io

- The `io.IOBase` finalizer now logs the `close()` method errors with `sys.unraisablehook`. Previously, errors were ignored silently by default, and only logged in Python Development Mode or on Python built on debug mode. (Contributed by Victor Stinner in gh-62948.)

7.16 ipaddress

- Add the `ipaddress.IPv4Address.ipv6_mapped` property, which returns the IPv4-mapped IPv6 address. (Contributed by Charles Machalow in gh-109466.)
- Fix `is_global` and `is_private` behavior in `IPv4Address`, `IPv6Address`, `IPv4Network` and `IPv6Network`.

7.17 itertools

- Added a `strict` option to `itertools.batched()`. This raises a `ValueError` if the final batch is shorter than the specified batch size. (Contributed by Raymond Hettinger in gh-113202.)

7.18 marshal

- Add the *allow_code* parameter in module functions. Passing `allow_code=False` prevents serialization and deserialization of code objects which are incompatible between Python versions. (Contributed by Serhiy Storchaka in gh-113626.)

7.19 math

- A new function `fma()` for fused multiply-add operations has been added. This function computes `x * y + z` with only a single round, and so avoids any intermediate loss of precision. It wraps the `fma()` function provided by C99, and follows the specification of the IEEE 754 "fusedMultiplyAdd" operation for special cases. (Contributed by Mark Dickinson and Victor Stinner in gh-73468.)

7.20 mimetypes

- Add the `guess_file_type()` function which works with file path. Passing file path instead of URL in `guess_type()` is soft deprecated. (Contributed by Serhiy Storchaka in gh-66543.)

7.21 mmap

- The `mmap.mmap` class now has an `seekable()` method that can be used when a seekable file-like object is required. The `seek()` method now returns the new absolute position. (Contributed by Donghee Na and Sylvie Liberman in gh-111835.)
- `mmap.mmap` now has a *trackfd* parameter on Unix; if it is `False`, the file descriptor specified by *fileno* will not be duplicated. (Contributed by Zackery Spytz and Petr Viktorin in gh-78502.)
- `mmap.mmap` is now protected from crashing on Windows when the mapped memory is inaccessible due to file system errors or access violations. (Contributed by Jannis Weigend in gh-118209.)

7.22 opcode

- Move `opcode.ENABLE_SPECIALIZATION` to `_opcode.ENABLE_SPECIALIZATION`. This field was added in 3.12, it was never documented and is not intended for external usage. (Contributed by Irit Katriel in gh-105481.)
- Removed `opcode.is_pseudo`, `opcode.MIN_PSEUDO_OPCODE` and `opcode.MAX_PSEUDO_OPCODE`, which were added in 3.12, were never documented or exposed through `dis`, and were not intended to be used externally.

7.23 os

- Add `os.process_cpu_count()` function to get the number of logical CPUs usable by the calling thread of the current process. (Contributed by Victor Stinner in gh-109649.)
- Add a low level interface for Linux's timer notification file descriptors via `os.timerfd_create()`, `os.timerfd_settime()`, `os.timerfd_settime_ns()`, `os.timerfd_gettime()`, and `os.timerfd_gettime_ns()`, `os.TFD_NONBLOCK`, `os.TFD_CLOEXEC`, `os.TFD_TIMER_ABSTIME`, and `os.TFD_TIMER_CANCEL_ON_SET` (Contributed by Masaru Tsuchiyama in gh-108277.)

- `os.cpu_count()` and `os.process_cpu_count()` can be overridden through the new environment variable `PYTHON_CPU_COUNT` or the new command-line option `-X cpu_count`. This option is useful for users who need to limit CPU resources of a container system without having to modify the container (application code). (Contributed by Donghee Na in gh-109595.)
- Add support of `os.lchmod()` and the *follow_symlinks* argument in `os.chmod()` on Windows. Note that the default value of *follow_symlinks* in `os.lchmod()` is `False` on Windows. (Contributed by Serhiy Storchaka in gh-59616.)
- Add support of `os.fchmod()` and a file descriptor in `os.chmod()` on Windows. (Contributed by Serhiy Storchaka in gh-113191.)
- `os.posix_spawn()` now accepts `env=None`, which makes the newly spawned process use the current process environment. (Contributed by Jakub Kulik in gh-113119.)
- `os.posix_spawn()` gains an `os.POSIX_SPAWN_CLOSEFROM` attribute for use in `file_actions=` on platforms that support `posix_spawn_file_actions_addclosefrom_np()`. (Contributed by Jakub Kulik in gh-113117.)
- `os.mkdir()` and `os.makedirs()` on Windows now support passing a *mode* value of `0o700` to apply access control to the new directory. This implicitly affects `tempfile.mkdtemp()` and is a mitigation for CVE-2024-4030. Other values for *mode* continue to be ignored. (Contributed by Steve Dower in gh-118486.)

7.24 os.path

- Add `os.path.isreserved()` to check if a path is reserved on the current system. This function is only available on Windows. (Contributed by Barney Gale in gh-88569.)
- On Windows, `os.path.isabs()` no longer considers paths starting with exactly one (back)slash to be absolute. (Contributed by Barney Gale and Jon Foster in gh-44626.)
- Add support of *dir_fd* and *follow_symlinks* keyword arguments in `shutil.chown()`. (Contributed by Berker Peksag and Tahia K in gh-62308)

7.25 pathlib

- Add `pathlib.UnsupportedOperation`, which is raised instead of `NotImplementedError` when a path operation isn't supported. (Contributed by Barney Gale in gh-89812.)
- Add `pathlib.Path.from_uri()`, a new constructor to create a `pathlib.Path` object from a 'file' URI (`file://`). (Contributed by Barney Gale in gh-107465.)
- Add `pathlib.PurePath.full_match()` for matching paths with shell-style wildcards, including the recursive wildcard "`**`". (Contributed by Barney Gale in gh-73435.)
- Add `pathlib.PurePath.parser` class attribute that stores the implementation of `os.path` used for low-level path parsing and joining: either `posixpath` or `ntpath`.
- Add *recurse_symlinks* keyword-only argument to `pathlib.Path.glob()` and `rglob()`. (Contributed by Barney Gale in gh-77609.)
- Add *follow_symlinks* keyword-only argument to `is_file()`, `is_dir()`, `owner()`, `group()`. (Contributed by Barney Gale in gh-105793, and Kamil Turek in gh-107962.)
- Return files and directories from `pathlib.Path.glob()` and `rglob()` when given a pattern that ends with "`**`". In earlier versions, only directories were returned. (Contributed by Barney Gale in gh-70303.)

