What's New in Python

Release 3.12.4

A. M. Kuchling

July 01, 2024

Python Software Foundation
Email: docs@python.org

Contents

Editor
Adam Turner

This article explains the new features in Python 3.12, compared to 3.11. Python 3.12 was released on October 2, 2023. For full details, see the changelog.

See also:

PEP 693 – Python 3.12 Release Schedule

1 Summary – Release highlights

Python 3.12 is the latest stable release of the Python programming language, with a mix of changes to the language and the standard library. The library changes focus on cleaning up deprecated APIs, usability, and correctness. Of note, the `distutils` package has been removed from the standard library. Filesystem support in `os` and `pathlib` has seen a number of improvements, and several modules have better performance.

The language changes focus on usability, as f-strings have had many limitations removed and 'Did you mean …' suggestions continue to improve. The new *type parameter syntax* and `type` statement improve ergonomics for using generic types and type aliases with static type checkers.

This article doesn't attempt to provide a complete specification of all new features, but instead gives a convenient overview. For full details, you should refer to the documentation, such as the Library Reference and Language Reference. If you want to understand the complete implementation and design rationale for a change, refer to the PEP for a particular new feature; but note that PEPs usually are not kept up-to-date once a feature has been fully implemented.

New syntax features:

- *PEP 695*, type parameter syntax and the `type` statement

New grammar features:

- *PEP 701*, f-strings in the grammar

Interpreter improvements:

- *PEP 684*, a unique per-interpreter GIL
- *PEP 669*, low impact monitoring
- *Improved 'Did you mean …' suggestions* for `NameError`, `ImportError`, and `SyntaxError` exceptions

Python data model improvements:

- *PEP 688*, using the buffer protocol from Python

Significant improvements in the standard library:

- The `pathlib.Path` class now supports subclassing
- The `os` module received several improvements for Windows support
- A command-line interface has been added to the `sqlite3` module
- `isinstance()` checks against `runtime-checkable protocols` enjoy a speed up of between two and 20 times
- The `asyncio` package has had a number of performance improvements, with some benchmarks showing a 75% speed up.
- A command-line interface has been added to the `uuid` module
- Due to the changes in *PEP 701*, producing tokens via the `tokenize` module is up to 64% faster.

Security improvements:

- Replace the builtin `hashlib` implementations of SHA1, SHA3, SHA2-384, SHA2-512, and MD5 with formally verified code from the HACL* project. These builtin implementations remain as fallbacks that are only used when OpenSSL does not provide them.

C API improvements:

- *PEP 697*, unstable C API tier
- *PEP 683*, immortal objects

CPython implementation improvements:

- *PEP 709*, comprehension inlining
- CPython support for the Linux `perf` profiler
- Implement stack overflow protection on supported platforms

New typing features:

- *PEP 692*, using `TypedDict` to annotate **kwargs
- *PEP 698*, `typing.override()` decorator

Important deprecations, removals or restrictions:

- **PEP 623**: Remove `wstr` from Unicode objects in Python's C API, reducing the size of every `str` object by at least 8 bytes.
- **PEP 632**: Remove the `distutils` package. See the migration guide for advice replacing the APIs it provided. The third-party Setuptools package continues to provide `distutils`, if you still require it in Python 3.12 and beyond.
- gh-95299: Do not pre-install `setuptools` in virtual environments created with `venv`. This means that `distutils`, `setuptools`, `pkg_resources`, and `easy_install` will no longer available by default; to access these run `pip install setuptools` in the activated virtual environment.
- The `asynchat`, `asyncore`, and `imp` modules have been removed, along with several `unittest.TestCase` *method aliases*.

2 New Features

2.1 PEP 695: Type Parameter Syntax

Generic classes and functions under **PEP 484** were declared using a verbose syntax that left the scope of type parameters unclear and required explicit declarations of variance.

PEP 695 introduces a new, more compact and explicit way to create generic classes and functions:

```
def max[T](args: Iterable[T]) -> T:
    ...

class list[T]:
    def __getitem__(self, index: int, /) -> T:
        ...

    def append(self, element: T) -> None:
        ...
```

In addition, the PEP introduces a new way to declare type aliases using the `type` statement, which creates an instance of `TypeAliasType`:

```
type Point = tuple[float, float]
```

Type aliases can also be generic:

```
type Point[T] = tuple[T, T]
```

The new syntax allows declaring `TypeVarTuple` and `ParamSpec` parameters, as well as `TypeVar` parameters with bounds or constraints:

```
type IntFunc[**P] = Callable[P, int]  # ParamSpec
type LabeledTuple[*Ts] = tuple[str, *Ts]  # TypeVarTuple
type HashableSequence[T: Hashable] = Sequence[T]  # TypeVar with bound
type IntOrStrSequence[T: (int, str)] = Sequence[T]  # TypeVar with constraints
```

The value of type aliases and the bound and constraints of type variables created through this syntax are evaluated only on demand (see lazy evaluation). This means type aliases are able to refer to other types defined later in the file.

Type parameters declared through a type parameter list are visible within the scope of the declaration and any nested scopes, but not in the outer scope. For example, they can be used in the type annotations for the methods of a generic class or in the class body. However, they cannot be used in the module scope after the class is defined. See type-params for a detailed description of the runtime semantics of type parameters.

In order to support these scoping semantics, a new kind of scope is introduced, the annotation scope. Annotation scopes behave for the most part like function scopes, but interact differently with enclosing class scopes. In Python 3.13, annotations will also be evaluated in annotation scopes.

See **PEP 695** for more details.

(PEP written by Eric Traut. Implementation by Jelle Zijlstra, Eric Traut, and others in gh-103764.)

2.2 PEP 701: Syntactic formalization of f-strings

PEP 701 lifts some restrictions on the usage of f-strings. Expression components inside f-strings can now be any valid Python expression, including strings reusing the same quote as the containing f-string, multi-line expressions, comments, backslashes, and unicode escape sequences. Let's cover these in detail:

- Quote reuse: in Python 3.11, reusing the same quotes as the enclosing f-string raises a `SyntaxError`, forcing the user to either use other available quotes (like using double quotes or triple quotes if the f-string uses single quotes). In Python 3.12, you can now do things like this:

```
>>> songs = ['Take me back to Eden', 'Alkaline', 'Ascensionism']
>>> f"This is the playlist: {", ".join(songs)}"
'This is the playlist: Take me back to Eden, Alkaline, Ascensionism'
```

  Note that before this change there was no explicit limit in how f-strings can be nested, but the fact that string quotes cannot be reused inside the expression component of f-strings made it impossible to nest f-strings arbitrarily. In fact, this is the most nested f-string that could be written:

```
>>> f"""{f'''{f'{f"{1+1}"}'}'''}"""
'2'
```

  As now f-strings can contain any valid Python expression inside expression components, it is now possible to nest f-strings arbitrarily:

```
>>> f"{f"{f"{f"{f"{f"{1+1}"}"}"}"}"}"
'2'
```

- Multi-line expressions and comments: In Python 3.11, f-string expressions must be defined in a single line, even if the expression within the f-string could normally span multiple lines (like literal lists being defined over multiple lines), making them harder to read. In Python 3.12 you can now define f-strings spanning multiple lines, and add inline comments:

```
>>> f"This is the playlist: {", ".join([
...     'Take me back to Eden',  # My, my, those eyes like fire
...     'Alkaline',              # Not acid nor alkaline
...     'Ascensionism'           # Take to the broken skies at last
... ])}"
'This is the playlist: Take me back to Eden, Alkaline, Ascensionism'
```

- Backslashes and unicode characters: before Python 3.12 f-string expressions couldn't contain any `\` character. This also affected unicode escape sequences (such as `\N{snowman}`) as these contain the `\N` part that previously could not be part of expression components of f-strings. Now, you can define expressions like this:

```
>>> print(f"This is the playlist: {"\n".join(songs)}")
This is the playlist: Take me back to Eden
Alkaline
Ascensionism
>>> print(f"This is the playlist: {"\N{BLACK HEART SUIT}".join(songs)}")
This is the playlist: Take me back to Eden♥Alkaline♥Ascensionism
```

See **PEP 701** for more details.

As a positive side-effect of how this feature has been implemented (by parsing f-strings with **the PEG parser**), now error messages for f-strings are more precise and include the exact location of the error. For example, in Python 3.11, the following f-string raises a `SyntaxError`:

```
>>> my_string = f"{x z y}" + f"{1 + 1}"
  File "<stdin>", line 1
    (x z y)
     ^^^
SyntaxError: f-string: invalid syntax. Perhaps you forgot a comma?
```

but the error message doesn't include the exact location of the error within the line and also has the expression artificially surrounded by parentheses. In Python 3.12, as f-strings are parsed with the PEG parser, error messages can be more precise and show the entire line:

```
>>> my_string = f"{x z y}" + f"{1 + 1}"
  File "<stdin>", line 1
    my_string = f"{x z y}" + f"{1 + 1}"
                   ^^^
SyntaxError: invalid syntax. Perhaps you forgot a comma?
```

(Contributed by Pablo Galindo, Batuhan Taskaya, Lysandros Nikolaou, Cristián Maureira-Fredes and Marta Gómez in gh-102856. PEP written by Pablo Galindo, Batuhan Taskaya, Lysandros Nikolaou and Marta Gómez).

2.3 PEP 684: A Per-Interpreter GIL

PEP 684 introduces a per-interpreter GIL, so that sub-interpreters may now be created with a unique GIL per interpreter. This allows Python programs to take full advantage of multiple CPU cores. This is currently only available through the C-API, though a Python API is **anticipated for 3.13**.

Use the new `Py_NewInterpreterFromConfig()` function to create an interpreter with its own GIL:

```
PyInterpreterConfig config = {
    .check_multi_interp_extensions = 1,
    .gil = PyInterpreterConfig_OWN_GIL,
};
PyThreadState *tstate = NULL;
PyStatus status = Py_NewInterpreterFromConfig(&tstate, &config);
if (PyStatus_Exception(status)) {
    return -1;
}
/* The new interpreter is now active in the current thread. */
```

For further examples how to use the C-API for sub-interpreters with a per-interpreter GIL, see Modules/_xxsubinterpretersmodule.c.

(Contributed by Eric Snow in gh-104210, etc.)

