What's New in Python

Release 3.11.2

A. M. Kuchling

April 05, 2023

Python Software Foundation
Email: docs@python.org

Contents

Release 3.11.2

Date April 05, 2023

Editor Pablo Galindo Salgado

This article explains the new features in Python 3.11, compared to 3.10.

For full details, see the changelog.

1 Summary – Release highlights

- Python 3.11 is between 10-60% faster than Python 3.10. On average, we measured a 1.25x speedup on the standard benchmark suite. See *Faster CPython* for details.

New syntax features:

- *PEP 654: Exception Groups and except**

New built-in features:

- *PEP 678: Exceptions can be enriched with notes*

New standard library modules:

- **PEP 680**: `tomllib` — Support for parsing TOML in the Standard Library

Interpreter improvements:

- *PEP 657: Fine-grained error locations in tracebacks*
- New `-P` command line option and `PYTHONSAFEPATH` environment variable to *disable automatically prepending potentially unsafe paths* to `sys.path`

New typing features:

- *PEP 646: Variadic generics*
- *PEP 655: Marking individual TypedDict items as required or not-required*
- *PEP 673: Self type*
- *PEP 675: Arbitrary literal string type*
- *PEP 681: Data class transforms*

Important deprecations, removals and restrictions:

- **PEP 594**: *Many legacy standard library modules have been deprecated* and will be removed in Python 3.13
- **PEP 624**: *Py_UNICODE encoder APIs have been removed*
- **PEP 670**: *Macros converted to static inline functions*

2 New Features

2.1 PEP 657: Fine-grained error locations in tracebacks

When printing tracebacks, the interpreter will now point to the exact expression that caused the error, instead of just the line. For example:

```
Traceback (most recent call last):
  File "distance.py", line 11, in <module>
    print(manhattan_distance(p1, p2))
          ^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "distance.py", line 6, in manhattan_distance
    return abs(point_1.x - point_2.x) + abs(point_1.y - point_2.y)
```

(continues on next page)

(continued from previous page)

```
                           ^^^^^^^^^
AttributeError: 'NoneType' object has no attribute 'x'
```

Previous versions of the interpreter would point to just the line, making it ambiguous which object was `None`. These enhanced errors can also be helpful when dealing with deeply nested `dict` objects and multiple function calls:

```
Traceback (most recent call last):
  File "query.py", line 37, in <module>
    magic_arithmetic('foo')
  File "query.py", line 18, in magic_arithmetic
    return add_counts(x) / 25
           ^^^^^^^^^^^^^
  File "query.py", line 24, in add_counts
    return 25 + query_user(user1) + query_user(user2)
                ^^^^^^^^^^^^^^^^^
  File "query.py", line 32, in query_user
    return 1 + query_count(db, response['a']['b']['c']['user'], retry=True)
                               ~~~~~~~~~~~~~~~~~~~~^^^^^
TypeError: 'NoneType' object is not subscriptable
```

As well as complex arithmetic expressions:

```
Traceback (most recent call last):
  File "calculation.py", line 54, in <module>
    result = (x / y / z) * (a / b / c)
             ~~~~~~^~~
ZeroDivisionError: division by zero
```

Additionally, the information used by the enhanced traceback feature is made available via a general API, that can be used to correlate bytecode instructions with source code location. This information can be retrieved using:

- The `codeobject.co_positions()` method in Python.
- The `PyCode_Addr2Location()` function in the C API.

See **PEP 657** for more details. (Contributed by Pablo Galindo, Batuhan Taskaya and Ammar Askar in bpo-43950.)

Note: This feature requires storing column positions in codeobjects, which may result in a small increase in interpreter memory usage and disk usage for compiled Python files. To avoid storing the extra information and deactivate printing the extra traceback information, use the `-X no_debug_ranges` command line option or the `PYTHONNODEBUGRANGES` environment variable.

2.2 PEP 654: Exception Groups and `except*`

PEP 654 introduces language features that enable a program to raise and handle multiple unrelated exceptions simultaneously. The builtin types `ExceptionGroup` and `BaseExceptionGroup` make it possible to group exceptions and raise them together, and the new `except*` syntax generalizes `except` to match subgroups of exception groups.

See **PEP 654** for more details.

(Contributed by Irit Katriel in bpo-45292. PEP written by Irit Katriel, Yury Selivanov and Guido van Rossum.)

2.3 PEP 678: Exceptions can be enriched with notes

The `add_note()` method is added to `BaseException`. It can be used to enrich exceptions with context information that is not available at the time when the exception is raised. The added notes appear in the default traceback.

See **PEP 678** for more details.

(Contributed by Irit Katriel in bpo-45607. PEP written by Zac Hatfield-Dodds.)

2.4 Windows `py.exe` launcher improvements

The copy of the launcher included with Python 3.11 has been significantly updated. It now supports company/tag syntax as defined in **PEP 514** using the `-V:<company>/<tag>` argument instead of the limited `-<major>.<minor>`. This allows launching distributions other than `PythonCore`, the one hosted on python.org.

When using `-V:` selectors, either company or tag can be omitted, but all installs will be searched. For example, `-V:OtherPython/` will select the "best" tag registered for `OtherPython`, while `-V:3.11` or `-V:/3.11` will select the "best" distribution with tag `3.11`.

When using the legacy `-<major>`, `-<major>.<minor>`, `-<major>-<bitness>` or `-<major>.<minor>-<bitness>` arguments, all existing behaviour should be preserved from past versions, and only releases from `PythonCore` will be selected. However, the `-64` suffix now implies "not 32-bit" (not necessarily x86-64), as there are multiple supported 64-bit platforms. 32-bit runtimes are detected by checking the runtime's tag for a `-32` suffix. All releases of Python since 3.5 have included this in their 32-bit builds.

3 New Features Related to Type Hints

This section covers major changes affecting **PEP 484** type hints and the `typing` module.

3.1 PEP 646: Variadic generics

PEP 484 previously introduced `TypeVar`, enabling creation of generics parameterised with a single type. **PEP 646** adds `TypeVarTuple`, enabling parameterisation with an *arbitrary* number of types. In other words, a `TypeVarTuple` is a *variadic* type variable, enabling *variadic* generics.

This enables a wide variety of use cases. In particular, it allows the type of array-like structures in numerical computing libraries such as NumPy and TensorFlow to be parameterised with the array *shape*. Static type checkers will now be able to catch shape-related bugs in code that uses these libraries.

See **PEP 646** for more details.

(Contributed by Matthew Rahtz in bpo-43224, with contributions by Serhiy Storchaka and Jelle Zijlstra. PEP written by Mark Mendoza, Matthew Rahtz, Pradeep Kumar Srinivasan, and Vincent Siles.)