7.26 pdb

- Add ability to move between chained exceptions during post mortem debugging in `pm()` using the new `exceptions [exc_number]` command for Pdb. (Contributed by Matthias Bussonnier in gh-106676.)
- Expressions/statements whose prefix is a pdb command are now correctly identified and executed. (Contributed by Tian Gao in gh-108464.)
- `sys.path[0]` will no longer be replaced by the directory of the script being debugged when `sys.flags.safe_path` is set (via the `-P` command line option or `PYTHONSAFEPATH` environment variable). (Contributed by Tian Gao and Christian Walther in gh-111762.)
- `zipapp` is supported as a debugging target. (Contributed by Tian Gao in gh-118501.)
- `breakpoint()` and `pdb.set_trace()` now enter the debugger immediately rather than on the next line of code to be executed. This change prevents the debugger from breaking outside of the context when `breakpoint()` is positioned at the end of the context. (Contributed by Tian Gao in gh-118579.)

7.27 queue

- Add `queue.Queue.shutdown()` (along with `queue.ShutDown`) for queue termination. (Contributed by Laurie Opperman and Yves Duprat in gh-104750.)

7.28 random

- Add a command-line interface. (Contributed by Hugo van Kemenade in gh-118131.)

7.29 re

- Rename `re.error` to `re.PatternError` for improved clarity. `re.error` is kept for backward compatibility.

7.30 site

- `.pth` files are now decoded by UTF-8 first, and then by the locale encoding if the UTF-8 decoding fails. (Contributed by Inada Naoki in gh-117802.)

7.31 sqlite3

- A `ResourceWarning` is now emitted if a `sqlite3.Connection` object is not `closed` explicitly. (Contributed by Erlend E. Aasland in gh-105539.)
- Add *filter* keyword-only parameter to `sqlite3.Connection.iterdump()` for filtering database objects to dump. (Contributed by Mariusz Felisiak in gh-91602.)

7.32 statistics

- Add `statistics.kde()` for kernel density estimation. This makes it possible to estimate a continuous probability density function from a fixed number of discrete samples. Also added `statistics.kde_random()` for sampling from the estimated probability density function. (Contributed by Raymond Hettinger in gh-115863.)

7.33 subprocess

- The `subprocess` module now uses the `os.posix_spawn()` function in more situations. Notably in the default case of `close_fds=True` on more recent versions of platforms including Linux, FreeBSD, and Solaris where the C library provides `posix_spawn_file_actions_addclosefrom_np()`. On Linux this should perform similar to our existing Linux `vfork()` based code. A private control knob `subprocess._USE_POSIX_SPAWN` can be set to `False` if you need to force `subprocess` not to ever use `os.posix_spawn()`. Please report your reason and platform details in the CPython issue tracker if you set this so that we can improve our API selection logic for everyone. (Contributed by Jakub Kulik in gh-113117.)

7.34 sys

- Add the `sys._is_interned()` function to test if the string was interned. This function is not guaranteed to exist in all implementations of Python. (Contributed by Serhiy Storchaka in gh-78573.)

7.35 tempfile

- On Windows, the default mode `0o700` used by `tempfile.mkdtemp()` now limits access to the new directory due to changes to `os.mkdir()`. This is a mitigation for CVE-2024-4030. (Contributed by Steve Dower in gh-118486.)

7.36 time

- On Windows, `time.monotonic()` now uses the `QueryPerformanceCounter()` clock to have a resolution better than 1 us, instead of the `GetTickCount64()` clock which has a resolution of 15.6 ms. (Contributed by Victor Stinner in gh-88494.)
- On Windows, `time.time()` now uses the `GetSystemTimePreciseAsFileTime()` clock to have a resolution better than 1 µs, instead of the `GetSystemTimeAsFileTime()` clock which has a resolution of 15.6 ms. (Contributed by Victor Stinner in gh-63207.)

7.37 tkinter

- Add `tkinter` widget methods: `tk_busy_hold()`, `tk_busy_configure()`, `tk_busy_cget()`, `tk_busy_forget()`, `tk_busy_current()`, and `tk_busy_status()`. (Contributed by Miguel, klappnase and Serhiy Storchaka in gh-72684.)
- The `tkinter` widget method `wm_attributes()` now accepts the attribute name without the minus prefix to get window attributes, e.g. `w.wm_attributes('alpha')` and allows to specify attributes and values to set as keyword arguments, e.g. `w.wm_attributes(alpha=0.5)`. Add new optional keyword-only parameter *return_python_dict*: calling `w.wm_attributes(return_python_dict=True)` returns the attributes as a dict instead of a tuple. (Contributed by Serhiy Storchaka in gh-43457.)

- Add new optional keyword-only parameter *return_ints* in the `Text.count()` method. Passing `return_ints=True` makes it always returning the single count as an integer instead of a 1-tuple or `None`. (Contributed by Serhiy Storchaka in gh-97928.)
- Add support of the "vsapi" element type in the `element_create()` method of `tkinter.ttk.Style`. (Contributed by Serhiy Storchaka in gh-68166.)
- Add the `after_info()` method for Tkinter widgets. (Contributed by Cheryl Sabella in gh-77020.)
- Add the `PhotoImage` method `copy_replace()` to copy a region from one image to other image, possibly with pixel zooming and/or subsampling. Add *from_coords* parameter to `PhotoImage` methods `copy()`, `zoom()` and `subsample()`. Add *zoom* and *subsample* parameters to `PhotoImage` method `copy()`. (Contributed by Serhiy Storchaka in gh-118225.)
- Add the `PhotoImage` methods `read()` to read an image from a file and `data()` to get the image data. Add *background* and *grayscale* parameters to `PhotoImage` method `write()`. (Contributed by Serhiy Storchaka in gh-118271.)

7.38 traceback

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

7.39 types

- `SimpleNamespace` constructor now allows specifying initial values of attributes as a positional argument which must be a mapping or an iterable of key-value pairs. (Contributed by Serhiy Storchaka in gh-108191.)

7.40 typing

- Add `typing.get_protocol_members()` to return the set of members defining a `typing.Protocol`. Add `typing.is_protocol()` to check whether a class is a `typing.Protocol`. (Contributed by Jelle Zijlstra in gh-104873.)
- Add `typing.ReadOnly`, a special typing construct to mark an item of a `typing.TypedDict` as read-only for type checkers. See **PEP 705** for more details.
- Add `typing.NoDefault`, a sentinel object used to represent the defaults of some parameters in the `typing` module. (Contributed by Jelle Zijlstra in gh-116126.)

7.41 unicodedata

- The Unicode database has been updated to version 15.1.0. (Contributed by James Gerity in gh-109559.)

7.42 venv

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

7.43 warnings

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

7.44 xml.etree.ElementTree

- Add the `close()` method for the iterator returned by `iterparse()` for explicit cleaning up. (Contributed by Serhiy Storchaka in gh-69893.)