2.4 PEP 669: Low impact monitoring for CPython

PEP 669 defines a new `API` for profilers, debuggers, and other tools to monitor events in CPython. It covers a wide range of events, including calls, returns, lines, exceptions, jumps, and more. This means that you only pay for what you use, providing support for near-zero overhead debuggers and coverage tools. See `sys.monitoring` for details.

(Contributed by Mark Shannon in gh-103082.)

2.5 PEP 688: Making the buffer protocol accessible in Python

PEP 688 introduces a way to use the buffer protocol from Python code. Classes that implement the `__buffer__()` method are now usable as buffer types.

The new `collections.abc.Buffer` ABC provides a standard way to represent buffer objects, for example in type annotations. The new `inspect.BufferFlags` enum represents the flags that can be used to customize buffer creation. (Contributed by Jelle Zijlstra in gh-102500.)

2.6 PEP 709: Comprehension inlining

Dictionary, list, and set comprehensions are now inlined, rather than creating a new single-use function object for each execution of the comprehension. This speeds up execution of a comprehension by up to two times. See **PEP 709** for further details.

Comprehension iteration variables remain isolated and don't overwrite a variable of the same name in the outer scope, nor are they visible after the comprehension. Inlining does result in a few visible behavior changes:

- There is no longer a separate frame for the comprehension in tracebacks, and tracing/profiling no longer shows the comprehension as a function call.
- The `symtable` module will no longer produce child symbol tables for each comprehension; instead, the comprehension's locals will be included in the parent function's symbol table.
- Calling `locals()` inside a comprehension now includes variables from outside the comprehension, and no longer includes the synthetic `.0` variable for the comprehension "argument".
- A comprehension iterating directly over `locals()` (e.g. `[k for k in locals()]`) may see "RuntimeError: dictionary changed size during iteration" when run under tracing (e.g. code coverage measurement). This is the same behavior already seen in e.g. `for k in locals():`. To avoid the error, first create a list of keys to iterate over: `keys = list(locals()); [k for k in keys]`.

(Contributed by Carl Meyer and Vladimir Matveev in **PEP 709**.)

2.7 Improved Error Messages

- Modules from the standard library are now potentially suggested as part of the error messages displayed by the interpreter when a `NameError` is raised to the top level. (Contributed by Pablo Galindo in gh-98254.)

```
>>> sys.version_info
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'sys' is not defined. Did you forget to import 'sys'?
```

- Improve the error suggestion for `NameError` exceptions for instances. Now if a `NameError` is raised in a method and the instance has an attribute that's exactly equal to the name in the exception, the suggestion will include `self.<NAME>` instead of the closest match in the method scope. (Contributed by Pablo Galindo in gh-99139.)

```
>>> class A:
...     def __init__(self):
...         self.blech = 1
...
...     def foo(self):
...         somethin = blech
...
```

(continues on next page)

(continued from previous page)

```
>>> A().foo()
Traceback (most recent call last):
  File "<stdin>", line 1
    somethin = blech
               ^^^^^
NameError: name 'blech' is not defined. Did you mean: 'self.blech'?
```

- Improve the `SyntaxError` error message when the user types `import x from y` instead of `from y import x`. (Contributed by Pablo Galindo in gh-98931.)

```
>>> import a.y.z from b.y.z
Traceback (most recent call last):
  File "<stdin>", line 1
    import a.y.z from b.y.z
    ^^^^^^^^^^^^^^^^^^^^^^^
SyntaxError: Did you mean to use 'from ... import ...' instead?
```

- `ImportError` exceptions raised from failed `from <module> import <name>` statements now include suggestions for the value of `<name>` based on the available names in `<module>`. (Contributed by Pablo Galindo in gh-91058.)

```
>>> from collections import chainmap
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: cannot import name 'chainmap' from 'collections'. Did you mean:
↪'ChainMap'?
```

3 New Features Related to Type Hints

This section covers major changes affecting **type hints** and the `typing` module.

3.1 PEP 692: Using `TypedDict` for more precise `**kwargs` typing

Typing `**kwargs` in a function signature as introduced by **PEP 484** allowed for valid annotations only in cases where all of the `**kwargs` were of the same type.

PEP 692 specifies a more precise way of typing `**kwargs` by relying on typed dictionaries:

```
from typing import TypedDict, Unpack

class Movie(TypedDict):
  name: str
  year: int

def foo(**kwargs: Unpack[Movie]): ...
```

See **PEP 692** for more details.

(Contributed by Franek Magiera in gh-103629.)

3.2 PEP 698: Override Decorator for Static Typing

A new decorator `typing.override()` has been added to the `typing` module. It indicates to type checkers that the method is intended to override a method in a superclass. This allows type checkers to catch mistakes where a method that is intended to override something in a base class does not in fact do so.

Example:

```
from typing import override

class Base:
  def get_color(self) -> str:
    return "blue"

class GoodChild(Base):
  @override  # ok: overrides Base.get_color
  def get_color(self) -> str:
    return "yellow"

class BadChild(Base):
  @override  # type checker error: does not override Base.get_color
  def get_colour(self) -> str:
    return "red"
```

See **PEP 698** for more details.

(Contributed by Steven Troxler in gh-101561.)

4 Other Language Changes

- The parser now raises `SyntaxError` when parsing source code containing null bytes. (Contributed by Pablo Galindo in gh-96670.)
- A backslash-character pair that is not a valid escape sequence now generates a `SyntaxWarning`, instead of `DeprecationWarning`. For example, `re.compile("\d+\.\d+")` now emits a `SyntaxWarning` (`"\d"` is an invalid escape sequence, use raw strings for regular expression: `re.compile(r"\d+\.\d+")`). In a future Python version, `SyntaxError` will eventually be raised, instead of `SyntaxWarning`. (Contributed by Victor Stinner in gh-98401.)
- Octal escapes with value larger than `0o377` (ex: `"\477"`), deprecated in Python 3.11, now produce a `SyntaxWarning`, instead of `DeprecationWarning`. In a future Python version they will be eventually a `SyntaxError`. (Contributed by Victor Stinner in gh-98401.)
- Variables used in the target part of comprehensions that are not stored to can now be used in assignment expressions (`:=`). For example, in `[(b := 1) for a, b.prop in some_iter]`, the assignment to `b` is now allowed. Note that assigning to variables stored to in the target part of comprehensions (like `a`) is still disallowed, as per **PEP 572**. (Contributed by Nikita Sobolev in gh-100581.)
- Exceptions raised in a class or type's `__set_name__` method are no longer wrapped by a `RuntimeError`. Context information is added to the exception as a **PEP 678** note. (Contributed by Irit Katriel in gh-77757.)
- When a `try-except*` construct handles the entire `ExceptionGroup` and raises one other exception, that exception is no longer wrapped in an `ExceptionGroup`. Also changed in version 3.11.4. (Contributed by Irit Katriel in gh-103590.)
- The Garbage Collector now runs only on the eval breaker mechanism of the Python bytecode evaluation loop instead of object allocations. The GC can also run when `PyErr_CheckSignals()` is called so C extensions that need

to run for a long time without executing any Python code also have a chance to execute the GC periodically. (Contributed by Pablo Galindo in gh-97922.)

- All builtin and extension callables expecting boolean parameters now accept arguments of any type instead of just `bool` and `int`. (Contributed by Serhiy Storchaka in gh-60203.)
- `memoryview` now supports the half-float type (the "e" format code). (Contributed by Donghee Na and Antoine Pitrou in gh-90751.)
- `slice` objects are now hashable, allowing them to be used as dict keys and set items. (Contributed by Will Bradshaw, Furkan Onder, and Raymond Hettinger in gh-101264.)
- `sum()` now uses Neumaier summation to improve accuracy and commutativity when summing floats or mixed ints and floats. (Contributed by Raymond Hettinger in gh-100425.)
- `ast.parse()` now raises `SyntaxError` instead of `ValueError` when parsing source code containing null bytes. (Contributed by Pablo Galindo in gh-96670.)
- The extraction methods in `tarfile`, and `shutil.unpack_archive()`, have a new a *filter* argument that allows limiting tar features than may be surprising or dangerous, such as creating files outside the destination directory. See tarfile extraction filters for details. In Python 3.14, the default will switch to `'data'`. (Contributed by Petr Viktorin in **PEP 706**.)
- `types.MappingProxyType` instances are now hashable if the underlying mapping is hashable. (Contributed by Serhiy Storchaka in gh-87995.)
- Add support for the perf profiler through the new environment variable `PYTHONPERFSUPPORT` and command-line option `-X perf`, as well as the new `sys.activate_stack_trampoline()`, `sys.deactivate_stack_trampoline()`, and `sys.is_stack_trampoline_active()` functions. (Design by Pablo Galindo. Contributed by Pablo Galindo and Christian Heimes with contributions from Gregory P. Smith [Google] and Mark Shannon in gh-96123.)

5 New Modules

- None.

6 Improved Modules

6.1 array

- The `array.array` class now supports subscripting, making it a generic type. (Contributed by Jelle Zijlstra in gh-98658.)

6.2 asyncio

- The performance of writing to sockets in `asyncio` has been significantly improved. `asyncio` now avoids unnecessary copying when writing to sockets and uses `sendmsg()` if the platform supports it. (Contributed by Kumar Aditya in gh-91166.)
- Add `asyncio.eager_task_factory()` and `asyncio.create_eager_task_factory()` functions to allow opting an event loop in to eager task execution, making some use-cases 2x to 5x faster. (Contributed by Jacob Bower & Itamar Oren in gh-102853, gh-104140, and gh-104138)

- On Linux, `asyncio` uses `asyncio.PidfdChildWatcher` by default if `os.pidfd_open()` is available and functional instead of `asyncio.ThreadedChildWatcher`. (Contributed by Kumar Aditya in gh-98024.)
- The event loop now uses the best available child watcher for each platform (`asyncio.PidfdChildWatcher` if supported and `asyncio.ThreadedChildWatcher` otherwise), so manually configuring a child watcher is not recommended. (Contributed by Kumar Aditya in gh-94597.)
- Add *loop_factory* parameter to `asyncio.run()` to allow specifying a custom event loop factory. (Contributed by Kumar Aditya in gh-99388.)
- Add C implementation of `asyncio.current_task()` for 4x-6x speedup. (Contributed by Itamar Oren and Pranav Thulasiram Bhat in gh-100344.)
- `asyncio.iscoroutine()` now returns `False` for generators as `asyncio` does not support legacy generator-based coroutines. (Contributed by Kumar Aditya in gh-102748.)
- `asyncio.wait()` and `asyncio.as_completed()` now accepts generators yielding tasks. (Contributed by Kumar Aditya in gh-78530.)