3.2 PEP 655: Marking individual `TypedDict` items as required or not-required

`Required` and `NotRequired` provide a straightforward way to mark whether individual items in a `TypedDict` must be present. Previously, this was only possible using inheritance.

All fields are still required by default, unless the *total* parameter is set to `False`, in which case all fields are still not-required by default. For example, the following specifies a `TypedDict` with one required and one not-required key:

```
class Movie(TypedDict):
   title: str
   year: NotRequired[int]
```

(continues on next page)

(continued from previous page)

```
m1: Movie = {"title": "Black Panther", "year": 2018}  # OK
m2: Movie = {"title": "Star Wars"}  # OK (year is not required)
m3: Movie = {"year": 2022}  # ERROR (missing required field title)
```

The following definition is equivalent:

```
class Movie(TypedDict, total=False):
   title: Required[str]
   year: int
```

See **PEP 655** for more details.

(Contributed by David Foster and Jelle Zijlstra in bpo-47087. PEP written by David Foster.)

3.3 PEP 673: `Self` type

The new `Self` annotation provides a simple and intuitive way to annotate methods that return an instance of their class. This behaves the same as the `TypeVar`-based approach **specified in PEP 484**, but is more concise and easier to follow.

Common use cases include alternative constructors provided as `classmethods`, and `__enter__()` methods that return `self`:

```
class MyLock:
    def __enter__(self) -> Self:
        self.lock()
        return self

    ...

class MyInt:
    @classmethod
    def fromhex(cls, s: str) -> Self:
        return cls(int(s, 16))

    ...
```

`Self` can also be used to annotate method parameters or attributes of the same type as their enclosing class.

See **PEP 673** for more details.

(Contributed by James Hilton-Balfe in bpo-46534. PEP written by Pradeep Kumar Srinivasan and James Hilton-Balfe.)

3.4 PEP 675: Arbitrary literal string type

The new `LiteralString` annotation may be used to indicate that a function parameter can be of any literal string type. This allows a function to accept arbitrary literal string types, as well as strings created from other literal strings. Type checkers can then enforce that sensitive functions, such as those that execute SQL statements or shell commands, are called only with static arguments, providing protection against injection attacks.

For example, a SQL query function could be annotated as follows:

```
def run_query(sql: LiteralString) -> ...
    ...

def caller(
    arbitrary_string: str,
    query_string: LiteralString,
```

(continues on next page)

(continued from previous page)

```
    table_name: LiteralString,
) -> None:
    run_query("SELECT * FROM students")       # ok
    run_query(query_string)                   # ok
    run_query("SELECT * FROM " + table_name)  # ok
    run_query(arbitrary_string)               # type checker error
    run_query(                                # type checker error
        f"SELECT * FROM students WHERE name = {arbitrary_string}"
    )
```

See **PEP 675** for more details.

(Contributed by Jelle Zijlstra in bpo-47088. PEP written by Pradeep Kumar Srinivasan and Graham Bleaney.)

3.5 PEP 681: Data class transforms

`dataclass_transform` may be used to decorate a class, metaclass, or a function that is itself a decorator. The presence of `@dataclass_transform()` tells a static type checker that the decorated object performs runtime "magic" that transforms a class, giving it `dataclass`-like behaviors.

For example:

```
# The create_model decorator is defined by a library.
@typing.dataclass_transform()
def create_model(cls: Type[T]) -> Type[T]:
    cls.__init__ = ...
    cls.__eq__ = ...
    cls.__ne__ = ...
    return cls

# The create_model decorator can now be used to create new model classes:
@create_model
class CustomerModel:
    id: int
    name: str

c = CustomerModel(id=327, name="Eric Idle")
```

See **PEP 681** for more details.

(Contributed by Jelle Zijlstra in gh-91860. PEP written by Erik De Bonte and Eric Traut.)

3.6 PEP 563 may not be the future

PEP 563 Postponed Evaluation of Annotations (the `from __future__ import annotations` future statement) that was originally planned for release in Python 3.10 has been put on hold indefinitely. See this message from the Steering Council for more information.

4 Other Language Changes

- Starred unpacking expressions can now be used in `for` statements. (See bpo-46725 for more details.)
- Asynchronous comprehensions are now allowed inside comprehensions in asynchronous functions. Outer comprehensions implicitly become asynchronous in this case. (Contributed by Serhiy Storchaka in bpo-33346.)
- A `TypeError` is now raised instead of an `AttributeError` in `with` statements and `contextlib.ExitStack.enter_context()` for objects that do not support the context manager protocol, and in `async with` statements and `contextlib.AsyncExitStack.enter_async_context()` for objects not supporting the asynchronous context manager protocol. (Contributed by Serhiy Storchaka in bpo-12022 and bpo-44471.)
- Added `object.__getstate__()`, which provides the default implementation of the `__getstate__()` method. `copy`ing and `pickle`ing instances of subclasses of builtin types `bytearray`, `set`, `frozenset`, `collections.OrderedDict`, `collections.deque`, `weakref.WeakSet`, and `datetime.tzinfo` now copies and pickles instance attributes implemented as slots. (Contributed by Serhiy Storchaka in bpo-26579.)
- Added a `-P` command line option and a `PYTHONSAFEPATH` environment variable, which disable the automatic prepending to `sys.path` of the script's directory when running a script, or the current directory when using `-c` and `-m`. This ensures only stdlib and installed modules are picked up by `import`, and avoids unintentionally or maliciously shadowing modules with those in a local (and typically user-writable) directory. (Contributed by Victor Stinner in gh-57684.)
- A `"z"` option was added to the formatspec that coerces negative to positive zero after rounding to the format precision. See **PEP 682** for more details. (Contributed by John Belmonte in gh-90153.)
- Bytes are no longer accepted on `sys.path`. Support broke sometime between Python 3.2 and 3.6, with no one noticing until after Python 3.10.0 was released. In addition, bringing back support would be problematic due to interactions between `-b` and `sys.path_importer_cache` when there is a mixture of `str` and `bytes` keys. (Contributed by Thomas Grainger in gh-91181.)

5 Other CPython Implementation Changes

- The special methods `__complex__()` for `complex` and `__bytes__()` for `bytes` are implemented to support the `typing.SupportsComplex` and `typing.SupportsBytes` protocols. (Contributed by Mark Dickinson and Dong-hee Na in bpo-24234.)
- `siphash13` is added as a new internal hashing algorithm. It has similar security properties as `siphash24`, but it is slightly faster for long inputs. `str`, `bytes`, and some other types now use it as the default algorithm for `hash()`. **PEP 552** hash-based .pyc files now use `siphash13` too. (Contributed by Inada Naoki in bpo-29410.)
- When an active exception is re-raised by a `raise` statement with no parameters, the traceback attached to this exception is now always `sys.exc_info()[1].__traceback__`. This means that changes made to the traceback in the current `except` clause are reflected in the re-raised exception. (Contributed by Irit Katriel in bpo-45711.)
- The interpreter state's representation of handled exceptions (aka `exc_info` or `_PyErr_StackItem`) now only has the `exc_value` field; `exc_type` and `exc_traceback` have been removed, as they can be derived from `exc_value`. (Contributed by Irit Katriel in bpo-45711.)