7.45 zipimport

- Gains support for ZIP64 format files. Everybody loves huge code right? (Contributed by Tim Hatch in gh-94146.)

8 Optimizations

- `textwrap.indent()` is now ~30% faster than before for large input. (Contributed by Inada Naoki in gh-107369.)
- The `subprocess` module uses `os.posix_spawn()` in more situations including the default where `close_fds=True` on many modern platforms. This should provide a noteworthy performance increase launching processes on FreeBSD and Solaris. See the *subprocess* section above for details. (Contributed by Jakub Kulik in gh-113117.)
- Several standard library modules have had their import times significantly improved. For example, the import time of the `typing` module has been reduced by around a third by removing dependencies on `re` and `contextlib`. Other modules to enjoy import-time speedups include `importlib.metadata`, `threading`, `enum`, `functools` and `email.utils`. (Contributed by Alex Waygood, Shantanu Jain, Adam Turner, Daniel Hollas and others in gh-109653.)

9 Removed Modules And APIs

9.1 PEP 594: dead batteries (and other module removals)

- **PEP 594** removed 19 modules from the standard library, deprecated in Python 3.11:
  - `aifc`. (Contributed by Victor Stinner in gh-104773.)
  - `audioop`. (Contributed by Victor Stinner in gh-104773.)
  - `chunk`. (Contributed by Victor Stinner in gh-104773.)
  - `cgi` and `cgitb`.
    - `cgi.FieldStorage` can typically be replaced with `urllib.parse.parse_qsl()` for `GET` and `HEAD` requests, and the `email.message` module or multipart PyPI project for `POST` and `PUT`.
    - `cgi.parse()` can be replaced by calling `urllib.parse.parse_qs()` directly on the desired query string, except for `multipart/form-data` input, which can be handled as described for `cgi.parse_multipart()`.
    - `cgi.parse_header()` can be replaced with the functionality in the `email` package, which implements the same MIME RFCs. For example, with `email.message.EmailMessage`:

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

    - `cgi.parse_multipart()` can be replaced with the functionality in the `email` package (e.g. `email.message.EmailMessage` and `email.message.Message`) which implements the same MIME RFCs, or with the multipart PyPI project.

    (Contributed by Victor Stinner in gh-104773.)
  - `crypt` module and its private `_crypt` extension. The `hashlib` module is a potential replacement for certain use cases. Otherwise, the following PyPI projects can be used:
    - bcrypt: Modern password hashing for your software and your servers.
    - passlib: Comprehensive password hashing framework supporting over 30 schemes.
    - argon2-cffi: The secure Argon2 password hashing algorithm.
    - legacycrypt: `ctypes` wrapper to the POSIX crypt library call and associated functionality.
    - crypt_r: Fork of the `crypt` module, wrapper to the *`crypt_r(3)`* library call and associated functionality.

    (Contributed by Victor Stinner in gh-104773.)
  - `imghdr`: use the projects filetype, puremagic, or python-magic instead. The `puremagic.what()` function can be used to replace the `imghdr.what()` function for all file formats that were supported by `imghdr`. (Contributed by Victor Stinner in gh-104773.)
  - `mailcap`. The `mimetypes` module provides an alternative. (Contributed by Victor Stinner in gh-104773.)
  - `msilib`. (Contributed by Zachary Ware in gh-104773.)
  - `nis`. (Contributed by Victor Stinner in gh-104773.)
  - `nntplib`: the nntplib PyPI project can be used instead. (Contributed by Victor Stinner in gh-104773.)
  - `ossaudiodev`: use the pygame project for audio playback. (Contributed by Victor Stinner in gh-104780.)

  - `pipes`: use the `subprocess` module instead. (Contributed by Victor Stinner in gh-104773.)
  - `sndhdr`: use the projects filetype, puremagic, or python-magic instead. (Contributed by Victor Stinner in gh-104773.)
  - `spwd`: the python-pam project can be used instead. (Contributed by Victor Stinner in gh-104773.)
  - `sunau`. (Contributed by Victor Stinner in gh-104773.)
  - `telnetlib`, use the projects telnetlib3 or Exscript instead. (Contributed by Victor Stinner in gh-104773.)
  - `uu`: the `base64` module is a modern alternative. (Contributed by Victor Stinner in gh-104773.)
  - `xdrlib`. (Contributed by Victor Stinner in gh-104773.)
- Remove the `2to3` program and the `lib2to3` module, deprecated in Python 3.11. (Contributed by Victor Stinner in gh-104780.)
- Remove the `tkinter.tix` module, deprecated in Python 3.6. The third-party Tix library which the module wrapped is unmaintained. (Contributed by Zachary Ware in gh-75552.)

9.2 configparser

- Remove the undocumented `configparser.LegacyInterpolation` class, deprecated in the docstring since Python 3.2, and with a deprecation warning since Python 3.11. (Contributed by Hugo van Kemenade in gh-104886.)

9.3 importlib

- Remove deprecated `__getitem__()` access for `importlib.metadata.EntryPoint` objects. (Contributed by Jason R. Coombs in gh-113175.)

9.4 locale

- Remove `locale.resetlocale()` function deprecated in Python 3.11: use `locale.setlocale(locale.LC_ALL, "")` instead. (Contributed by Victor Stinner in gh-104783.)

9.5 logging

- `logging`: Remove undocumented and untested `Logger.warn()` and `LoggerAdapter.warn()` methods and `logging.warn()` function. Deprecated since Python 3.3, they were aliases to the `logging.Logger.warning()` method, `logging.LoggerAdapter.warning()` method and `logging.warning()` function. (Contributed by Victor Stinner in gh-105376.)

9.6 pathlib

- Remove support for using `pathlib.Path` objects as context managers. This functionality was deprecated and made a no-op in Python 3.9.

9.7 re

- Remove undocumented, never working, and deprecated `re.template` function and `re.TEMPLATE` flag (and `re.T` alias). (Contributed by Serhiy Storchaka and Nikita Sobolev in gh-105687.)

9.8 turtle

- Remove the `turtle.RawTurtle.settiltangle()` method, deprecated in docs since Python 3.1 and with a deprecation warning since Python 3.11. (Contributed by Hugo van Kemenade in gh-104876.)

9.9 typing

- Namespaces `typing.io` and `typing.re`, deprecated in Python 3.8, are now removed. The items in those namespaces can be imported directly from `typing`. (Contributed by Sebastian Rittau in gh-92871.)
- Remove support for the keyword-argument method of creating `typing.TypedDict` types, deprecated in Python 3.11. (Contributed by Tomas Roun in gh-104786.)