6.3 calendar

- Add enums `calendar.Month` and `calendar.Day` defining months of the year and days of the week. (Contributed by Prince Roshan in gh-103636.)

6.4 csv

- Add `csv.QUOTE_NOTNULL` and `csv.QUOTE_STRINGS` flags to provide finer grained control of `None` and empty strings by `csv.writer` objects.

6.5 dis

- Pseudo instruction opcodes (which are used by the compiler but do not appear in executable bytecode) are now exposed in the `dis` module. `HAVE_ARGUMENT` is still relevant to real opcodes, but it is not useful for pseudo instructions. Use the new `dis.hasarg` collection instead. (Contributed by Irit Katriel in gh-94216.)
- Add the `dis.hasexc` collection to signify instructions that set an exception handler. (Contributed by Irit Katriel in gh-94216.)

6.6 fractions

- Objects of type `fractions.Fraction` now support float-style formatting. (Contributed by Mark Dickinson in gh-100161.)

6.7 importlib.resources

- `importlib.resources.as_file()` now supports resource directories. (Contributed by Jason R. Coombs in gh-97930.)
- Rename first parameter of `importlib.resources.files()` to *anchor*. (Contributed by Jason R. Coombs in gh-100598.)

6.8 inspect

- Add `inspect.markcoroutinefunction()` to mark sync functions that return a coroutine for use with `inspect.iscoroutinefunction()`. (Contributed by Carlton Gibson in gh-99247.)
- Add `inspect.getasyncgenstate()` and `inspect.getasyncgenlocals()` for determining the current state of asynchronous generators. (Contributed by Thomas Krennwallner in gh-79940.)
- The performance of `inspect.getattr_static()` has been considerably improved. Most calls to the function should be at least 2x faster than they were in Python 3.11. (Contributed by Alex Waygood in gh-103193.)

6.9 itertools

- Add `itertools.batched()` for collecting into even-sized tuples where the last batch may be shorter than the rest. (Contributed by Raymond Hettinger in gh-98363.)

6.10 math

- Add `math.sumprod()` for computing a sum of products. (Contributed by Raymond Hettinger in gh-100485.)
- Extend `math.nextafter()` to include a *steps* argument for moving up or down multiple steps at a time. (Contributed by Matthias Goergens, Mark Dickinson, and Raymond Hettinger in gh-94906.)

6.11 os

- Add `os.PIDFD_NONBLOCK` to open a file descriptor for a process with `os.pidfd_open()` in non-blocking mode. (Contributed by Kumar Aditya in gh-93312.)
- `os.DirEntry` now includes an `os.DirEntry.is_junction()` method to check if the entry is a junction. (Contributed by Charles Machalow in gh-99547.)
- Add `os.listdrives()`, `os.listvolumes()` and `os.listmounts()` functions on Windows for enumerating drives, volumes and mount points. (Contributed by Steve Dower in gh-102519.)
- `os.stat()` and `os.lstat()` are now more accurate on Windows. The `st_birthtime` field will now be filled with the creation time of the file, and `st_ctime` is deprecated but still contains the creation time (but in the future will return the last metadata change, for consistency with other platforms). `st_dev` may be up to 64 bits and `st_ino` up to 128 bits depending on your file system, and `st_rdev` is always set to zero rather than incorrect values. Both functions may be significantly faster on newer releases of Windows. (Contributed by Steve Dower in gh-99726.)
- As of 3.12.4, `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.)

6.12 os.path

- Add `os.path.isjunction()` to check if a given path is a junction. (Contributed by Charles Machalow in gh-99547.)
- Add `os.path.splitroot()` to split a path into a triad `(drive, root, tail)`. (Contributed by Barney Gale in gh-101000.)

6.13 pathlib

- Add support for subclassing `pathlib.PurePath` and `pathlib.Path`, plus their Posix- and Windows-specific variants. Subclasses may override the `pathlib.PurePath.with_segments()` method to pass information between path instances.
- Add `pathlib.Path.walk()` for walking the directory trees and generating all file or directory names within them, similar to `os.walk()`. (Contributed by Stanislav Zmiev in gh-90385.)
- Add *walk_up* optional parameter to `pathlib.PurePath.relative_to()` to allow the insertion of `..` entries in the result; this behavior is more consistent with `os.path.relpath()`. (Contributed by Domenico Ragusa in gh-84538.)
- Add `pathlib.Path.is_junction()` as a proxy to `os.path.isjunction()`. (Contributed by Charles Machalow in gh-99547.)
- Add *case_sensitive* optional parameter to `pathlib.Path.glob()`, `pathlib.Path.rglob()` and `pathlib.PurePath.match()` for matching the path's case sensitivity, allowing for more precise control over the matching process.

6.14 pdb

- Add convenience variables to hold values temporarily for debug session and provide quick access to values like the current frame or the return value. (Contributed by Tian Gao in gh-103693.)

6.15 random

- Add `random.binomialvariate()`. (Contributed by Raymond Hettinger in gh-81620.)
- Add a default of `lambd=1.0` to `random.expovariate()`. (Contributed by Raymond Hettinger in gh-100234.)

6.16 shutil

- `shutil.make_archive()` now passes the *root_dir* argument to custom archivers which support it. In this case it no longer temporarily changes the current working directory of the process to *root_dir* to perform archiving. (Contributed by Serhiy Storchaka in gh-74696.)
- `shutil.rmtree()` now accepts a new argument *onexc* which is an error handler like *onerror* but which expects an exception instance rather than a *(typ, val, tb)* triplet. *onerror* is deprecated. (Contributed by Irit Katriel in gh-102828.)
- `shutil.which()` now consults the *PATHEXT* environment variable to find matches within *PATH* on Windows even when the given *cmd* includes a directory component. (Contributed by Charles Machalow in gh-103179.)

`shutil.which()` will call `NeedCurrentDirectoryForExePathW` when querying for executables on Windows to determine if the current working directory should be prepended to the search path. (Contributed by Charles Machalow in gh-103179.)

`shutil.which()` will return a path matching the *cmd* with a component from `PATHEXT` prior to a direct match elsewhere in the search path on Windows. (Contributed by Charles Machalow in gh-103179.)

6.17 sqlite3

- Add a command-line interface. (Contributed by Erlend E. Aasland in gh-77617.)
- Add the `sqlite3.Connection.autocommit` attribute to `sqlite3.Connection` and the *autocommit* parameter to `sqlite3.connect()` to control **PEP 249**-compliant transaction handling. (Contributed by Erlend E. Aasland in gh-83638.)
- Add *entrypoint* keyword-only parameter to `sqlite3.Connection.load_extension()`, for overriding the SQLite extension entry point. (Contributed by Erlend E. Aasland in gh-103015.)
- Add `sqlite3.Connection.getconfig()` and `sqlite3.Connection.setconfig()` to `sqlite3.Connection` to make configuration changes to a database connection. (Contributed by Erlend E. Aasland in gh-103489.)

6.18 statistics

- Extend `statistics.correlation()` to include as a `ranked` method for computing the Spearman correlation of ranked data. (Contributed by Raymond Hettinger in gh-95861.)

6.19 sys

- Add the `sys.monitoring` namespace to expose the new *PEP 669* monitoring API. (Contributed by Mark Shannon in gh-103082.)
- Add `sys.activate_stack_trampoline()` and `sys.deactivate_stack_trampoline()` for activating and deactivating stack profiler trampolines, and `sys.is_stack_trampoline_active()` for querying if stack profiler trampolines are active. (Contributed by Pablo Galindo and Christian Heimes with contributions from Gregory P. Smith [Google] and Mark Shannon in gh-96123.)
- Add `sys.last_exc` which holds the last unhandled exception that was raised (for post-mortem debugging use cases). Deprecate the three fields that have the same information in its legacy form: `sys.last_type`, `sys.last_value` and `sys.last_traceback`. (Contributed by Irit Katriel in gh-102778.)
- `sys._current_exceptions()` now returns a mapping from thread-id to an exception instance, rather than to a `(typ, exc, tb)` tuple. (Contributed by Irit Katriel in gh-103176.)
- `sys.setrecursionlimit()` and `sys.getrecursionlimit()`. The recursion limit now applies only to Python code. Builtin functions do not use the recursion limit, but are protected by a different mechanism that prevents recursion from causing a virtual machine crash.

6.20 tempfile

- The `tempfile.NamedTemporaryFile` function has a new optional parameter *delete_on_close* (Contributed by Evgeny Zorin in gh-58451.)
- `tempfile.mkdtemp()` now always returns an absolute path, even if the argument provided to the *dir* parameter is a relative path.
- As of 3.12.4 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.)

6.21 threading

- Add `threading.settrace_all_threads()` and `threading.setprofile_all_threads()` that allow to set tracing and profiling functions in all running threads in addition to the calling one. (Contributed by Pablo Galindo in gh-93503.)

6.22 tkinter

- `tkinter.Canvas.coords()` now flattens its arguments. It now accepts not only coordinates as separate arguments (`x1, y1, x2, y2, ...`) and a sequence of coordinates (`[x1, y1, x2, y2, ...]`), but also coordinates grouped in pairs (`(x1, y1), (x2, y2), ...` and `[(x1, y1), (x2, y2), ...]`), like `create_*()` methods. (Contributed by Serhiy Storchaka in gh-94473.)

6.23 tokenize

- The `tokenize` module includes the changes introduced in **PEP 701**. (Contributed by Marta Gómez Macías and Pablo Galindo in gh-102856.) See *Porting to Python 3.12* for more information on the changes to the `tokenize` module.

6.24 types

- Add `types.get_original_bases()` to allow for further introspection of user-defined-generics when subclassed. (Contributed by James Hilton-Balfe and Alex Waygood in gh-101827.)