- A new command line option, `AppendPath`, has been added for the Windows installer. It behaves similarly to `PrependPath`, but appends the install and scripts directories instead of prepending them. (Contributed by Bastian Neuburger in bpo-44934.)
- The `PyConfig.module_search_paths_set` field must now be set to `1` for initialization to use `PyConfig.module_search_paths` to initialize `sys.path`. Otherwise, initialization will recalculate the path and replace any values added to `module_search_paths`.
- The output of the `--help` option now fits in 50 lines/80 columns. Information about Python environment variables and `-X` options is now available using the respective `--help-env` and `--help-xoptions` flags, and with the new `--help-all`. (Contributed by Éric Araujo in bpo-46142.)
- Converting between `int` and `str` in bases other than 2 (binary), 4, 8 (octal), 16 (hexadecimal), or 32 such as base 10 (decimal) now raises a `ValueError` if the number of digits in string form is above a limit to avoid potential denial of service attacks due to the algorithmic complexity. This is a mitigation for CVE-2020-10735. This limit can be configured or disabled by environment variable, command line flag, or `sys` APIs. See the integer string conversion length limitation documentation. The default limit is 4300 digits in string form.

6 New Modules

- `tomllib`: For parsing TOML. See **PEP 680** for more details. (Contributed by Taneli Hukkinen in bpo-40059.)
- `wsgiref.types`: **WSGI**-specific types for static type checking. (Contributed by Sebastian Rittau in bpo-42012.)

7 Improved Modules

7.1 asyncio

- Added the `TaskGroup` class, an asynchronous context manager holding a group of tasks that will wait for all of them upon exit. For new code this is recommended over using `create_task()` and `gather()` directly. (Contributed by Yury Selivanov and others in gh-90908.)
- Added `timeout()`, an asynchronous context manager for setting a timeout on asynchronous operations. For new code this is recommended over using `wait_for()` directly. (Contributed by Andrew Svetlov in gh-90927.)
- Added the `Runner` class, which exposes the machinery used by `run()`. (Contributed by Andrew Svetlov in gh-91218.)
- Added the `Barrier` class to the synchronization primitives in the asyncio library, and the related `BrokenBarrierError` exception. (Contributed by Yves Duprat and Andrew Svetlov in gh-87518.)
- Added keyword argument *all_errors* to `asyncio.loop.create_connection()` so that multiple connection errors can be raised as an `ExceptionGroup`.
- Added the `asyncio.StreamWriter.start_tls()` method for upgrading existing stream-based connections to TLS. (Contributed by Ian Good in bpo-34975.)
- Added raw datagram socket functions to the event loop: `sock_sendto()`, `sock_recvfrom()` and `sock_recvfrom_into()`. These have implementations in `SelectorEventLoop` and `ProactorEventLoop`. (Contributed by Alex Grönholm in bpo-46805.)
- Added `cancelling()` and `uncancel()` methods to `Task`. These are primarily intended for internal use, notably by `TaskGroup`.

7.2 contextlib

- Added non parallel-safe `chdir()` context manager to change the current working directory and then restore it on exit. Simple wrapper around `chdir()`. (Contributed by Filipe Laíns in bpo-25625)

7.3 dataclasses

- Change field default mutability check, allowing only defaults which are hashable instead of any object which is not an instance of `dict`, `list` or `set`. (Contributed by Eric V. Smith in bpo-44674.)

7.4 datetime

- Add `datetime.UTC`, a convenience alias for `datetime.timezone.utc`. (Contributed by Kabir Kwatra in gh-91973.)
- `datetime.date.fromisoformat()`, `datetime.time.fromisoformat()` and `datetime.datetime.fromisoformat()` can now be used to parse most ISO 8601 formats (barring only those that support fractional hours and minutes). (Contributed by Paul Ganssle in gh-80010.)

7.5 enum

- Renamed `EnumMeta` to `EnumType` (`EnumMeta` kept as an alias).
- Added `StrEnum`, with members that can be used as (and must be) strings.
- Added `ReprEnum`, which only modifies the `__repr__()` of members while returning their literal values (rather than names) for `__str__()` and `__format__()` (used by `str()`, `format()` and f-strings).
- Changed `IntEnum`, `IntFlag` and `StrEnum` to now inherit from `ReprEnum`, so their `str()` output now matches `format()` (both `str(AnIntEnum.ONE)` and `format(AnIntEnum.ONE)` return `'1'`, whereas before `str(AnIntEnum.ONE)` returned `'AnIntEnum.ONE'`.
- Changed `Enum.__format__()` (the default for `format()`, `str.format()` and f-strings) of enums with mixed-in types (e.g. `int`, `str`) to also include the class name in the output, not just the member's key. This matches the existing behavior of `enum.Enum.__str__()`, returning e.g. `'AnEnum.MEMBER'` for an enum `AnEnum(str, Enum)` instead of just `'MEMBER'`.
- Added a new *boundary* class parameter to `Flag` enums and the `FlagBoundary` enum with its options, to control how to handle out-of-range flag values.
- Added the `verify()` enum decorator and the `EnumCheck` enum with its options, to check enum classes against several specific constraints.
- Added the `member()` and `nonmember()` decorators, to ensure the decorated object is/is not converted to an enum member.
- Added the `property()` decorator, which works like `property()` except for enums. Use this instead of `types.DynamicClassAttribute()`.
- Added the `global_enum()` enum decorator, which adjusts `__repr__()` and `__str__()` to show values as members of their module rather than the enum class. For example, `'re.ASCII'` for the `ASCII` member of `re.RegexFlag` rather than `'RegexFlag.ASCII'`.
- Enhanced `Flag` to support `len()`, iteration and `in`/`not in` on its members. For example, the following now works: `len(AFlag(3)) == 2 and list(AFlag(3)) == (AFlag.ONE, AFlag.TWO)`
- Changed `Enum` and `Flag` so that members are now defined before `__init_subclass__()` is called; `dir()` now includes methods, etc., from mixed-in data types.
- Changed `Flag` to only consider primary values (power of two) canonical while composite values (`3`, `6`, `10`, etc.) are considered aliases; inverted flags are coerced to their positive equivalent.