9.10 unittest

- Remove the following `unittest` functions, deprecated in Python 3.11:
  - `unittest.findTestCases()`
  - `unittest.makeSuite()`
  - `unittest.getTestCaseNames()`

  Use `TestLoader` methods instead:
  - `unittest.TestLoader.loadTestsFromModule()`
  - `unittest.TestLoader.loadTestsFromTestCase()`
  - `unittest.TestLoader.getTestCaseNames()`

  (Contributed by Hugo van Kemenade in gh-104835.)
- Remove the untested and undocumented `unittest.TestProgram.usageExit()` method, deprecated in Python 3.11. (Contributed by Hugo van Kemenade in gh-104992.)

9.11 urllib

- Remove *cafile*, *capath* and *cadefault* parameters of the `urllib.request.urlopen()` function, deprecated in Python 3.6: pass the *context* parameter instead. Use `ssl.SSLContext.load_cert_chain()` to load specific certificates, or let `ssl.create_default_context()` select the system's trusted CA certificates for you. (Contributed by Victor Stinner in gh-105382.)

9.12 webbrowser

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

10 New Deprecations

- Removed chained `classmethod` descriptors (introduced in gh-63272). This can no longer be used to wrap other descriptors such as `property`. The core design of this feature was flawed and caused a number of downstream problems. To "pass-through" a `classmethod`, consider using the `__wrapped__` attribute that was added in Python 3.10. (Contributed by Raymond Hettinger in gh-89519.)
- `array`: `array`'s `'u'` format code, deprecated in docs since Python 3.3, emits `DeprecationWarning` since 3.13 and will be removed in Python 3.16. Use the `'w'` format code instead. (Contributed by Hugo van Kemenade in gh-80480.)
- `ctypes`: Deprecate undocumented `ctypes.SetPointerType()` and `ctypes.ARRAY()` functions. Replace `ctypes.ARRAY(item_type, size)` with `item_type * size`. (Contributed by Victor Stinner in gh-105733.)
- `decimal`: Deprecate non-standard format specifier "N" for `decimal.Decimal`. It was not documented and only supported in the C implementation. (Contributed by Serhiy Storchaka in gh-89902.)
- `dis`: The `dis.HAVE_ARGUMENT` separator is deprecated. Check membership in `hasarg` instead. (Contributed by Irit Katriel in gh-109319.)
- frame-objects: Calling `frame.clear()` on a suspended frame raises `RuntimeError` (as has always been the case for an executing frame). (Contributed by Irit Katriel in gh-79932.)
- `getopt` and `optparse` modules: They are now soft deprecated: the `argparse` module should be used for new projects. Previously, the `optparse` module was already deprecated, its removal was not scheduled, and no warnings was emitted: so there is no change in practice. (Contributed by Victor Stinner in gh-106535.)
- `gettext`: Emit deprecation warning for non-integer numbers in `gettext` functions and methods that consider plural forms even if the translation was not found. (Contributed by Serhiy Storchaka in gh-88434.)
- `glob`: The undocumented `glob.glob0()` and `glob.glob1()` functions are deprecated. Use `glob.glob()` and pass a directory to its *root_dir* argument instead. (Contributed by Barney Gale in gh-117337.)
- `http.server`: `http.server.CGIHTTPRequestHandler` now emits a `DeprecationWarning` as it will be removed in 3.15. Process-based CGI HTTP servers have been out of favor for a very long time. This code was outdated, unmaintained, and rarely used. It has a high potential for both security and functionality bugs. This includes removal of the `--cgi` flag to the `python -m http.server` command line in 3.15.
- `mimetypes`: Passing file path instead of URL in `guess_type()` is soft deprecated. Use `guess_file_type()` instead. (Contributed by Serhiy Storchaka in gh-66543.)

- re: Passing optional arguments *maxsplit*, *count* and *flags* in module-level functions `re.split()`, `re.sub()` and `re.subn()` as positional arguments is now deprecated. In future Python versions these parameters will be keyword-only. (Contributed by Serhiy Storchaka in gh-56166.)

- pathlib: `pathlib.PurePath.is_reserved()` is deprecated and scheduled for removal in Python 3.15. Use `os.path.isreserved()` to detect reserved paths on Windows.

- platform: `java_ver()` is deprecated and will be removed in 3.15. It was largely untested, had a confusing API, and was only useful for Jython support. (Contributed by Nikita Sobolev in gh-116349.)

- pydoc: Deprecate undocumented `pydoc.ispackage()` function. (Contributed by Zackery Spytz in gh-64020.)

- sqlite3: Passing more than one positional argument to `sqlite3.connect()` and the `sqlite3.Connection` constructor is deprecated. The remaining parameters will become keyword-only in Python 3.15.

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

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

  Deprecate passing the callback callable by keyword for the following `sqlite3.Connection` APIs:

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

  The affected parameters will become positional-only in Python 3.15.

  (Contributed by Erlend E. Aasland in gh-107948 and gh-108278.)

- sys: `sys._enablelegacywindowsfsencoding()` function. Replace it with the `PYTHONLEGACYWINDOWSFSENCODING` environment variable. (Contributed by Inada Naoki in gh-73427.)

- tarfile: The undocumented and unused `tarfile` attribute of `tarfile.TarFile` is deprecated and scheduled for removal in Python 3.16.

- traceback: The field *exc_type* of `traceback.TracebackException` is deprecated. Use *exc_type_str* instead.

- typing:

  - Creating a `typing.NamedTuple` class using keyword arguments to denote the fields (`NT = NamedTuple("NT", x=int, y=int)`) is deprecated, and will be disallowed in Python 3.15. Use the class-based syntax or the functional syntax instead. (Contributed by Alex Waygood in gh-105566.)

  - When using the functional syntax to create a `typing.NamedTuple` class or a `typing.TypedDict` class, failing to pass a value to the 'fields' parameter (`NT = NamedTuple("NT")` or `TD = TypedDict("TD")`) is deprecated. Passing `None` to the 'fields' parameter (`NT = NamedTuple("NT", None)` or `TD = TypedDict("TD", None)`) is also deprecated. Both will be disallowed in Python 3.15. To create a NamedTuple class with 0 fields, use `class NT(NamedTuple): pass` or `NT = NamedTuple("NT", [])`. To create a TypedDict class with 0 fields, use `class TD(TypedDict): pass` or `TD = TypedDict("TD", {})`. (Contributed by Alex Waygood in gh-105566 and gh-105570.)

  - `typing.no_type_check_decorator()` is deprecated, and scheduled for removal in Python 3.15. After eight years in the `typing` module, it has yet to be supported by any major type checkers. (Contributed by Alex Waygood in gh-106309.)