6.25 typing

- `isinstance()` checks against `runtime-checkable protocols` now use `inspect.getattr_static()` rather than `hasattr()` to lookup whether attributes exist. This means that descriptors and `__getattr__()` methods are no longer unexpectedly evaluated during `isinstance()` checks against runtime-checkable protocols. However, it may also mean that some objects which used to be considered instances of a runtime-checkable protocol may no longer be considered instances of that protocol on Python 3.12+, and vice versa. Most users are unlikely to be affected by this change. (Contributed by Alex Waygood in gh-102433.)
- The members of a runtime-checkable protocol are now considered "frozen" at runtime as soon as the class has been created. Monkey-patching attributes onto a runtime-checkable protocol will still work, but will have no impact on `isinstance()` checks comparing objects to the protocol. For example:

```
>>> from typing import Protocol, runtime_checkable
>>> @runtime_checkable
... class HasX(Protocol):
...     x = 1
...
>>> class Foo: ...
...
>>> f = Foo()
>>> isinstance(f, HasX)
False
>>> f.x = 1
>>> isinstance(f, HasX)
True
>>> HasX.y = 2
>>> isinstance(f, HasX)  # unchanged, even though HasX now also has a "y"␣
↪attribute
True
```

This change was made in order to speed up `isinstance()` checks against runtime-checkable protocols.

- The performance profile of `isinstance()` checks against `runtime-checkable protocols` has changed significantly. Most `isinstance()` checks against protocols with only a few members should be at least 2x faster than in 3.11, and some may be 20x faster or more. However, `isinstance()` checks against protocols with many members may be slower than in Python 3.11. (Contributed by Alex Waygood in gh-74690 and gh-103193.)
- All `typing.TypedDict` and `typing.NamedTuple` classes now have the `__orig_bases__` attribute. (Contributed by Adrian Garcia Badaracco in gh-103699.)
- Add `frozen_default` parameter to `typing.dataclass_transform()`. (Contributed by Erik De Bonte in gh-99957.)

6.26 unicodedata

- The Unicode database has been updated to version 15.0.0. (Contributed by Benjamin Peterson in gh-96734).

6.27 unittest

Add a `--durations` command line option, showing the N slowest test cases:

```
python3 -m unittest --durations=3 lib.tests.test_threading
.....
Slowest test durations
----------------------------------------------------------------------
1.210s     test_timeout (Lib.test.test_threading.BarrierTests)
1.003s     test_default_timeout (Lib.test.test_threading.BarrierTests)
0.518s     test_timeout (Lib.test.test_threading.EventTests)

(0.000 durations hidden.  Use -v to show these durations.)
----------------------------------------------------------------------
Ran 158 tests in 9.869s

OK (skipped=3)
```

(Contributed by Giampaolo Rodola in gh-48330)

6.28 uuid

- Add a command-line interface. (Contributed by Adam Chhina in gh-88597.)

7 Optimizations

- Remove `wstr` and `wstr_length` members from Unicode objects. It reduces object size by 8 or 16 bytes on 64bit platform. (**PEP 623**) (Contributed by Inada Naoki in gh-92536.)
- Add experimental support for using the BOLT binary optimizer in the build process, which improves performance by 1-5%. (Contributed by Kevin Modzelewski in gh-90536 and tuned by Donghee Na in gh-101525)
- Speed up the regular expression substitution (functions `re.sub()` and `re.subn()` and corresponding `re.Pattern` methods) for replacement strings containing group references by 2–3 times. (Contributed by Serhiy Storchaka in gh-91524.)
- Speed up `asyncio.Task` creation by deferring expensive string formatting. (Contributed by Itamar Oren in gh-103793.)
- The `tokenize.tokenize()` and `tokenize.generate_tokens()` functions are up to 64% faster as a side effect of the changes required to cover **PEP 701** in the `tokenize` module. (Contributed by Marta Gómez Macías and Pablo Galindo in gh-102856.)
- Speed up `super()` method calls and attribute loads via the new `LOAD_SUPER_ATTR` instruction. (Contributed by Carl Meyer and Vladimir Matveev in gh-103497.)

8 CPython bytecode changes

- Remove the `LOAD_METHOD` instruction. It has been merged into `LOAD_ATTR`. `LOAD_ATTR` will now behave like the old `LOAD_METHOD` instruction if the low bit of its oparg is set. (Contributed by Ken Jin in gh-93429.)
- Remove the `JUMP_IF_FALSE_OR_POP` and `JUMP_IF_TRUE_OR_POP` instructions. (Contributed by Irit Katriel in gh-102859.)
- Remove the `PRECALL` instruction. (Contributed by Mark Shannon in gh-92925.)
- Add the `BINARY_SLICE` and `STORE_SLICE` instructions. (Contributed by Mark Shannon in gh-94163.)
- Add the `CALL_INTRINSIC_1` instructions. (Contributed by Mark Shannon in gh-99005.)
- Add the `CALL_INTRINSIC_2` instruction. (Contributed by Irit Katriel in gh-101799.)
- Add the `CLEANUP_THROW` instruction. (Contributed by Brandt Bucher in gh-90997.)
- Add the `END_SEND` instruction. (Contributed by Mark Shannon in gh-103082.)
- Add the `LOAD_FAST_AND_CLEAR` instruction as part of the implementation of **PEP 709**. (Contributed by Carl Meyer in gh-101441.)
- Add the `LOAD_FAST_CHECK` instruction. (Contributed by Dennis Sweeney in gh-93143.)
- Add the `LOAD_FROM_DICT_OR_DEREF`, `LOAD_FROM_DICT_OR_GLOBALS`, and `LOAD_LOCALS` opcodes as part of the implementation of **PEP 695**. Remove the `LOAD_CLASSDEREF` opcode, which can be replaced with `LOAD_LOCALS` plus `LOAD_FROM_DICT_OR_DEREF`. (Contributed by Jelle Zijlstra in gh-103764.)
- Add the `LOAD_SUPER_ATTR` instruction. (Contributed by Carl Meyer and Vladimir Matveev in gh-103497.)
- Add the `RETURN_CONST` instruction. (Contributed by Wenyang Wang in gh-101632.)

9 Demos and Tools

- Remove the `Tools/demo/` directory which contained old demo scripts. A copy can be found in the old-demos project. (Contributed by Victor Stinner in gh-97681.)
- Remove outdated example scripts of the `Tools/scripts/` directory. A copy can be found in the old-demos project. (Contributed by Victor Stinner in gh-97669.)

10 Deprecated

- `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 `ast` 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.)
- `asyncio`:
  - The child watcher classes `asyncio.MultiLoopChildWatcher`, `asyncio.FastChildWatcher`, `asyncio.AbstractChildWatcher` and `asyncio.SafeChildWatcher` are deprecated and will be removed in Python 3.14. (Contributed by Kumar Aditya in gh-94597.)
  - `asyncio.set_child_watcher()`, `asyncio.get_child_watcher()`, `asyncio.AbstractEventLoopPolicy.set_child_watcher()` and `asyncio.AbstractEventLoopPolicy.get_child_watcher()` are deprecated and will be removed in Python 3.14. (Contributed by Kumar Aditya in gh-94597.)
  - The `get_event_loop()` method of the default event loop policy now emits a `DeprecationWarning` if there is no current event loop set and it decides to create one. (Contributed by Serhiy Storchaka and Guido van Rossum in gh-100160.)
- `calendar`: `calendar.January` and `calendar.February` constants are deprecated and replaced by `calendar.JANUARY` and `calendar.FEBRUARY`. (Contributed by Prince Roshan in gh-103636.)
- `collections.abc`: Deprecated `collections.abc.ByteString`. Prefer `Sequence` or `collections.abc.Buffer`. For use in typing, prefer a union, like `bytes | bytearray`, or `collections.abc.Buffer`. (Contributed by Shantanu Jain in gh-91896.)
- `datetime`: `datetime.datetime`'s `utcnow()` and `utcfromtimestamp()` are deprecated and will be removed in a future version. Instead, use timezone-aware objects to represent datetimes in UTC: respectively, call `now()` and `fromtimestamp()` with the *tz* parameter set to `datetime.UTC`. (Contributed by Paul Ganssle in gh-103857.)
- `email`: Deprecate the *isdst* parameter in `email.utils.localtime()`. (Contributed by Alan Williams in gh-72346.)

- `importlib.abc`: Deprecated the following classes, scheduled for removal in Python 3.14:
  - `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`: Deprecate the support for copy, deepcopy, and pickle operations, which is undocumented, inefficient, historically buggy, and inconsistent. 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`: In Python 3.14, the default `multiprocessing` 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 contexts and start methods.
- `pkgutil`: `pkgutil.find_loader()` and `pkgutil.get_loader()` are deprecated and will be removed in Python 3.14; use `importlib.util.find_spec()` instead. (Contributed by Nikita Sobolev in gh-97850.)
- `pty`: The module has two undocumented `master_open()` and `slave_open()` functions that have been deprecated since Python 2 but only gained a proper `DeprecationWarning` in 3.12. Remove them in 3.14. (Contributed by Soumendra Ganguly and Gregory P. Smith in gh-85984.)
- `os`:
  - The `st_ctime` fields return by `os.stat()` and `os.lstat()` on Windows are deprecated. In a future release, they will contain the last metadata change time, consistent with other platforms. For now, they still contain the creation time, which is also available in the new `st_birthtime` field. (Contributed by Steve Dower in gh-99726.)
  - On POSIX platforms, `os.fork()` can now raise a `DeprecationWarning` when it can detect being called from a multithreaded process. There has always been a fundamental incompatibility with the POSIX platform when doing so. Even if such code *appeared* to work. We added the warning to raise awareness as issues encountered by code doing this are becoming more frequent. See the `os.fork()` documentation for more details along with this discussion on fork being incompatible with threads for *why* we're now surfacing this longstanding platform compatibility problem to developers.

  When this warning appears due to usage of `multiprocessing` or `concurrent.futures` the fix is to use a different `multiprocessing` start method such as `"spawn"` or `"forkserver"`.
- `shutil`: The *onerror* argument of `shutil.rmtree()` is deprecated; use *onexc* instead. (Contributed by Irit Katriel in gh-102828.)
- `sqlite3`:
  - default adapters and converters are now deprecated. Instead, use the sqlite3-adapter-converter-recipes and tailor them to your needs. (Contributed by Erlend E. Aasland in gh-90016.)
  - In `execute()`, `DeprecationWarning` is now emitted when named placeholders are used together with parameters supplied as a sequence instead of as a `dict`. Starting from Python 3.14, using named placeholders with parameters supplied as a sequence will raise a `ProgrammingError`. (Contributed by Erlend E. Aasland in gh-101698.)