7.6 fcntl

- On FreeBSD, the `F_DUP2FD` and `F_DUP2FD_CLOEXEC` flags respectively are supported, the former equals to `dup2` usage while the latter set the `FD_CLOEXEC` flag in addition.

7.7 fractions

- Support **PEP 515**-style initialization of `Fraction` from string. (Contributed by Sergey B Kirpichev in bpo-44258.)
- `Fraction` now implements an `__int__` method, so that an `isinstance(some_fraction, typing.SupportsInt)` check passes. (Contributed by Mark Dickinson in bpo-44547.)

7.8 functools

- `functools.singledispatch()` now supports `types.UnionType` and `typing.Union` as annotations to the dispatch argument.:

```
>>> from functools import singledispatch
>>> @singledispatch
... def fun(arg, verbose=False):
...     if verbose:
...         print("Let me just say,", end=" ")
...     print(arg)
...
>>> @fun.register
... def _(arg: int | float, verbose=False):
...     if verbose:
...         print("Strength in numbers, eh?", end=" ")
...     print(arg)
...
>>> from typing import Union
>>> @fun.register
... def _(arg: Union[list, set], verbose=False):
...     if verbose:
...         print("Enumerate this:")
...     for i, elem in enumerate(arg):
...         print(i, elem)
...
```

(Contributed by Yurii Karabas in bpo-46014.)

7.9 hashlib

- `hashlib.blake2b()` and `hashlib.blake2s()` now prefer libb2 over Python's vendored copy. (Contributed by Christian Heimes in bpo-47095.)
- The internal `_sha3` module with SHA3 and SHAKE algorithms now uses *tiny_sha3* instead of the *Keccak Code Package* to reduce code and binary size. The `hashlib` module prefers optimized SHA3 and SHAKE implementations from OpenSSL. The change affects only installations without OpenSSL support. (Contributed by Christian Heimes in bpo-47098.)
- Add `hashlib.file_digest()`, a helper function for efficient hashing of files or file-like objects. (Contributed by Christian Heimes in gh-89313.)

7.10 IDLE and idlelib

- Apply syntax highlighting to `.pyi` files. (Contributed by Alex Waygood and Terry Jan Reedy in bpo-45447.)
- Include prompts when saving Shell with inputs and outputs. (Contributed by Terry Jan Reedy in gh-95191.)

7.11 inspect

- Add `getmembers_static()` to return all members without triggering dynamic lookup via the descriptor protocol. (Contributed by Weipeng Hong in bpo-30533.)
- Add `ismethodwrapper()` for checking if the type of an object is a `MethodWrapperType`. (Contributed by Hakan Çelik in bpo-29418.)
- Change the frame-related functions in the `inspect` module to return new `FrameInfo` and `Traceback` class instances (backwards compatible with the previous named tuple-like interfaces) that includes the extended **PEP 657** position information (end line number, column and end column). The affected functions are:
  - `inspect.getframeinfo()`
  - `inspect.getouterframes()`
  - `inspect.getinnerframes()`,
  - `inspect.stack()`
  - `inspect.trace()`

  (Contributed by Pablo Galindo in gh-88116.)

7.12 locale

- Add `locale.getencoding()` to get the current locale encoding. It is similar to `locale.getpreferredencoding(False)` but ignores the Python UTF-8 Mode.

7.13 logging

- Added `getLevelNamesMapping()` to return a mapping from logging level names (e.g. `'CRITICAL'`) to the values of their corresponding levels (e.g. `50`, by default). (Contributed by Andrei Kulakovin in gh-88024.)
- Added a `createSocket()` method to `SysLogHandler`, to match `SocketHandler.createSocket()`. It is called automatically during handler initialization and when emitting an event, if there is no active socket. (Contributed by Kirill Pinchuk in gh-88457.)

7.14 math

- Add `math.exp2()`: return 2 raised to the power of x. (Contributed by Gideon Mitchell in bpo-45917.)
- Add `math.cbrt()`: return the cube root of x. (Contributed by Ajith Ramachandran in bpo-44357.)
- The behaviour of two `math.pow()` corner cases was changed, for consistency with the IEEE 754 specification. The operations `math.pow(0.0, -math.inf)` and `math.pow(-0.0, -math.inf)` now return `inf`. Previously they raised `ValueError`. (Contributed by Mark Dickinson in bpo-44339.)
- The `math.nan` value is now always available. (Contributed by Victor Stinner in bpo-46917.)

7.15 operator

- A new function `operator.call` has been added, such that `operator.call(obj, *args, **kwargs) == obj(*args, **kwargs)`. (Contributed by Antony Lee in bpo-44019.)

7.16 os

- On Windows, `os.urandom()` now uses `BCryptGenRandom()`, instead of `CryptGenRandom()` which is deprecated. (Contributed by Dong-hee Na in bpo-44611.)

7.17 pathlib

- `glob()` and `rglob()` return only directories if *pattern* ends with a pathname components separator: `sep` or `altsep`. (Contributed by Eisuke Kawasima in bpo-22276 and bpo-33392.)

7.18 re

- Atomic grouping (`(?>...)`) and possessive quantifiers (`*+`, `++`, `?+`, `{m,n}+`) are now supported in regular expressions. (Contributed by Jeffrey C. Jacobs and Serhiy Storchaka in bpo-433030.)

7.19 shutil

- Add optional parameter *dir_fd* in `shutil.rmtree()`. (Contributed by Serhiy Storchaka in bpo-46245.)

7.20 socket

- Add CAN Socket support for NetBSD. (Contributed by Thomas Klausner in bpo-30512.)
- `create_connection()` has an option to raise, in case of failure to connect, an `ExceptionGroup` containing all errors instead of only raising the last error. (Contributed by Irit Katriel in bpo-29980.)

7.21 sqlite3

- You can now disable the authorizer by passing `None` to `set_authorizer()`. (Contributed by Erlend E. Aasland in bpo-44491.)
- Collation name `create_collation()` can now contain any Unicode character. Collation names with invalid characters now raise `UnicodeEncodeError` instead of `sqlite3.ProgrammingError`. (Contributed by Erlend E. Aasland in bpo-44688.)
- `sqlite3` exceptions now include the SQLite extended error code as `sqlite_errorcode` and the SQLite error name as `sqlite_errorname`. (Contributed by Aviv Palivoda, Daniel Shahaf, and Erlend E. Aasland in bpo-16379 and bpo-24139.)
- Add `setlimit()` and `getlimit()` to `sqlite3.Connection` for setting and getting SQLite limits by connection basis. (Contributed by Erlend E. Aasland in bpo-45243.)
- `sqlite3` now sets `sqlite3.threadsafety` based on the default threading mode the underlying SQLite library has been compiled with. (Contributed by Erlend E. Aasland in bpo-45613.)
- `sqlite3` C callbacks now use unraisable exceptions if callback tracebacks are enabled. Users can now register an `unraisable hook handler` to improve their debug experience. (Contributed by Erlend E. Aasland in bpo-45828.)
- Fetch across rollback no longer raises `InterfaceError`. Instead we leave it to the SQLite library to handle these cases. (Contributed by Erlend E. Aasland in bpo-44092.)