  - `typing.AnyStr` is deprecated. In Python 3.16, it will be removed from `typing.__all__`, and a `DeprecationWarning` will be emitted when it is imported or accessed. It will be removed entirely in Python 3.18. Use the new type parameter syntax instead. (Contributed by Michael The in gh-107116.)

- user-defined-funcs: Assignment to a function's `__code__` attribute where the new code object's type does not match the function's type, is deprecated. The different types are: plain function, generator, async generator and coroutine. (Contributed by Irit Katriel in gh-81137.)

- `wave`: Deprecate the `getmark()`, `setmark()` and `getmarkers()` methods of the `wave.Wave_read` and `wave.Wave_write` classes. They will be removed in Python 3.15. (Contributed by Victor Stinner in gh-105096.)

10.1 Pending Removal in Python 3.14

- `argparse`: The *type*, *choices*, and *metavar* parameters of `argparse.BooleanOptionalAction` are deprecated and will be removed in 3.14. (Contributed by Nikita Sobolev in gh-92248.)

- `ast`: The following features have been deprecated in documentation since Python 3.8, now cause a `DeprecationWarning` to be emitted at runtime when they are accessed or used, and will be removed in Python 3.14:

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

  Use `ast.Constant` instead. (Contributed by Serhiy Storchaka in gh-90953.)

- `collections.abc`: Deprecated `ByteString`. Prefer `Sequence` or `Buffer`. For use in typing, prefer a union, like `bytes | bytearray`, or `collections.abc.Buffer`. (Contributed by Shantanu Jain in gh-91896.)

- `email`: Deprecated the *isdst* parameter in `email.utils.localtime()`. (Contributed by Alan Williams in gh-72346.)

- `importlib`: `__package__` and `__cached__` will cease to be set or taken into consideration by the import system (gh-97879).

- `importlib.abc` deprecated 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.)

- `itertools` had undocumented, inefficient, historically buggy, and inconsistent support for copy, deepcopy, and pickle operations. This will be removed in 3.14 for a significant reduction in code volume and maintenance burden. (Contributed by Raymond Hettinger in gh-101588.)

- `multiprocessing`: The default start method will change to a safer one on Linux, BSDs, and other non-macOS POSIX platforms where `'fork'` is currently the default (gh-84559). Adding a runtime warning about this was deemed too disruptive as the majority of code is not expected to care. Use the `get_context()` or `set_start_method()` APIs to explicitly specify when your code *requires* `'fork'`. See multiprocessing-start-methods.
- `pathlib`: `is_relative_to()` and `relative_to()`: passing additional arguments is deprecated.
- `pkgutil`: `find_loader()` and `get_loader()` now raise `DeprecationWarning`; use `importlib.util.find_spec()` instead. (Contributed by Nikita Sobolev in gh-97850.)
- `pty`:
  - `master_open()`: use `pty.openpty()`.
  - `slave_open()`: use `pty.openpty()`.
- `sqlite3`:
  - `version` and `version_info`.
  - `execute()` and `executemany()` if named placeholders are used and *parameters* is a sequence instead of a `dict`.
  - date and datetime adapter, date and timestamp converter: see the `sqlite3` documentation for suggested replacement recipes.
- `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 3.14. (Contributed by Nikita Sobolev in gh-101866.)
- `typing`: `ByteString`, deprecated since Python 3.9, now causes a `DeprecationWarning` to be emitted when it is used.
- `urllib`: `urllib.parse.Quoter` is deprecated: it was not intended to be a public API. (Contributed by Gregory P. Smith in gh-88168.)

10.2 Pending Removal in Python 3.15

- `http.server.CGIHTTPRequestHandler` will be removed along with its related `--cgi` flag to `python -m http.server`. It was obsolete and rarely used. No direct replacement exists. *Anything* is better than CGI to interface a web server with a request handler.
- `locale`: `locale.getdefaultlocale()` was deprecated in Python 3.11 and originally planned for removal in Python 3.13 (gh-90817), but removal has been postponed to Python 3.15. Use `locale.setlocale()`, `locale.getencoding()` and `locale.getlocale()` instead. (Contributed by Hugo van Kemenade in gh-111187.)
- `pathlib`: `pathlib.PurePath.is_reserved()` is deprecated and scheduled for removal in Python 3.15. Use `os.path.isreserved()` to detect reserved paths on Windows.
- `platform`: `java_ver()` is deprecated and will be removed in 3.15. It was largely untested, had a confusing API, and was only useful for Jython support. (Contributed by Nikita Sobolev in gh-116349.)
- `threading`: Passing any arguments to `threading.RLock()` is now deprecated. C version allows any numbers of args and kwargs, but they are just ignored. Python version does not allow any arguments. All arguments will be removed from `threading.RLock()` in Python 3.15. (Contributed by Nikita Sobolev in gh-102029.)
- `typing.NamedTuple`:

  - The undocumented keyword argument syntax for creating `NamedTuple` classes (`NT = NamedTuple("NT", x=int)`) is deprecated, and will be disallowed in 3.15. Use the class-based syntax or the functional syntax instead.
  - When using the functional syntax to create a `NamedTuple` class, failing to pass a value to the *fields* parameter (`NT = NamedTuple("NT")`) is deprecated. Passing `None` to the *fields* parameter (`NT = NamedTuple("NT", None)`) is also deprecated. Both will be disallowed in Python 3.15. To create a `NamedTuple` class with 0 fields, use `class NT(NamedTuple): pass` or `NT = NamedTuple("NT", [])`.

- `typing.TypedDict`: When using the functional syntax to create a `TypedDict` class, failing to pass a value to the *fields* parameter (`TD = TypedDict("TD")`) is deprecated. Passing `None` to the *fields* parameter (`TD = TypedDict("TD", None)`) is also deprecated. Both will be disallowed in Python 3.15. To create a `TypedDict` class with 0 fields, use `class TD(TypedDict): pass` or `TD = TypedDict("TD", {})`.
- `wave`: Deprecate the `getmark()`, `setmark()` and `getmarkers()` methods of the `wave.Wave_read` and `wave.Wave_write` classes. They will be removed in Python 3.15. (Contributed by Victor Stinner in gh-105096.)

10.3 Pending Removal in Python 3.16

- `array.array` `'u'` type (`wchar_t`): use the `'w'` type instead (`Py_UCS4`).

10.4 Pending Removal in Future Versions

The following APIs were deprecated in earlier Python versions and will be removed, although there is currently no date scheduled for their removal.

- `argparse`: Nesting argument groups and nesting mutually exclusive groups are deprecated.
- `builtins`:
  - `~bool`, bitwise inversion on bool.
  - `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.
- `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.
- `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 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.