- `sys`: The `sys.last_type`, `sys.last_value` and `sys.last_traceback` fields are deprecated. Use `sys.last_exc` instead. (Contributed by Irit Katriel in gh-102778.)
- `tarfile`: Extracting tar archives without specifying *filter* is deprecated until Python 3.14, when `'data'` filter will become the default. See tarfile-extraction-filter for details.
- `typing`:
  - `typing.Hashable` and `typing.Sized`, aliases for `collections.abc.Hashable` and `collections.abc.Sized` respectively, are deprecated. (gh-94309.)
  - `typing.ByteString`, deprecated since Python 3.9, now causes a `DeprecationWarning` to be emitted when it is used. (Contributed by Alex Waygood in gh-91896.)
- `xml.etree.ElementTree`: The module now emits `DeprecationWarning` when testing the truth value of an `xml.etree.ElementTree.Element`. Before, the Python implementation emitted `FutureWarning`, and the C implementation emitted nothing. (Contributed by Jacob Walls in gh-83122.)
- The 3-arg signatures (type, value, traceback) of `coroutine throw()`, `generator throw()` and `async generator throw()` are deprecated and may be removed in a future version of Python. Use the single-arg versions of these functions instead. (Contributed by Ofey Chan in gh-89874.)
- `DeprecationWarning` is now raised when `__package__` on a module differs from `__spec__.parent` (previously it was `ImportWarning`). (Contributed by Brett Cannon in gh-65961.)
- Setting `__package__` or `__cached__` on a module is deprecated, and will cease to be set or taken into consideration by the import system in Python 3.14. (Contributed by Brett Cannon in gh-65961.)
- The bitwise inversion operator (`~`) on bool is deprecated. It will throw an error in Python 3.14. Use `not` for logical negation of bools instead. In the rare case that you really need the bitwise inversion of the underlying `int`, convert to int explicitly: `~int(x)`. (Contributed by Tim Hoffmann in gh-103487.)
- Accessing `co_lnotab` on code objects was deprecated in Python 3.10 via **PEP 626**, but it only got a proper `DeprecationWarning` in 3.12, therefore it will be removed in 3.14. (Contributed by Nikita Sobolev in gh-101866.)

10.1 Pending Removal in Python 3.13

The following modules and APIs have been deprecated in earlier Python releases, and will be removed in Python 3.13.

Modules (see **PEP 594**):

- `aifc`
- `audioop`
- `cgi`
- `cgitb`
- `chunk`
- `crypt`
- `imghdr`
- `mailcap`
- `msilib`
- `nis`
- `nntplib`

- `ossaudiodev`
- `pipes`
- `sndhdr`
- `spwd`
- `sunau`
- `telnetlib`
- `uu`
- `xdrlib`

Other modules:

- `lib2to3`, and the **`2to3`** program (gh-84540)

APIs:

- `configparser.LegacyInterpolation` (gh-90765)
- `locale.resetlocale()` (gh-90817)
- `turtle.RawTurtle.settiltangle()` (gh-50096)
- `unittest.findTestCases()` (gh-50096)
- `unittest.getTestCaseNames()` (gh-50096)
- `unittest.makeSuite()` (gh-50096)
- `unittest.TestProgram.usageExit()` (gh-67048)
- `webbrowser.MacOSX` (gh-86421)
- `classmethod` descriptor chaining (gh-89519)
- `importlib.resources` deprecated methods:
  - `contents()`
  - `is_resource()`
  - `open_binary()`
  - `open_text()`
  - `path()`
  - `read_binary()`
  - `read_text()`

  Use `importlib.resources.files()` instead. Refer to importlib-resources: Migrating from Legacy (gh-106531)

10.2 Pending Removal in Python 3.14

The following APIs have been deprecated and will be removed in Python 3.14.

- `argparse`: The *type*, *choices*, and *metavar* parameters of `argparse.BooleanOptionalAction`
- `ast`:
  - `ast.Num`
  - `ast.Str`
  - `ast.Bytes`
  - `ast.NameConstant`
  - `ast.Ellipsis`
- `asyncio`:
  - `asyncio.MultiLoopChildWatcher`
  - `asyncio.FastChildWatcher`
  - `asyncio.AbstractChildWatcher`
  - `asyncio.SafeChildWatcher`
  - `asyncio.set_child_watcher()`
  - `asyncio.get_child_watcher()`,
  - `asyncio.AbstractEventLoopPolicy.set_child_watcher()`
  - `asyncio.AbstractEventLoopPolicy.get_child_watcher()`
- `collections.abc`: `collections.abc.ByteString`.
- `email`: the *isdst* parameter in `email.utils.localtime()`.
- `importlib.abc`:
  - `importlib.abc.ResourceReader`
  - `importlib.abc.Traversable`
  - `importlib.abc.TraversableResources`
- `itertools`: Support for copy, deepcopy, and pickle operations.
- `pkgutil`:
  - `pkgutil.find_loader()`
  - `pkgutil.get_loader()`.
- `pty`:
  - `pty.master_open()`
  - `pty.slave_open()`
- `shutil`: The *onerror* argument of `shutil.rmtree()`
- `typing`: `typing.ByteString`
- `xml.etree.ElementTree`: Testing the truth value of an `xml.etree.ElementTree.Element`.
- The `__package__` and `__cached__` attributes on module objects.
- The `co_lnotab` attribute of code objects.

10.3 Pending Removal in Python 3.15

The following APIs have been deprecated and will be removed in Python 3.15.

APIs:

- `locale.getdefaultlocale()` (gh-90817)

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.

- `array`'s `'u'` format code (gh-57281)
- `typing.Text` (gh-92332)
- 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)

11 Removed

11.1 asynchat and asyncore

- These two modules have been removed according to the schedule in **PEP 594**, having been deprecated in Python 3.6. Use `asyncio` instead. (Contributed by Nikita Sobolev in gh-96580.)

11.2 configparser

- Several names deprecated in the `configparser` way back in 3.2 have been removed per gh-89336:
  - `configparser.ParsingError` no longer has a `filename` attribute or argument. Use the `source` attribute and argument instead.
  - `configparser` no longer has a `SafeConfigParser` class. Use the shorter `ConfigParser` name instead.
  - `configparser.ConfigParser` no longer has a `readfp` method. Use `read_file()` instead.

11.3 distutils

- Remove the `distutils` package. It was deprecated in Python 3.10 by **PEP 632** "Deprecate distutils module". For projects still using `distutils` and cannot be updated to something else, the `setuptools` project can be installed: it still provides `distutils`. (Contributed by Victor Stinner in gh-92584.)

11.4 ensurepip

- Remove the bundled setuptools wheel from `ensurepip`, and stop installing setuptools in environments created by `venv`.

  `pip (>= 22.1)` does not require setuptools to be installed in the environment. `setuptools`-based (and `distutils`-based) packages can still be used with `pip install`, since pip will provide `setuptools` in the build environment it uses for building a package.

  `easy_install`, `pkg_resources`, `setuptools` and `distutils` are no longer provided by default in environments created with `venv` or bootstrapped with `ensurepip`, since they are part of the `setuptools` package. For projects relying on these at runtime, the `setuptools` project should be declared as a dependency and installed separately (typically, using pip).

  (Contributed by Pradyun Gedam in gh-95299.)

11.5 enum

- Remove `enum`'s `EnumMeta.__getattr__`, which is no longer needed for enum attribute access. (Contributed by Ethan Furman in gh-95083.)

11.6 ftplib

- Remove `ftplib`'s `FTP_TLS.ssl_version` class attribute: use the *context* parameter instead. (Contributed by Victor Stinner in gh-94172.)

11.7 gzip

- Remove the `filename` attribute of `gzip`'s `gzip.GzipFile`, deprecated since Python 2.6, use the `name` attribute instead. In write mode, the `filename` attribute added `'.gz'` file extension if it was not present. (Contributed by Victor Stinner in gh-94196.)

11.8 hashlib

- Remove the pure Python implementation of `hashlib`'s `hashlib.pbkdf2_hmac()`, deprecated in Python 3.10. Python 3.10 and newer requires OpenSSL 1.1.1 (**PEP 644**): this OpenSSL version provides a C implementation of `pbkdf2_hmac()` which is faster. (Contributed by Victor Stinner in gh-94199.)

11.9 importlib

- Many previously deprecated cleanups in `importlib` have now been completed:
  - References to, and support for `module_repr()` has been removed. (Contributed by Barry Warsaw in gh-97850.)
  - `importlib.util.set_package`, `importlib.util.set_loader` and `importlib.util.module_for_loader` have all been removed. (Contributed by Brett Cannon and Nikita Sobolev in gh-65961 and gh-97850.)
  - Support for `find_loader()` and `find_module()` APIs have been removed. (Contributed by Barry Warsaw in gh-98040.)

- `importlib.abc.Finder`, `pkgutil.ImpImporter`, and `pkgutil.ImpLoader` have been removed. (Contributed by Barry Warsaw in gh-98040.)

11.10 imp

- The `imp` module has been removed. (Contributed by Barry Warsaw in gh-98040.)

  To migrate, consult the following correspondence table:

imp	importlib
`imp.NullImporter`	Insert `None` into `sys.path_importer_cache`
`imp.cache_from_source`	`importlib.util.cache_from_source()`
`imp.find_module`	`importlib.util.find_spec()`
`imp.get_magic()`	`importlib.util.MAGIC_NUMBER`
`imp.get_suffixes`	`importlib.machinery.SOURCE_SUFFIXES`, `importlib.machinery.EXTENSION_SUFFIXES`, and `importlib.machinery.BYTECODE_SUFFIXES`
`imp.get_tag()`	`sys.implementation.cache_tag`
`imp.load_module`	`importlib.import_module()`
`imp.new_module(r`	`types.ModuleType(name)`
`imp.reload()`	`importlib.reload()`
`imp.source_from_`	`importlib.util.source_from_cache()`
`imp.load_source`	*See below*

  Replace `imp.load_source()` with:

```
import importlib.util
import importlib.machinery

def load_source(modname, filename):
    loader = importlib.machinery.SourceFileLoader(modname, filename)
    spec = importlib.util.spec_from_file_location(modname, filename,␣
↪loader=loader)
    module = importlib.util.module_from_spec(spec)
    # The module is always executed and not cached in sys.modules.
    # Uncomment the following line to cache the module.
    # sys.modules[module.__name__] = module
    loader.exec_module(module)
    return module
```

- Remove `imp` functions and attributes with no replacements:
  - Undocumented functions:

    * `imp.init_builtin()`
    * `imp.load_compiled()`
    * `imp.load_dynamic()`
    * `imp.load_package()`
  - `imp.lock_held()`, `imp.acquire_lock()`, `imp.release_lock()`: the locking scheme has changed in Python 3.3 to per-module locks.
  - `imp.find_module()` constants: `SEARCH_ERROR`, `PY_SOURCE`, `PY_COMPILED`, `C_EXTENSION`, `PY_RESOURCE`, `PKG_DIRECTORY`, `C_BUILTIN`, `PY_FROZEN`, `PY_CODERESOURCE`, `IMP_HOOK`.