- Add `serialize()` and `deserialize()` to `sqlite3.Connection` for serializing and deserializing databases. (Contributed by Erlend E. Aasland in bpo-41930.)
- Add `create_window_function()` to `sqlite3.Connection` for creating aggregate window functions. (Contributed by Erlend E. Aasland in bpo-34916.)
- Add `blobopen()` to `sqlite3.Connection`. `sqlite3.Blob` allows incremental I/O operations on blobs. (Contributed by Aviv Palivoda and Erlend E. Aasland in bpo-24905.)

7.22 string

- Add `get_identifiers()` and `is_valid()` to `string.Template`, which respectively return all valid placeholders, and whether any invalid placeholders are present. (Contributed by Ben Kehoe in gh-90465.)

7.23 sys

- `sys.exc_info()` now derives the `type` and `traceback` fields from the `value` (the exception instance), so when an exception is modified while it is being handled, the changes are reflected in the results of subsequent calls to `exc_info()`. (Contributed by Irit Katriel in bpo-45711.)
- Add `sys.exception()` which returns the active exception instance (equivalent to `sys.exc_info()[1]`). (Contributed by Irit Katriel in bpo-46328.)
- Add the `sys.flags.safe_path` flag. (Contributed by Victor Stinner in gh-57684.)

7.24 sysconfig

- Three new installation schemes (*posix_venv*, *nt_venv* and *venv*) were added and are used when Python creates new virtual environments or when it is running from a virtual environment. The first two schemes (*posix_venv* and *nt_venv*) are OS-specific for non-Windows and Windows, the *venv* is essentially an alias to one of them according to the OS Python runs on. This is useful for downstream distributors who modify `sysconfig.get_preferred_scheme()`. Third party code that creates new virtual environments should use the new *venv* installation scheme to determine the paths, as does `venv`. (Contributed by Miro Hrončok in bpo-45413.)

7.25 tempfile

- `SpooledTemporaryFile` objects now fully implement the methods of `io.BufferedIOBase` or `io.TextIOBase` (depending on file mode). This lets them work correctly with APIs that expect file-like objects, such as compression modules. (Contributed by Carey Metcalfe in gh-70363.)

7.26 threading

- On Unix, if the `sem_clockwait()` function is available in the C library (glibc 2.30 and newer), the `threading.Lock.acquire()` method now uses the monotonic clock (`time.CLOCK_MONOTONIC`) for the timeout, rather than using the system clock (`time.CLOCK_REALTIME`), to not be affected by system clock changes. (Contributed by Victor Stinner in bpo-41710.)

7.27 time

- On Unix, `time.sleep()` now uses the `clock_nanosleep()` or `nanosleep()` function, if available, which has a resolution of 1 nanosecond (10^{-9} seconds), rather than using `select()` which has a resolution of 1 microsecond (10^{-6} seconds). (Contributed by Benjamin Szőke and Victor Stinner in bpo-21302.)
- On Windows 8.1 and newer, `time.sleep()` now uses a waitable timer based on high-resolution timers which has a resolution of 100 nanoseconds (10^{-7} seconds). Previously, it had a resolution of 1 millisecond (10^{-3} seconds). (Contributed by Benjamin Szőke, Dong-hee Na, Eryk Sun and Victor Stinner in bpo-21302 and bpo-45429.)

7.28 tkinter

- Added method `info_patchlevel()` which returns the exact version of the Tcl library as a named tuple similar to `sys.version_info`. (Contributed by Serhiy Storchaka in gh-91827.)

7.29 traceback

- Add `traceback.StackSummary.format_frame_summary()` to allow users to override which frames appear in the traceback, and how they are formatted. (Contributed by Ammar Askar in bpo-44569.)
- Add `traceback.TracebackException.print()`, which prints the formatted `TracebackException` instance to a file. (Contributed by Irit Katriel in bpo-33809.)

7.30 typing

For major changes, see *New Features Related to Type Hints*.

- Add `typing.assert_never()` and `typing.Never`. `typing.assert_never()` is useful for asking a type checker to confirm that a line of code is not reachable. At runtime, it raises an `AssertionError`. (Contributed by Jelle Zijlstra in gh-90633.)
- Add `typing.reveal_type()`. This is useful for asking a type checker what type it has inferred for a given expression. At runtime it prints the type of the received value. (Contributed by Jelle Zijlstra in gh-90572.)
- Add `typing.assert_type()`. This is useful for asking a type checker to confirm that the type it has inferred for a given expression matches the given type. At runtime it simply returns the received value. (Contributed by Jelle Zijlstra in gh-90638.)
- `typing.TypedDict` types can now be generic. (Contributed by Samodya Abeysiriwardane in gh-89026.)
- `NamedTuple` types can now be generic. (Contributed by Serhiy Storchaka in bpo-43923.)
- Allow subclassing of `typing.Any`. This is useful for avoiding type checker errors related to highly dynamic class, such as mocks. (Contributed by Shantanu Jain in gh-91154.)
- The `typing.final()` decorator now sets the `__final__` attributed on the decorated object. (Contributed by Jelle Zijlstra in gh-90500.)
- The `typing.get_overloads()` function can be used for introspecting the overloads of a function. `typing.clear_overloads()` can be used to clear all registered overloads of a function. (Contributed by Jelle Zijlstra in gh-89263.)
- The `__init__()` method of `Protocol` subclasses is now preserved. (Contributed by Adrian Garcia Badarasco in gh-88970.)
- The representation of empty tuple types (`Tuple[()]`) is simplified. This affects introspection, e.g. `get_args(Tuple[()])` now evaluates to `()` instead of `((),)`. (Contributed by Serhiy Storchaka in gh-91137.)
- Loosen runtime requirements for type annotations by removing the callable check in the private `typing._type_check` function. (Contributed by Gregory Beauregard in gh-90802.)

- `typing.get_type_hints()` now supports evaluating strings as forward references in PEP 585 generic aliases. (Contributed by Niklas Rosenstein in gh-85542.)
- `typing.get_type_hints()` no longer adds `Optional` to parameters with `None` as a default. (Contributed by Nikita Sobolev in gh-90353.)
- `typing.get_type_hints()` now supports evaluating bare stringified `ClassVar` annotations. (Contributed by Gregory Beauregard in gh-90711.)
- `typing.no_type_check()` no longer modifies external classes and functions. It also now correctly marks classmethods as not to be type checked. (Contributed by Nikita Sobolev in gh-90729.)