11 CPython Bytecode Changes

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

12 C API Changes

12.1 New Features

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

  New functions return not only `1` for true and `0` for false, but also `-1` for error.

  (Contributed by Serhiy Storchaka in gh-108511.)
- If Python is built in debug mode or `with assertions`, `PyTuple_SET_ITEM()` and `PyList_SET_ITEM()` now check the index argument with an assertion. (Contributed by Victor Stinner in gh-106168.)
- Add `PyModule_Add()` function: similar to `PyModule_AddObjectRef()` and `PyModule_AddObject()` but always steals a reference to the value. (Contributed by Serhiy Storchaka in gh-86493.)
- Add `PyDict_GetItemRef()` and `PyDict_GetItemStringRef()` functions: similar to `PyDict_GetItemWithError()` but returning a strong reference instead of a borrowed reference. Moreover, these functions return -1 on error and so checking `PyErr_Occurred()` is not needed. (Contributed by Victor Stinner in gh-106004.)
- Added `PyDict_SetDefaultRef()`, which is similar to `PyDict_SetDefault()` but returns a strong reference instead of a borrowed reference. This function returns `-1` on error, `0` on insertion, and `1` if the key was already present in the dictionary. (Contributed by Sam Gross in gh-112066.)

- Add `PyDict_ContainsString()` function: same as `PyDict_Contains()`, but *key* is specified as a `const char*` UTF-8 encoded bytes string, rather than a `PyObject*`. (Contributed by Victor Stinner in gh-108314.)
- Added `PyList_GetItemRef()` function: similar to `PyList_GetItem()` but returns a strong reference instead of a borrowed reference.
- Add `Py_IsFinalizing()` function: check if the main Python interpreter is shutting down. (Contributed by Victor Stinner in gh-108014.)
- Add `PyLong_AsInt()` function: similar to `PyLong_AsLong()`, but store the result in a C `int` instead of a C `long`. Previously, it was known as the private function `_PyLong_AsInt()` (with an underscore prefix). (Contributed by Victor Stinner in gh-108014.)
- Python built with `configure --with-trace-refs` (tracing references) now supports the Limited API. (Contributed by Victor Stinner in gh-108634.)
- Add `PyObject_VisitManagedDict()` and `PyObject_ClearManagedDict()` functions which must be called by the traverse and clear functions of a type using `Py_TPFLAGS_MANAGED_DICT` flag. The pythoncapi-compat project can be used to get these functions on Python 3.11 and 3.12. (Contributed by Victor Stinner in gh-107073.)
- Add `PyUnicode_EqualToUTF8AndSize()` and `PyUnicode_EqualToUTF8()` functions: compare Unicode object with a `const char*` UTF-8 encoded string and return true (`1`) if they are equal, or false (`0`) otherwise. These functions do not raise exceptions. (Contributed by Serhiy Storchaka in gh-110289.)
- Add `PyThreadState_GetUnchecked()` function: similar to `PyThreadState_Get()`, but don't kill the process with a fatal error if it is NULL. The caller is responsible to check if the result is NULL. Previously, the function was private and known as `_PyThreadState_UncheckedGet()`. (Contributed by Victor Stinner in gh-108867.)
- Add `PySys_AuditTuple()` function: similar to `PySys_Audit()`, but pass event arguments as a Python `tuple` object. (Contributed by Victor Stinner in gh-85283.)
- `PyArg_ParseTupleAndKeywords()` now supports non-ASCII keyword parameter names. (Contributed by Serhiy Storchaka in gh-110815.)
- Add `PyMem_RawMalloc()`, `PyMem_RawCalloc()`, `PyMem_RawRealloc()` and `PyMem_RawFree()` to the limited C API (version 3.13). (Contributed by Victor Stinner in gh-85283.)
- Add `PySys_Audit()` and `PySys_AuditTuple()` functions to the limited C API. (Contributed by Victor Stinner in gh-85283.)
- Add `PyErr_FormatUnraisable()` function: similar to `PyErr_WriteUnraisable()`, but allow customizing the warning message. (Contributed by Serhiy Storchaka in gh-108082.)
- Add `PyList_Extend()` and `PyList_Clear()` functions: similar to Python `list.extend()` and `list.clear()` methods. (Contributed by Victor Stinner in gh-111138.)
- Add `PyDict_Pop()` and `PyDict_PopString()` functions: remove a key from a dictionary and optionally return the removed value. This is similar to `dict.pop()`, but without the default value and not raising `KeyError` if the key is missing. (Contributed by Stefan Behnel and Victor Stinner in gh-111262.)
- Add `Py_HashPointer()` function to hash a pointer. (Contributed by Victor Stinner in gh-111545.)
- Add `PyObject_GenericHash()` function that implements the default hashing function of a Python object. (Contributed by Serhiy Storchaka in gh-113024.)
- Add PyTime C API:
  - `PyTime_t` type.
  - `PyTime_MIN` and `PyTime_MAX` constants.

- Add functions:
  - `PyTime_AsSecondsDouble()`.
  - `PyTime_Monotonic()`.
  - `PyTime_MonotonicRaw()`.
  - `PyTime_PerfCounter()`.
  - `PyTime_PerfCounterRaw()`.
  - `PyTime_Time()`.
  - `PyTime_TimeRaw()`.

  (Contributed by Victor Stinner and Petr Viktorin in gh-110850.)

- Add `PyLong_AsNativeBytes()`, `PyLong_FromNativeBytes()` and `PyLong_FromUnsignedNativeBytes()` functions to simplify converting between native integer types and Python `int` objects. (Contributed by Steve Dower in gh-111140.)
- Add `PyType_GetFullyQualifiedName()` function to get the type's fully qualified name. Equivalent to `f"{type.__module__}.{type.__qualname__}"`, or `type.__qualname__` if `type.__module__` is not a string or is equal to `"builtins"`. (Contributed by Victor Stinner in gh-111696.)
- Add `PyType_GetModuleName()` function to get the type's module name. Equivalent to getting the `type.__module__` attribute. (Contributed by Eric Snow and Victor Stinner in gh-111696.)
- Add support for `%T`, `%#T`, `%N` and `%#N` formats to `PyUnicode_FromFormat()`: format the fully qualified name of an object type and of a type: call `PyType_GetModuleName()`. See **PEP 737** for more information. (Contributed by Victor Stinner in gh-111696.)
- Add `Py_GetConstant()` and `Py_GetConstantBorrowed()` functions to get constants. For example, `Py_GetConstant(Py_CONSTANT_ZERO)` returns a strong reference to the constant zero. (Contributed by Victor Stinner in gh-115754.)
- Add `PyType_GetModuleByDef()` to the limited C API (Contributed by Victor Stinner in gh-116936.)
- Add two new functions to the C-API, `PyRefTracer_SetTracer()` and `PyRefTracer_GetTracer()`, that allows to track object creation and destruction the same way the `tracemalloc` module does. (Contributed by Pablo Galindo in gh-93502.)
- Add `PyEval_GetFrameBuiltins()`, `PyEval_GetFrameGlobals()`, and `PyEval_GetFrameLocals()` to the C API. These replacements for `PyEval_GetBuiltins()`, `PyEval_GetGlobals()`, and `PyEval_GetLocals()` return strong references rather than borrowed references. (Added as part of **PEP 667**.)
- Add `PyMutex` API, a lightweight mutex that occupies a single byte. The `PyMutex_Lock()` function will release the GIL (if currently held) if the operation needs to block. (Contributed by Sam Gross in gh-108724.)