11.11 io

- Remove `io`'s `io.OpenWrapper` and `_pyio.OpenWrapper`, deprecated in Python 3.10: just use `open()` instead. The `open()` (`io.open()`) function is a built-in function. Since Python 3.10, `_pyio.open()` is also a static method. (Contributed by Victor Stinner in gh-94169.)

11.12 locale

- Remove `locale`'s `locale.format()` function, deprecated in Python 3.7: use `locale.format_string()` instead. (Contributed by Victor Stinner in gh-94226.)

11.13 smtpd

- The `smtpd` module has been removed according to the schedule in **PEP 594**, having been deprecated in Python 3.4.7 and 3.5.4. Use the aiosmtpd PyPI module or any other `asyncio`-based server instead. (Contributed by Oleg Iarygin in gh-93243.)

11.14 sqlite3

- The following undocumented `sqlite3` features, deprecated in Python 3.10, are now removed:
  - `sqlite3.enable_shared_cache()`
  - `sqlite3.OptimizedUnicode`

  If a shared cache must be used, open the database in URI mode using the `cache=shared` query parameter.

  The `sqlite3.OptimizedUnicode` text factory has been an alias for `str` since Python 3.3. Code that previously set the text factory to `OptimizedUnicode` can either use `str` explicitly, or rely on the default value which is also `str`.

  (Contributed by Erlend E. Aasland in gh-92548.)

11.15 ssl

- Remove ssl's `ssl.RAND_pseudo_bytes()` function, deprecated in Python 3.6: use `os.urandom()` or `ssl.RAND_bytes()` instead. (Contributed by Victor Stinner in gh-94199.)
- Remove the `ssl.match_hostname()` function. It was deprecated in Python 3.7. OpenSSL performs hostname matching since Python 3.7, Python no longer uses the `ssl.match_hostname()` function. (Contributed by Victor Stinner in gh-94199.)
- Remove the `ssl.wrap_socket()` function, deprecated in Python 3.7: instead, create a `ssl.SSLContext` object and call its `ssl.SSLContext.wrap_socket` method. Any package that still uses `ssl.wrap_socket()` is broken and insecure. The function neither sends a SNI TLS extension nor validates the server hostname. Code is subject to CWE-295 (Improper Certificate Validation). (Contributed by Victor Stinner in gh-94199.)

11.16 unittest

- Remove many long-deprecated `unittest` features:
  - A number of `TestCase` method aliases:

Deprecated alias	Method Name	Deprecated in
`failUnless`	`assertTrue()`	3.1
`failIf`	`assertFalse()`	3.1
`failUnlessEqual`	`assertEqual()`	3.1
`failIfEqual`	`assertNotEqual()`	3.1
`failUnlessAlmostEqual`	`assertAlmostEqual()`	3.1
`failIfAlmostEqual`	`assertNotAlmostEqual()`	3.1
`failUnlessRaises`	`assertRaises()`	3.1
`assert_`	`assertTrue()`	3.2
`assertEquals`	`assertEqual()`	3.2
`assertNotEquals`	`assertNotEqual()`	3.2
`assertAlmostEquals`	`assertAlmostEqual()`	3.2
`assertNotAlmostEquals`	`assertNotAlmostEqual()`	3.2
`assertRegexpMatches`	`assertRegex()`	3.2
`assertRaisesRegexp`	`assertRaisesRegex()`	3.2
`assertNotRegexpMatches`	`assertNotRegex()`	3.5

    You can use https://github.com/isidentical/teyit to automatically modernise your unit tests.
  - Undocumented and broken `TestCase` method `assertDictContainsSubset` (deprecated in Python 3.2).
  - Undocumented `TestLoader.loadTestsFromModule` parameter *use_load_tests* (deprecated and ignored since Python 3.5).
  - An alias of the `TextTestResult` class: `_TextTestResult` (deprecated in Python 3.2).

  (Contributed by Serhiy Storchaka in gh-89325.)

11.17 webbrowser

- Remove support for obsolete browsers from `webbrowser`. The removed browsers include: Grail, Mosaic, Netscape, Galeon, Skipstone, Iceape, Firebird, and Firefox versions 35 and below (gh-102871).

11.18 xml.etree.ElementTree

- Remove the `ElementTree.Element.copy()` method of the pure Python implementation, deprecated in Python 3.10, use the `copy.copy()` function instead. The C implementation of `xml.etree.ElementTree` has no `copy()` method, only a `__copy__()` method. (Contributed by Victor Stinner in gh-94383.)

11.19 zipimport

- Remove `zipimport`'s `find_loader()` and `find_module()` methods, deprecated in Python 3.10: use the `find_spec()` method instead. See **PEP 451** for the rationale. (Contributed by Victor Stinner in gh-94379.)

11.20 Others

- Remove the `suspicious` rule from the documentation `Makefile` and `Doc/tools/rstlint.py`, both in favor of sphinx-lint. (Contributed by Julien Palard in gh-98179.)
- Remove the *keyfile* and *certfile* parameters from the `ftplib`, `imaplib`, `poplib` and `smtplib` modules, and the *key_file*, *cert_file* and *check_hostname* parameters from the `http.client` module, all deprecated since Python 3.6. Use the *context* parameter (*ssl_context* in `imaplib`) instead. (Contributed by Victor Stinner in gh-94172.)
- Remove `Jython` compatibility hacks from several stdlib modules and tests. (Contributed by Nikita Sobolev in gh-99482.)
- Remove `_use_broken_old_ctypes_structure_semantics_` flag from `ctypes` module. (Contributed by Nikita Sobolev in gh-99285.)

12 Porting to Python 3.12

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

12.1 Changes in the Python API

- 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.)
- Remove `randrange()` functionality deprecated since Python 3.10. Formerly, `randrange(10.0)` losslessly converted to `randrange(10)`. Now, it raises a `TypeError`. Also, the exception raised for non-integer values such as `randrange(10.5)` or `randrange('10')` has been changed from `ValueError` to `TypeError`. This also prevents bugs where `randrange(1e25)` would silently select from a larger range than `randrange(10**25)`. (Originally suggested by Serhiy Storchaka gh-86388.)

- `argparse.ArgumentParser` changed encoding and error handler for reading arguments from file (e.g. `fromfile_prefix_chars` option) from default text encoding (e.g. `locale.getpreferredencoding(False)`) to filesystem encoding and error handler. Argument files should be encoded in UTF-8 instead of ANSI Codepage on Windows.

- Remove the `asyncore`-based `smtpd` module deprecated in Python 3.4.7 and 3.5.4. A recommended replacement is the `asyncio`-based aiosmtpd PyPI module.

- `shlex.split()`: Passing `None` for *s* argument now raises an exception, rather than reading `sys.stdin`. The feature was deprecated in Python 3.9. (Contributed by Victor Stinner in gh-94352.)

- The `os` module no longer accepts bytes-like paths, like `bytearray` and `memoryview` types: only the exact `bytes` type is accepted for bytes strings. (Contributed by Victor Stinner in gh-98393.)

- `syslog.openlog()` and `syslog.closelog()` now fail if used in subinterpreters. `syslog.syslog()` may still be used in subinterpreters, but now only if `syslog.openlog()` has already been called in the main interpreter. These new restrictions do not apply to the main interpreter, so only a very small set of users might be affected. This change helps with interpreter isolation. Furthermore, `syslog` is a wrapper around process-global resources, which are best managed from the main interpreter. (Contributed by Donghee Na in gh-99127.)

- The undocumented locking behavior of `cached_property()` is removed, because it locked across all instances of the class, leading to high lock contention. This means that a cached property getter function could now run more than once for a single instance, if two threads race. For most simple cached properties (e.g. those that are idempotent and simply calculate a value based on other attributes of the instance) this will be fine. If synchronization is needed, implement locking within the cached property getter function or around multi-threaded access points.

- `sys._current_exceptions()` now returns a mapping from thread-id to an exception instance, rather than to a `(typ, exc, tb)` tuple. (Contributed by Irit Katriel in gh-103176.)

- When extracting tar files using `tarfile` or `shutil.unpack_archive()`, pass the *filter* argument to limit features that may be surprising or dangerous. See tarfile-extraction-filter for details.

- The output of the `tokenize.tokenize()` and `tokenize.generate_tokens()` functions is now changed due to the changes introduced in **PEP 701**. This means that `STRING` tokens are not emitted any more for f-strings and the tokens described in **PEP 701** are now produced instead: `FSTRING_START`, `FSTRING_MIDDLE` and `FSTRING_END` are now emitted for f-string "string" parts in addition to the appropriate tokens for the tokenization in the expression components. For example for the f-string `f"start {1+1} end"` the old version of the tokenizer emitted:

```
1,0-1,18:           STRING         'f"start {1+1} end"'
```

  while the new version emits:

```
1,0-1,2:            FSTRING_START  'f"'
1,2-1,8:            FSTRING_MIDDLE 'start '
1,8-1,9:            OP             '{'
1,9-1,10:           NUMBER         '1'
1,10-1,11:          OP             '+'
1,11-1,12:          NUMBER         '1'
1,12-1,13:          OP             '}'
1,13-1,17:          FSTRING_MIDDLE ' end'
1,17-1,18:          FSTRING_END    '"'
```

  Additionally, there may be some minor behavioral changes as a consequence of the changes required to support **PEP 701**. Some of these changes include:

  - The `type` attribute of the tokens emitted when tokenizing some invalid Python characters such as `!` has changed from `ERRORTOKEN` to `OP`.