7.31 unicodedata

- The Unicode database has been updated to version 14.0.0. (Contributed by Benjamin Peterson in bpo-45190).

7.32 unittest

- Added methods `enterContext()` and `enterClassContext()` of class `TestCase`, method `enterAsyncContext()` of class `IsolatedAsyncioTestCase` and function `unittest.enterModuleContext()`. (Contributed by Serhiy Storchaka in bpo-45046.)

7.33 venv

- When new Python virtual environments are created, the *venv* sysconfig installation scheme is used to determine the paths inside the environment. When Python runs in a virtual environment, the same installation scheme is the default. That means that downstream distributors can change the default sysconfig install scheme without changing behavior of virtual environments. Third party code that also creates new virtual environments should do the same. (Contributed by Miro Hrončok in bpo-45413.)

7.34 warnings

- `warnings.catch_warnings()` now accepts arguments for `warnings.simplefilter()`, providing a more concise way to locally ignore warnings or convert them to errors. (Contributed by Zac Hatfield-Dodds in bpo-47074.)

7.35 zipfile

- Added support for specifying member name encoding for reading metadata in a `ZipFile`'s directory and file headers. (Contributed by Stephen J. Turnbull and Serhiy Storchaka in bpo-28080.)
- Added `ZipFile.mkdir()` for creating new directories inside ZIP archives. (Contributed by Sam Ezeh in gh-49083.)
- Added `stem`, `suffix` and `suffixes` to `zipfile.Path`. (Contributed by Miguel Brito in gh-88261.)

8 Optimizations

This section covers specific optimizations independent of the *Faster CPython* project, which is covered in its own section.

- The compiler now optimizes simple printf-style % formatting on string literals containing only the format codes `%s`, `%r` and `%a` and makes it as fast as a corresponding f-string expression. (Contributed by Serhiy Storchaka in bpo-28307.)
- Integer division (`//`) is better tuned for optimization by compilers. It is now around 20% faster on x86-64 when dividing an `int` by a value smaller than `2**30`. (Contributed by Gregory P. Smith and Tim Peters in gh-90564.)
- `sum()` is now nearly 30% faster for integers smaller than `2**30`. (Contributed by Stefan Behnel in gh-68264.)
- Resizing lists is streamlined for the common case, speeding up `list.append()` by ≈15% and simple list comprehensions by up to 20-30% (Contributed by Dennis Sweeney in gh-91165.)
- Dictionaries don't store hash values when all keys are Unicode objects, decreasing `dict` size. For example, `sys.getsizeof(dict.fromkeys("abcdefg"))` is reduced from 352 bytes to 272 bytes (23% smaller) on 64-bit platforms. (Contributed by Inada Naoki in bpo-46845.)
- Using `asyncio.DatagramProtocol` is now orders of magnitude faster when transferring large files over UDP, with speeds over 100 times higher for a ≈60 MiB file. (Contributed by msoxzw in gh-91487.)
- `math` functions `comb()` and `perm()` are now ≈10 times faster for large arguments (with a larger speedup for larger *k*). (Contributed by Serhiy Storchaka in bpo-37295.)
- The `statistics` functions `mean()`, `variance()` and `stdev()` now consume iterators in one pass rather than converting them to a `list` first. This is twice as fast and can save substantial memory. (Contributed by Raymond Hettinger in gh-90415.)
- `unicodedata.normalize()` now normalizes pure-ASCII strings in constant time. (Contributed by Dong-hee Na in bpo-44987.)

9 Faster CPython

CPython 3.11 is an average of 25% faster than CPython 3.10 as measured with the pyperformance benchmark suite, when compiled with GCC on Ubuntu Linux. Depending on your workload, the overall speedup could be 10-60%.

This project focuses on two major areas in Python: *Faster Startup* and *Faster Runtime*. Optimizations not covered by this project are listed separately under *Optimizations*.

9.1 Faster Startup

Frozen imports / Static code objects

Python caches bytecode in the __pycache__ directory to speed up module loading.

Previously in 3.10, Python module execution looked like this:

```
Read __pycache__ -> Unmarshal -> Heap allocated code object -> Evaluate
```

In Python 3.11, the core modules essential for Python startup are "frozen". This means that their codeobjects (and bytecode) are statically allocated by the interpreter. This reduces the steps in module execution process to:

```
Statically allocated code object -> Evaluate
```

Interpreter startup is now 10-15% faster in Python 3.11. This has a big impact for short-running programs using Python.

(Contributed by Eric Snow, Guido van Rossum and Kumar Aditya in many issues.)

9.2 Faster Runtime

Cheaper, lazy Python frames

Python frames, holding execution information, are created whenever Python calls a Python function. The following are new frame optimizations:

- Streamlined the frame creation process.
- Avoided memory allocation by generously re-using frame space on the C stack.
- Streamlined the internal frame struct to contain only essential information. Frames previously held extra debugging and memory management information.

Old-style frame objects are now created only when requested by debuggers or by Python introspection functions such as `sys._getframe()` and `inspect.currentframe()`. For most user code, no frame objects are created at all. As a result, nearly all Python functions calls have sped up significantly. We measured a 3-7% speedup in pyperformance.

(Contributed by Mark Shannon in bpo-44590.)

Inlined Python function calls

During a Python function call, Python will call an evaluating C function to interpret that function's code. This effectively limits pure Python recursion to what's safe for the C stack.

In 3.11, when CPython detects Python code calling another Python function, it sets up a new frame, and "jumps" to the new code inside the new frame. This avoids calling the C interpreting function altogether.

Most Python function calls now consume no C stack space, speeding them up. In simple recursive functions like fibonacci or factorial, we observed a 1.7x speedup. This also means recursive functions can recurse significantly deeper (if the user increases the recursion limit with `sys.setrecursionlimit()`). We measured a 1-3% improvement in pyperformance.

(Contributed by Pablo Galindo and Mark Shannon in bpo-45256.)

PEP 659: Specializing Adaptive Interpreter

PEP 659 is one of the key parts of the Faster CPython project. The general idea is that while Python is a dynamic language, most code has regions where objects and types rarely change. This concept is known as *type stability*.

At runtime, Python will try to look for common patterns and type stability in the executing code. Python will then replace the current operation with a more specialized one. This specialized operation uses fast paths available only to those use cases/types, which generally outperform their generic counterparts. This also brings in another concept called *inline caching*, where Python caches the results of expensive operations directly in the bytecode.

The specializer will also combine certain common instruction pairs into one superinstruction, reducing the overhead during execution.

Python will only specialize when it sees code that is "hot" (executed multiple times). This prevents Python from wasting time on run-once code. Python can also de-specialize when code is too dynamic or when the use changes. Specialization is attempted periodically, and specialization attempts are not too expensive, allowing specialization to adapt to new circumstances.