13 Build Changes

- The `configure` option `--with-system-libmpdec` now defaults to `yes`. The bundled copy of `libmpdecimal` will be removed in Python 3.15.
- Autoconf 2.71 and aclocal 1.16.4 are now required to regenerate the `configure` script. (Contributed by Christian Heimes in gh-89886.)
- SQLite 3.15.2 or newer is required to build the `sqlite3` extension module. (Contributed by Erlend Aasland in gh-105875.)

- Python built with `configure --with-trace-refs` (tracing references) is now ABI compatible with the Python release build and debug build. (Contributed by Victor Stinner in gh-108634.)
- Building CPython now requires a compiler with support for the C11 atomic library, GCC built-in atomic functions, or MSVC interlocked intrinsics.
- The `errno`, `fcntl`, `grp`, `md5`, `pwd`, `resource`, `termios`, `winsound`, `_ctypes_test`, `_multiprocessing.posixshmem`, `_scproxy`, `_stat`, `_statistics`, `_testconsole`, `_testimportmultiple` and `_uuid` C extensions are now built with the limited C API. (Contributed by Victor Stinner in gh-85283.)
- `wasm32-wasi` is now a **PEP 11** tier 2 platform. (Contributed by Brett Cannon in gh-115192.)
- `wasm32-emscripten` is no longer a **PEP 11** supported platform. (Contributed by Brett Cannon in gh-115192.)
- Python now bundles the mimalloc library. It is licensed under the MIT license; see mimalloc license. The bundled mimalloc has custom changes, see gh-113141 for details. (Contributed by Dino Viehland in gh-109914.)
- On POSIX systems, the pkg-config (`.pc`) filenames now include the ABI flags. For example, the free-threaded build generates `python-3.13t.pc` and the debug build generates `python-3.13d.pc`.

14 Porting to Python 3.13

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

14.1 Changes in the Python API

- An `OSError` is now raised by `getpass.getuser()` for any failure to retrieve a username, instead of `ImportError` on non-Unix platforms or `KeyError` on Unix platforms where the password database is empty.
- The `threading` module now expects the `_thread` module to have an `_is_main_interpreter` attribute. It is a function with no arguments that returns `True` if the current interpreter is the main interpreter.

  Any library or application that provides a custom `_thread` module must provide `_is_main_interpreter()`, just like the module's other "private" attributes. (See gh-112826.)
- `mailbox.Maildir` now ignores files with a leading dot. (Contributed by Zackery Spytz in gh-65559.)
- `pathlib.Path.glob()` and `rglob()` now return both files and directories if a pattern that ends with "`**`" is given, rather than directories only. Users may add a trailing slash to match only directories.
- The value of the `mode` attribute of `gzip.GzipFile` was changed from integer (`1` or `2`) to string (`'rb'` or `'wb'`). The value of the `mode` attribute of the readable file-like object returned by `zipfile.ZipFile.open()` was changed from `'r'` to `'rb'`. (Contributed by Serhiy Storchaka in gh-115961.)
- `functools.partial` now emits a `FutureWarning` when it is used as a method. Its behavior will be changed in future Python versions. Wrap it in `staticmethod()` if you want to preserve the old behavior. (Contributed by Serhiy Storchaka in gh-121027.)
- Calling `locals()` in an optimized scope now produces an independent snapshot on each call, and hence no longer implicitly updates previously returned references. Obtaining the legacy CPython behaviour now requires explicit calls to update the initially returned dictionary with the results of subsequent calls to `locals()`. Code execution functions that implicitly target `locals()` (such as `exec` and `eval`) must be passed an explicit namespace to access their results in an optimized scope. (Changed as part of **PEP 667**.)
- Calling `locals()` from a comprehension at module or class scope (including via `exec` or `eval`) once more behaves as if the comprehension were running as an independent nested function (i.e. the local variables from

the containing scope are not included). In Python 3.12, this had changed to include the local variables from the containing scope when implementing **PEP 709**. (Changed as part of **PEP 667**.)

- Accessing `FrameType.f_locals` in an optimized scope now returns a write-through proxy rather than a snapshot that gets updated at ill-specified times. If a snapshot is desired, it must be created explicitly with `dict` or the proxy's `.copy()` method. (Changed as part of **PEP 667**.)

14.2 Changes in the C API

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

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

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

  should migrate to the new macros as follows:

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

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

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

14.3 Removed C APIs

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

  (Contributed by Victor Stinner in gh-105107.)
- Remove old buffer protocols deprecated in Python 3.0. Use bufferobjects instead.
  - `PyObject_CheckReadBuffer()`: Use `PyObject_CheckBuffer()` to test if the object supports the buffer protocol. Note that `PyObject_CheckBuffer()` doesn't guarantee that `PyObject_GetBuffer()` will succeed. To test if the object is actually readable, see the next example of `PyObject_GetBuffer()`.

- `PyObject_AsCharBuffer()`, `PyObject_AsReadBuffer()`: `PyObject_GetBuffer()` and `PyBuffer_Release()` instead:

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

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

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

(Contributed by Inada Naoki in gh-85275.)

- Remove the following old functions to configure the Python initialization, deprecated in Python 3.11:
  - `PySys_AddWarnOptionUnicode()`: use `PyConfig.warnoptions` instead.
  - `PySys_AddWarnOption()`: use `PyConfig.warnoptions` instead.
  - `PySys_AddXOption()`: use `PyConfig.xoptions` instead.
  - `PySys_HasWarnOptions()`: use `PyConfig.xoptions` instead.
  - `PySys_SetPath()`: set `PyConfig.module_search_paths` instead.
  - `Py_SetPath()`: set `PyConfig.module_search_paths` instead.
  - `Py_SetStandardStreamEncoding()`: set `PyConfig.stdio_encoding` instead, and set also maybe `PyConfig.legacy_windows_stdio` (on Windows).
  - `_Py_SetProgramFullPath()`: set `PyConfig.executable` instead.