  - Incomplete single-line strings now also raise `tokenize.TokenError` as incomplete multiline strings do.

  - Some incomplete or invalid Python code now raises `tokenize.TokenError` instead of returning arbitrary `ERRORTOKEN` tokens when tokenizing it.
  - Mixing tabs and spaces as indentation in the same file is not supported anymore and will raise a `TabError`.

- 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 should provide `_is_main_interpreter()`. (See gh-112826.)

13 Build Changes

- Python no longer uses `setup.py` to build shared C extension modules. Build parameters like headers and libraries are detected in `configure` script. Extensions are built by `Makefile`. Most extensions use `pkg-config` and fall back to manual detection. (Contributed by Christian Heimes in gh-93939.)
- `va_start()` with two parameters, like `va_start(args, format)`, is now required to build Python. `va_start()` is no longer called with a single parameter. (Contributed by Kumar Aditya in gh-93207.)
- CPython now uses the ThinLTO option as the default link time optimization policy if the Clang compiler accepts the flag. (Contributed by Donghee Na in gh-89536.)
- Add `COMPILEALL_OPTS` variable in `Makefile` to override `compileall` options (default: `-j0`) in `make install`. Also merged the 3 `compileall` commands into a single command to build .pyc files for all optimization levels (0, 1, 2) at once. (Contributed by Victor Stinner in gh-99289.)
- Add platform triplets for 64-bit LoongArch:
  - loongarch64-linux-gnusf
  - loongarch64-linux-gnuf32
  - loongarch64-linux-gnu

  (Contributed by Zhang Na in gh-90656.)
- `PYTHON_FOR_REGEN` now require Python 3.10 or newer.
- Autoconf 2.71 and aclocal 1.16.4 is now required to regenerate `!configure`. (Contributed by Christian Heimes in gh-89886.)
- Windows builds and macOS installers from python.org now use OpenSSL 3.0.

14 C API Changes

14.1 New Features

- **PEP 697**: Introduce the Unstable C API tier, intended for low-level tools like debuggers and JIT compilers. This API may change in each minor release of CPython without deprecation warnings. Its contents are marked by the `PyUnstable_` prefix in names.

  Code object constructors:
  - `PyUnstable_Code_New()` (renamed from `PyCode_New`)
  - `PyUnstable_Code_NewWithPosOnlyArgs()` (renamed from `PyCode_NewWithPosOnlyArgs`)

  Extra storage for code objects (**PEP 523**):

  - `PyUnstable_Eval_RequestCodeExtraIndex()` (renamed from `_PyEval_RequestCodeExtraIndex`)
  - `PyUnstable_Code_GetExtra()` (renamed from `_PyCode_GetExtra`)
  - `PyUnstable_Code_SetExtra()` (renamed from `_PyCode_SetExtra`)

  The original names will continue to be available until the respective API changes.

  (Contributed by Petr Viktorin in gh-101101.)

- **PEP 697**: Add an API for extending types whose instance memory layout is opaque:
  - `PyType_Spec.basicsize` can be zero or negative to specify inheriting or extending the base class size.
  - `PyObject_GetTypeData()` and `PyType_GetTypeDataSize()` added to allow access to subclass-specific instance data.
  - `Py_TPFLAGS_ITEMS_AT_END` and `PyObject_GetItemData()` added to allow safely extending certain variable-sized types, including `PyType_Type`.
  - `Py_RELATIVE_OFFSET` added to allow defining `members` in terms of a subclass-specific struct.

  (Contributed by Petr Viktorin in gh-103509.)

- Add the new limited C API function `PyType_FromMetaclass()`, which generalizes the existing `PyType_FromModuleAndSpec()` using an additional metaclass argument. (Contributed by Wenzel Jakob in gh-93012.)

- API for creating objects that can be called using the vectorcall protocol was added to the Limited API:
  - `Py_TPFLAGS_HAVE_VECTORCALL`
  - `PyVectorcall_NARGS()`
  - `PyVectorcall_Call()`
  - `vectorcallfunc`

  The `Py_TPFLAGS_HAVE_VECTORCALL` flag is now removed from a class when the class's `__call__()` method is reassigned. This makes vectorcall safe to use with mutable types (i.e. heap types without the immutable flag, `Py_TPFLAGS_IMMUTABLETYPE`). Mutable types that do not override `tp_call` now inherit the `Py_TPFLAGS_HAVE_VECTORCALL` flag. (Contributed by Petr Viktorin in gh-93274.)

  The `Py_TPFLAGS_MANAGED_DICT` and `Py_TPFLAGS_MANAGED_WEAKREF` flags have been added. This allows extensions classes to support object `__dict__` and weakrefs with less bookkeeping, using less memory and with faster access.

- API for performing calls using the vectorcall protocol was added to the Limited API:
  - `PyObject_Vectorcall()`
  - `PyObject_VectorcallMethod()`
  - `PY_VECTORCALL_ARGUMENTS_OFFSET`

  This means that both the incoming and outgoing ends of the vector call protocol are now available in the Limited API. (Contributed by Wenzel Jakob in gh-98586.)

- Add two new public functions, `PyEval_SetProfileAllThreads()` and `PyEval_SetTraceAllThreads()`, that allow to set tracing and profiling functions in all running threads in addition to the calling one. (Contributed by Pablo Galindo in gh-93503.)

- Add new function `PyFunction_SetVectorcall()` to the C API which sets the vectorcall field of a given `PyFunctionObject`. (Contributed by Andrew Frost in gh-92257.)

- The C API now permits registering callbacks via `PyDict_AddWatcher()`, `PyDict_Watch()` and related APIs to be called whenever a dictionary is modified. This is intended for use by optimizing interpreters, JIT compilers, or debuggers. (Contributed by Carl Meyer in gh-91052.)

- Add `PyType_AddWatcher()` and `PyType_Watch()` API to register callbacks to receive notification on changes to a type. (Contributed by Carl Meyer in gh-91051.)

- Add `PyCode_AddWatcher()` and `PyCode_ClearWatcher()` APIs to register callbacks to receive notification on creation and destruction of code objects. (Contributed by Itamar Oren in gh-91054.)

- Add `PyFrame_GetVar()` and `PyFrame_GetVarString()` functions to get a frame variable by its name. (Contributed by Victor Stinner in gh-91248.)

- Add `PyErr_GetRaisedException()` and `PyErr_SetRaisedException()` for saving and restoring the current exception. These functions return and accept a single exception object, rather than the triple arguments of the now-deprecated `PyErr_Fetch()` and `PyErr_Restore()`. This is less error prone and a bit more efficient. (Contributed by Mark Shannon in gh-101578.)

- Add `_PyErr_ChainExceptions1`, which takes an exception instance, to replace the legacy-API `_PyErr_ChainExceptions`, which is now deprecated. (Contributed by Mark Shannon in gh-101578.)

- Add `PyException_GetArgs()` and `PyException_SetArgs()` as convenience functions for retrieving and modifying the `args` passed to the exception's constructor. (Contributed by Mark Shannon in gh-101578.)

- Add `PyErr_DisplayException()`, which takes an exception instance, to replace the legacy-api `PyErr_Display()`. (Contributed by Irit Katriel in gh-102755).

- **PEP 683**: Introduce *Immortal Objects*, which allows objects to bypass reference counts, and related changes to the C-API:

  - **`_Py_IMMORTAL_REFCNT`: The reference count that defines an object** as immortal.
  - `_Py_IsImmortal` Checks if an object has the immortal reference count.
  - **`PyObject_HEAD_INIT` This will now initialize reference count to** `_Py_IMMORTAL_REFCNT` when used with `Py_BUILD_CORE`.
  - **`SSTATE_INTERNED_IMMORTAL` An identifier for interned unicode objects** that are immortal.
  - **`SSTATE_INTERNED_IMMORTAL_STATIC` An identifier for interned unicode** objects that are immortal and static
  - **`sys.getunicodeinternedsize` This returns the total number of unicode** objects that have been interned. This is now needed for `refleak.py` to correctly track reference counts and allocated blocks

  (Contributed by Eddie Elizondo in gh-84436.)

- **PEP 684**: Add the new `Py_NewInterpreterFromConfig()` function and `PyInterpreterConfig`, which may be used to create sub-interpreters with their own GILs. (See *PEP 684: A Per-Interpreter GIL* for more info.) (Contributed by Eric Snow in gh-104110.)

- In the limited C API version 3.12, `Py_INCREF()` and `Py_DECREF()` functions are now implemented as opaque function calls to hide implementation details. (Contributed by Victor Stinner in gh-105387.)

14.2 Porting to Python 3.12

- Legacy Unicode APIs based on `Py_UNICODE*` representation has been removed. Please migrate to APIs based on UTF-8 or `wchar_t*`.
- Argument parsing functions like `PyArg_ParseTuple()` doesn't support `Py_UNICODE*` based format (e.g. `u`, `Z`) anymore. Please migrate to other formats for Unicode like `s`, `z`, `es`, and `U`.
- `tp_weaklist` for all static builtin types is always `NULL`. This is an internal-only field on `PyTypeObject` but we're pointing out the change in case someone happens to be accessing the field directly anyway. To avoid breakage, consider using the existing public C-API instead, or, if necessary, the (internal-only) `_PyObject_GET_WEAKREFS_LISTPTR()` macro.
- This internal-only `PyTypeObject.tp_subclasses` may now not be a valid object pointer. Its type was changed to `void*` to reflect this. We mention this in case someone happens to be accessing the internal-only field directly.