(PEP written by Mark Shannon, with ideas inspired by Stefan Brunthaler. See **PEP 659** for more information. Implementation by Mark Shannon and Brandt Bucher, with additional help from Irit Katriel and Dennis Sweeney.)

Operation	Form	Specialization	Operation speedup (up to)	Contributor(s)

Binary operations	`x + x`
	`x - x`
	`x * x`
		Binary add, multiply and subtract for common types such as `int`, `float` and `str` take custom fast paths for their underlying types.	10%	Mark Shannon, Dong-hee Na, Brandt Bucher, Dennis Sweeney

Subscript	`a[i]`	Subscripting container types such as `list`, `tuple` and `dict` directly index the underlying data structures. Subscripting custom `__getitem__()` is also inlined similar to *Inlined Python function calls*.	10-25%	Irit Katriel, Mark Shannon
Store subscript	`a[i] = z`	Similar to subscripting specialization above.	10-25%	Dennis Sweeney
Calls	`f(arg)` `C(arg)`	Calls to common builtin (C) functions and types such as `len()` and `str` directly call their underlying C version. This avoids going through the internal calling convention.	20%	Mark Shannon, Ken Jin
Load global variable	`print` `len`	The object's index in the globals/builtins namespace is cached. Loading globals and builtins require zero namespace lookups.	[1]	Mark Shannon
Load attribute	`o.attr`	Similar to loading global variables. The attribute's index inside the class/object's namespace is cached. In most cases, attribute loading will require zero namespace lookups.	[2]	Mark Shannon
Load methods for call	`o.meth()`	The actual address of the method is cached. Method loading now has no namespace lookups – even for classes with long inheritance chains.	10-20%	Ken Jin, Mark Shannon
Store attribute	`o.attr = z`	Similar to load attribute optimization.	2% in pyperformance	Mark Shannon
Unpack Sequence	`*seq`	Specialized for common containers such as `list` and `tuple`. Avoids internal calling convention.	8%	Brandt Bucher

9.3 Misc

- Objects now require less memory due to lazily created object namespaces. Their namespace dictionaries now also share keys more freely. (Contributed Mark Shannon in bpo-45340 and bpo-40116.)
- "Zero-cost" exceptions are implemented, eliminating the cost of `try` statements when no exception is raised. (Contributed by Mark Shannon in bpo-40222.)
- A more concise representation of exceptions in the interpreter reduced the time required for catching an exception by about 10%. (Contributed by Irit Katriel in bpo-45711.)
- `re`'s regular expression matching engine has been partially refactored, and now uses computed gotos (or

[1] A similar optimization already existed since Python 3.8. 3.11 specializes for more forms and reduces some overhead.

[2] A similar optimization already existed since Python 3.10. 3.11 specializes for more forms. Furthermore, all attribute loads should be sped up by bpo-45947.

"threaded code") on supported platforms. As a result, Python 3.11 executes the pyperformance regular expression benchmarks up to 10% faster than Python 3.10. (Contributed by Brandt Bucher in gh-91404.)

9.4 FAQ

How should I write my code to utilize these speedups?

Write Pythonic code that follows common best practices; you don't have to change your code. The Faster CPython project optimizes for common code patterns we observe.

Will CPython 3.11 use more memory?

Maybe not; we don't expect memory use to exceed 20% higher than 3.10. This is offset by memory optimizations for frame objects and object dictionaries as mentioned above.

I don't see any speedups in my workload. Why?

Certain code won't have noticeable benefits. If your code spends most of its time on I/O operations, or already does most of its computation in a C extension library like NumPy, there won't be significant speedups. This project currently benefits pure-Python workloads the most.

Furthermore, the pyperformance figures are a geometric mean. Even within the pyperformance benchmarks, certain benchmarks have slowed down slightly, while others have sped up by nearly 2x!

Is there a JIT compiler?

No. We're still exploring other optimizations.

9.5 About

Faster CPython explores optimizations for CPython. The main team is funded by Microsoft to work on this full-time. Pablo Galindo Salgado is also funded by Bloomberg LP to work on the project part-time. Finally, many contributors are volunteers from the community.

10 CPython bytecode changes

The bytecode now contains inline cache entries, which take the form of the newly-added `CACHE` instructions. Many opcodes expect to be followed by an exact number of caches, and instruct the interpreter to skip over them at runtime. Populated caches can look like arbitrary instructions, so great care should be taken when reading or modifying raw, adaptive bytecode containing quickened data.

10.1 New opcodes

- `ASYNC_GEN_WRAP`, `RETURN_GENERATOR` and `SEND`, used in generators and co-routines.
- `COPY_FREE_VARS`, which avoids needing special caller-side code for closures.
- `JUMP_BACKWARD_NO_INTERRUPT`, for use in certain loops where handling interrupts is undesirable.
- `MAKE_CELL`, to create cell-objects.
- `CHECK_EG_MATCH` and `PREP_RERAISE_STAR`, to handle the *new exception groups and except** added in **PEP 654**.

- PUSH_EXC_INFO, for use in exception handlers.
- RESUME, a no-op, for internal tracing, debugging and optimization checks.

10.2 Replaced opcodes

Replaced Opcode(s)	New Opcode(s)	Notes
BINARY_* INPLACE_*	BINARY_OP	Replaced all numeric binary/in-place opcodes with a single opcode
CALL_FUNCTION CALL_FUNCTION_KW CALL_METHOD	CALL KW_NAMES PRECALL PUSH_NULL	Decouples argument shifting for methods from handling of keyword arguments; allows better specialization of calls
DUP_TOP DUP_TOP_TWO ROT_TWO ROT_THREE ROT_FOUR ROT_N	COPY SWAP	Stack manipulation instructions
JUMP_IF_NOT_EXC_MATCH	CHECK_EXC_MATCH	Now performs check but doesn't jump
JUMP_ABSOLUTE POP_JUMP_IF_FALSE POP_JUMP_IF_TRUE	JUMP_BACKWARD POP_JUMP_BACKWARD_IF_* POP_JUMP_FORWARD_IF_*	See[3]; TRUE, FALSE, NONE and NOT_NONE variants for each direction
SETUP_WITH SETUP_ASYNC_WITH	BEFORE_WITH	with block setup

[3] All jump opcodes are now relative, including the existing JUMP_IF_TRUE_OR_POP and JUMP_IF_FALSE_OR_POP. The argument is now an offset from the current instruction rather than an absolute location.