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

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

- Remove `PyEval_AcquireLock()` and `PyEval_ReleaseLock()` functions, deprecated in Python 3.2. They didn't update the current thread state. They can be replaced with:
  - `PyEval_SaveThread()` and `PyEval_RestoreThread()`;
  - low-level `PyEval_AcquireThread()` and `PyEval_RestoreThread()`;
  - or `PyGILState_Ensure()` and `PyGILState_Release()`.

  (Contributed by Victor Stinner in gh-105182.)

- Remove private `_PyObject_FastCall()` function: use `PyObject_Vectorcall()` which is available since Python 3.8 (**PEP 590**). (Contributed by Victor Stinner in gh-106023.)

- Remove `cpython/pytime.h` header file: it only contained private functions. (Contributed by Victor Stinner in gh-106316.)
- Remove `_PyInterpreterState_Get()` alias to `PyInterpreterState_Get()` which was kept for backward compatibility with Python 3.8. The pythoncapi-compat project can be used to get `PyInterpreterState_Get()` on Python 3.8 and older. (Contributed by Victor Stinner in gh-106320.)
- The `PyModule_AddObject()` function is now soft deprecated: `PyModule_Add()` or `PyModule_AddObjectRef()` functions should be used instead. (Contributed by Serhiy Storchaka in gh-86493.)

14.4 Deprecated C APIs

- Deprecate the old `Py_UNICODE` and `PY_UNICODE_TYPE` types: use directly the `wchar_t` type instead. Since Python 3.3, `Py_UNICODE` and `PY_UNICODE_TYPE` are just aliases to `wchar_t`. (Contributed by Victor Stinner in gh-105156.)
- Deprecate old Python initialization functions:
  - `PySys_ResetWarnOptions()`: clear `sys.warnoptions` and `warnings.filters` instead.
  - `Py_GetExecPrefix()`: get `sys.exec_prefix` instead.
  - `Py_GetPath()`: get `sys.path` instead.
  - `Py_GetPrefix()`: get `sys.prefix` instead.
  - `Py_GetProgramFullPath()`: get `sys.executable` instead.
  - `Py_GetProgramName()`: get `sys.executable` instead.
  - `Py_GetPythonHome()`: get `PyConfig.home` or `PYTHONHOME` environment variable instead.

  Functions scheduled for removal in Python 3.15. (Contributed by Victor Stinner in gh-105145.)
- Deprecate the `PyImport_ImportModuleNoBlock()` function which is just an alias to `PyImport_ImportModule()` since Python 3.3. Scheduled for removal in Python 3.15. (Contributed by Victor Stinner in gh-105396.)
- Deprecate the `PyWeakref_GetObject()` and `PyWeakref_GET_OBJECT()` functions, which return a borrowed reference: use the new `PyWeakref_GetRef()` function instead, it returns a strong reference. The pythoncapi-compat project can be used to get `PyWeakref_GetRef()` on Python 3.12 and older. (Contributed by Victor Stinner in gh-105927.)
- Deprecate the `PyEval_GetBuiltins()`, `PyEval_GetGlobals()`, and `PyEval_GetLocals()` functions, which return a borrowed reference. Refer to the deprecation notices on each function for their recommended replacements. (Soft deprecated as part of **PEP 667**.)

14.5 Pending Removal in Python 3.14

- Creating immutable types (`Py_TPFLAGS_IMMUTABLETYPE`) with mutable bases using the C API.
- Functions to configure the Python 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.

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

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

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

14.6 Pending Removal in Python 3.15

- The bundled copy of `libmpdecimal`.
- `PyImport_ImportModuleNoBlock()`: use `PyImport_ImportModule()`.
- `PyWeakref_GET_OBJECT()`: use `PyWeakref_GetRef()` instead.
- `PyWeakref_GetObject()`: use `PyWeakref_GetRef()` instead.
- `Py_UNICODE_WIDE` type: use `wchar_t` instead.
- `Py_UNICODE` type: use `wchar_t` instead.
- Python initialization functions:
  - `PySys_ResetWarnOptions()`: clear `sys.warnoptions` and `warnings.filters` instead.
  - `Py_GetExecPrefix()`: get `sys.exec_prefix` instead.
  - `Py_GetPath()`: get `sys.path` instead.

  - `Py_GetPrefix()`: get `sys.prefix` instead.
  - `Py_GetProgramFullPath()`: get `sys.executable` instead.
  - `Py_GetProgramName()`: get `sys.executable` instead.
  - `Py_GetPythonHome()`: get `PyConfig.home` or `PYTHONHOME` environment variable instead.

14.7 Pending Removal in Future Versions

The following APIs were deprecated in earlier Python versions and will be removed, although there is currently no date scheduled for their removal.

- `Py_TPFLAGS_HAVE_FINALIZE`: no needed since Python 3.8.
- `PyErr_Fetch()`: use `PyErr_GetRaisedException()`.
- `PyErr_NormalizeException()`: use `PyErr_GetRaisedException()`.
- `PyErr_Restore()`: use `PyErr_SetRaisedException()`.
- `PyModule_GetFilename()`: use `PyModule_GetFilenameObject()`.
- `PyOS_AfterFork()`: use `PyOS_AfterFork_Child()`.
- `PySlice_GetIndicesEx()`.
- `PyUnicode_AsDecodedObject()`.
- `PyUnicode_AsDecodedUnicode()`.
- `PyUnicode_AsEncodedObject()`.
- `PyUnicode_AsEncodedUnicode()`.
- `PyUnicode_READY()`: not needed since Python 3.12.
- `_PyErr_ChainExceptions()`.
- `PyBytesObject.ob_shash` member: call `PyObject_Hash()` instead.
- `PyDictObject.ma_version_tag` member.
- TLS API:
  - `PyThread_create_key()`: use `PyThread_tss_alloc()`.
  - `PyThread_delete_key()`: use `PyThread_tss_free()`.
  - `PyThread_set_key_value()`: use `PyThread_tss_set()`.
  - `PyThread_get_key_value()`: use `PyThread_tss_get()`.
  - `PyThread_delete_key_value()`: use `PyThread_tss_delete()`.
  - `PyThread_ReInitTLS()`: no longer needed.
- Remove undocumented `PY_TIMEOUT_MAX` constant from the limited C API. (Contributed by Victor Stinner in gh-110014.)

15 Regression Test Changes

- Python built with `configure --with-pydebug` now supports a `-X presite=package.module` command-line option. If used, it specifies a module that should be imported early in the lifecycle of the interpreter, before `site.py` is executed. (Contributed by Łukasz Langa in gh-110769.)

Index

E

P

R