  To get a list of subclasses, call the Python method `__subclasses__()` (using `PyObject_CallMethod()`, for example).
- Add support of more formatting options (left aligning, octals, uppercase hexadecimals, `intmax_t`, `ptrdiff_t`, `wchar_t` C strings, variable width and precision) in `PyUnicode_FromFormat()` and `PyUnicode_FromFormatV()`. (Contributed by Serhiy Storchaka in gh-98836.)
- An unrecognized format character in `PyUnicode_FromFormat()` and `PyUnicode_FromFormatV()` now sets a `SystemError`. In previous versions it caused all the rest of the format string to be copied as-is to the result string, and any extra arguments discarded. (Contributed by Serhiy Storchaka in gh-95781.)
- Fix wrong sign placement in `PyUnicode_FromFormat()` and `PyUnicode_FromFormatV()`. (Contributed by Philip Georgi in gh-95504.)
- Extension classes wanting to add a `__dict__` or weak reference slot should use `Py_TPFLAGS_MANAGED_DICT` and `Py_TPFLAGS_MANAGED_WEAKREF` instead of `tp_dictoffset` and `tp_weaklistoffset`, respectively. The use of `tp_dictoffset` and `tp_weaklistoffset` is still supported, but does not fully support multiple inheritance (gh-95589), and performance may be worse. Classes declaring `Py_TPFLAGS_MANAGED_DICT` should call `_PyObject_VisitManagedDict()` and `_PyObject_ClearManagedDict()` to traverse and clear their instance's dictionaries. To clear weakrefs, call `PyObject_ClearWeakRefs()`, as before.
- The `PyUnicode_FSDecoder()` function no longer accepts bytes-like paths, like `bytearray` and `memoryview` types: only the exact `bytes` type is accepted for bytes strings. (Contributed by Victor Stinner in gh-98393.)
- The `Py_CLEAR`, `Py_SETREF` and `Py_XSETREF` macros now only evaluate their arguments once. If an argument has side effects, these side effects are no longer duplicated. (Contributed by Victor Stinner in gh-98724.)
- The interpreter's error indicator is now always normalized. This means that `PyErr_SetObject()`, `PyErr_SetString()` and the other functions that set the error indicator now normalize the exception before storing it. (Contributed by Mark Shannon in gh-101578.)
- `_Py_RefTotal` is no longer authoritative and only kept around for ABI compatibility. Note that it is an internal global and only available on debug builds. If you happen to be using it then you'll need to start using `_Py_GetGlobalRefTotal()`.
- The following functions now select an appropriate metaclass for the newly created type:
  - `PyType_FromSpec()`
  - `PyType_FromSpecWithBases()`
  - `PyType_FromModuleAndSpec()`

Creating classes whose metaclass overrides `tp_new` is deprecated, and in Python 3.14+ it will be disallowed. Note that these functions ignore `tp_new` of the metaclass, possibly allowing incomplete initialization.

Note that `PyType_FromMetaclass()` (added in Python 3.12) already disallows creating classes whose metaclass overrides `tp_new` (`__new__()` in Python).

Since `tp_new` overrides almost everything `PyType_From*` functions do, the two are incompatible with each other. The existing behavior – ignoring the metaclass for several steps of type creation – is unsafe in general, since (meta)classes assume that `tp_new` was called. There is no simple general workaround. One of the following may work for you:

- If you control the metaclass, avoid using `tp_new` in it:
  - If initialization can be skipped, it can be done in `tp_init` instead.
  - If the metaclass doesn't need to be instantiated from Python, set its `tp_new` to `NULL` using the `Py_TPFLAGS_DISALLOW_INSTANTIATION` flag. This makes it acceptable for `PyType_From*` functions.
- Avoid `PyType_From*` functions: if you don't need C-specific features (slots or setting the instance size), create types by calling the metaclass.
- If you *know* the `tp_new` can be skipped safely, filter the deprecation warning out using `warnings.catch_warnings()` from Python.

- `PyOS_InputHook` and `PyOS_ReadlineFunctionPointer` are no longer called in subinterpreters. This is because clients generally rely on process-wide global state (since these callbacks have no way of recovering extension module state).

  This also avoids situations where extensions may find themselves running in a subinterpreter that they don't support (or haven't yet been loaded in). See gh-104668 for more info.

- `PyLongObject` has had its internals changed for better performance. Although the internals of `PyLongObject` are private, they are used by some extension modules. The internal fields should no longer be accessed directly, instead the API functions beginning `PyLong_...` should be used instead. Two new *unstable* API functions are provided for efficient access to the value of `PyLongObject`s which fit into a single machine word:
  - `PyUnstable_Long_IsCompact()`
  - `PyUnstable_Long_CompactValue()`

- Custom allocators, set via `PyMem_SetAllocator()`, are now required to be thread-safe, regardless of memory domain. Allocators that don't have their own state, including "hooks", are not affected. If your custom allocator is not already thread-safe and you need guidance then please create a new GitHub issue and CC `@ericsnowcurrently`.

14.3 Deprecated

- In accordance with **PEP 699**, the `ma_version_tag` field in `PyDictObject` is deprecated for extension modules. Accessing this field will generate a compiler warning at compile time. This field will be removed in Python 3.14. (Contributed by Ramvikrams and Kumar Aditya in gh-101193. PEP by Ken Jin.)

- Deprecate global configuration variable:
  - `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. (Contributed by Victor Stinner in gh-77782.)

- Creating `immutable types` with mutable bases is deprecated and will be disabled in Python 3.14. (gh-95388)
- The `structmember.h` header is deprecated, though it continues to be available and there are no plans to remove it.

  Its contents are now available just by including `Python.h`, with a `Py` prefix added if it was missing:

  - `PyMemberDef`, `PyMember_GetOne()` and `PyMember_SetOne()`
  - Type macros like `Py_T_INT`, `Py_T_DOUBLE`, etc. (previously `T_INT`, `T_DOUBLE`, etc.)
  - The flags `Py_READONLY` (previously `READONLY`) and `Py_AUDIT_READ` (previously all uppercase)

  Several items are not exposed from `Python.h`:

  - `T_OBJECT` (use `Py_T_OBJECT_EX`)
  - `T_NONE` (previously undocumented, and pretty quirky)
  - The macro `WRITE_RESTRICTED` which does nothing.
  - The macros `RESTRICTED` and `READ_RESTRICTED`, equivalents of `Py_AUDIT_READ`.
  - In some configurations, `<stddef.h>` is not included from `Python.h`. It should be included manually when using `offsetof()`.

  The deprecated header continues to provide its original contents under the original names. Your old code can stay unchanged, unless the extra include and non-namespaced macros bother you greatly.

  (Contributed in gh-47146 by Petr Viktorin, based on earlier work by Alexander Belopolsky and Matthias Braun.)

- `PyErr_Fetch()` and `PyErr_Restore()` are deprecated. Use `PyErr_GetRaisedException()` and `PyErr_SetRaisedException()` instead. (Contributed by Mark Shannon in gh-101578.)

- `PyErr_Display()` is deprecated. Use `PyErr_DisplayException()` instead. (Contributed by Irit Katriel in gh-102755).
- `_PyErr_ChainExceptions` is deprecated. Use `_PyErr_ChainExceptions1` instead. (Contributed by Irit Katriel in gh-102192.)
- Using `PyType_FromSpec()`, `PyType_FromSpecWithBases()` or `PyType_FromModuleAndSpec()` to create a class whose metaclass overrides `tp_new` is deprecated. Call the metaclass instead.

Pending Removal in Python 3.14

- The `ma_version_tag` field in `PyDictObject` for extension modules (**PEP 699**; gh-101193).
- 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.
- Creating `immutable types` with mutable bases (gh-95388).

Pending Removal in Python 3.15

- `PyImport_ImportModuleNoBlock()`: use `PyImport_ImportModule()`
- `Py_UNICODE_WIDE` type: use `wchar_t`
- `Py_UNICODE` type: use `wchar_t`
- Python initialization functions:
  - `PySys_ResetWarnOptions()`: clear `sys.warnoptions` and `warnings.filters`
  - `Py_GetExecPrefix()`: get `sys.exec_prefix`
  - `Py_GetPath()`: get `sys.path`
  - `Py_GetPrefix()`: get `sys.prefix`
  - `Py_GetProgramFullPath()`: get `sys.executable`
  - `Py_GetProgramName()`: get `sys.executable`
  - `Py_GetPythonHome()`: get `PyConfig.home` or the `PYTHONHOME` environment variable

Pending Removal in Future Versions

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

- `Py_TPFLAGS_HAVE_FINALIZE`: unneeded since Python 3.8
- `PyErr_Fetch()`: use `PyErr_GetRaisedException()`
- `PyErr_NormalizeException()`: use `PyErr_GetRaisedException()`
- `PyErr_Restore()`: use `PyErr_SetRaisedException()`
- `PyModule_GetFilename()`: use `PyModule_GetFilenameObject()`
- `PyOS_AfterFork()`: use `PyOS_AfterFork_Child()`
- `PySlice_GetIndicesEx()`: use `PySlice_Unpack()` and `PySlice_AdjustIndices()`
- `PyUnicode_AsDecodedObject()`: use `PyCodec_Decode()`
- `PyUnicode_AsDecodedUnicode()`: use `PyCodec_Decode()`
- `PyUnicode_AsEncodedObject()`: use `PyCodec_Encode()`
- `PyUnicode_AsEncodedUnicode()`: use `PyCodec_Encode()`
- `PyUnicode_READY()`: unneeded since Python 3.12
- `PyErr_Display()`: use `PyErr_DisplayException()`
- `_PyErr_ChainExceptions()`: use `_PyErr_ChainExceptions1`
- `PyBytesObject.ob_shash` member: call `PyObject_Hash()` instead
- `PyDictObject.ma_version_tag` member
- Thread Local Storage (TLS) API:
  - `PyThread_create_key()`: use `PyThread_tss_alloc()`
  - `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()`: unneeded since Python 3.7

14.4 Removed

- Remove the `token.h` header file. There was never any public tokenizer C API. The `token.h` header file was only designed to be used by Python internals. (Contributed by Victor Stinner in gh-92651.)
- Legacy Unicode APIs have been removed. See **PEP 623** for detail.
  - `PyUnicode_WCHAR_KIND`
  - `PyUnicode_AS_UNICODE()`
  - `PyUnicode_AsUnicode()`
  - `PyUnicode_AsUnicodeAndSize()`
  - `PyUnicode_AS_DATA()`
  - `PyUnicode_FromUnicode()`
  - `PyUnicode_GET_SIZE()`
  - `PyUnicode_GetSize()`
  - `PyUnicode_GET_DATA_SIZE()`
- Remove the `PyUnicode_InternImmortal()` function macro. (Contributed by Victor Stinner in gh-85858.)

15 Notable changes in 3.12.4

15.1 ipaddress

- Fixed `is_global` and `is_private` behavior in `IPv4Address`, `IPv6Address`, `IPv4Network` and `IPv6Network`.

Index

E

P