10.3 Changed/removed opcodes

- Changed `MATCH_CLASS` and `MATCH_KEYS` to no longer push an additional boolean value to indicate success/failure. Instead, `None` is pushed on failure in place of the tuple of extracted values.
- Changed opcodes that work with exceptions to reflect them now being represented as one item on the stack instead of three (see gh-89874).
- Removed `COPY_DICT_WITHOUT_KEYS`, `GEN_START`, `POP_BLOCK`, `SETUP_FINALLY` and `YIELD_FROM`.

11 Deprecated

This section lists Python APIs that have been deprecated in Python 3.11.

Deprecated C APIs are *listed separately*.

11.1 Language/Builtins

- Chaining `classmethod` descriptors (introduced in bpo-19072) is now deprecated. It 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.)
- Octal escapes in string and bytes literals with values larger than `0o377` (255 in decimal) now produce a `DeprecationWarning`. In a future Python version, they will raise a `SyntaxWarning` and eventually a `SyntaxError`. (Contributed by Serhiy Storchaka in gh-81548.)
- The delegation of `int()` to `__trunc__()` is now deprecated. Calling `int(a)` when `type(a)` implements `__trunc__()` but not `__int__()` or `__index__()` now raises a `DeprecationWarning`. (Contributed by Zackery Spytz in bpo-44977.)

11.2 Modules

- **PEP 594** led to the deprecations of the following modules slated for removal in Python 3.13:

`aifc`	`chunk`	`msilib`	`pipes`	`telnetlib`
`audioop`	`crypt`	`nis`	`sndhdr`	`uu`
`cgi`	`imghdr`	`nntplib`	`spwd`	`xdrlib`
`cgitb`	`mailcap`	`ossaudiodev`	`sunau`	

  (Contributed by Brett Cannon in bpo-47061 and Victor Stinner in gh-68966.)
- The `asynchat`, `asyncore` and `smtpd` modules have been deprecated since at least Python 3.6. Their documentation and deprecation warnings have now been updated to note they will be removed in Python 3.12. (Contributed by Hugo van Kemenade in bpo-47022.)
- The `lib2to3` package and 2to3 tool are now deprecated and may not be able to parse Python 3.10 or newer. See **PEP 617**, introducing the new PEG parser, for details. (Contributed by Victor Stinner in bpo-40360.)
- Undocumented modules `sre_compile`, `sre_constants` and `sre_parse` are now deprecated. (Contributed by Serhiy Storchaka in bpo-47152.)

11.3 Standard Library

- The following have been deprecated in `configparser` since Python 3.2. Their deprecation warnings have now been updated to note they will be removed in Python 3.12:
  - the `configparser.SafeConfigParser` class
  - the `configparser.ParsingError.filename` property
  - the `configparser.RawConfigParser.readfp()` method

  (Contributed by Hugo van Kemenade in bpo-45173.)
- `configparser.LegacyInterpolation` has been deprecated in the docstring since Python 3.2, and is not listed in the `configparser` documentation. It now emits a `DeprecationWarning` and will be removed in Python 3.13. Use `configparser.BasicInterpolation` or `configparser.ExtendedInterpolation` instead. (Contributed by Hugo van Kemenade in bpo-46607.)
- The older set of `importlib.resources` functions were deprecated in favor of the replacements added in Python 3.9 and will be removed in a future Python version, due to not supporting resources located within package subdirectories:
  - `importlib.resources.contents()`
  - `importlib.resources.is_resource()`
  - `importlib.resources.open_binary()`
  - `importlib.resources.open_text()`
  - `importlib.resources.read_binary()`
  - `importlib.resources.read_text()`
  - `importlib.resources.path()`
- The `locale.getdefaultlocale()` function is deprecated and will be removed in Python 3.13. Use `locale.setlocale()`, `locale.getpreferredencoding(False)` and `locale.getlocale()` functions instead. (Contributed by Victor Stinner in gh-90817.)
- The `locale.resetlocale()` function is deprecated and will be removed in Python 3.13. Use `locale.setlocale(locale.LC_ALL, "")` instead. (Contributed by Victor Stinner in gh-90817.)
- Stricter rules will now be applied for numerical group references and group names in regular expressions. Only sequences of ASCII digits will now be accepted as a numerical reference, and the group name in `bytes` patterns and replacement strings can only contain ASCII letters, digits and underscores. For now, a deprecation warning is raised for syntax violating these rules. (Contributed by Serhiy Storchaka in gh-91760.)
- In the `re` module, the `re.template()` function and the corresponding `re.TEMPLATE` and `re.T` flags are deprecated, as they were undocumented and lacked an obvious purpose. They will be removed in Python 3.13. (Contributed by Serhiy Storchaka and Miro Hrončok in gh-92728.)
- `turtle.settiltangle()` has been deprecated since Python 3.1; it now emits a deprecation warning and will be removed in Python 3.13. Use `turtle.tiltangle()` instead (it was earlier incorrectly marked as deprecated, and its docstring is now corrected). (Contributed by Hugo van Kemenade in bpo-45837.)
- `typing.Text`, which exists solely to provide compatibility support between Python 2 and Python 3 code, is now deprecated. Its removal is currently unplanned, but users are encouraged to use `str` instead wherever possible. (Contributed by Alex Waygood in gh-92332.)
- The keyword argument syntax for constructing `typing.TypedDict` types is now deprecated. Support will be removed in Python 3.13. (Contributed by Jingchen Ye in gh-90224.)
- `webbrowser.MacOSX` is deprecated and will be removed in Python 3.13. It is untested, undocumented, and not used by `webbrowser` itself. (Contributed by Dong-hee Na in bpo-42255.)
- The behavior of returning a value from a `TestCase` and `IsolatedAsyncioTestCase` test methods (other than the default `None` value) is now deprecated.

- Deprecated the following not-formally-documented `unittest` functions, scheduled for removal in Python 3.13:
  - `unittest.findTestCases()`
  - `unittest.makeSuite()`
  - `unittest.getTestCaseNames()`

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

  (Contributed by Erlend E. Aasland in bpo-5846.)

12 Pending Removal in Python 3.12

The following Python APIs have been deprecated in earlier Python releases, and will be removed in Python 3.12.

C APIs pending removal are *listed separately*.

- The `asynchat` module
- The `asyncore` module
- The entire distutils package
- The `imp` module
- The `typing.io` namespace
- The `typing.re` namespace
- `cgi.log()`
- `importlib.find_loader()`
- `importlib.abc.Loader.module_repr()`
- `importlib.abc.MetaPathFinder.find_module()`
- `importlib.abc.PathEntryFinder.find_loader()`
- `importlib.abc.PathEntryFinder.find_module()`
- `importlib.machinery.BuiltinImporter.find_module()`
- `importlib.machinery.BuiltinLoader.module_repr()`
- `importlib.machinery.FileFinder.find_loader()`
- `importlib.machinery.FileFinder.find_module()`
- `importlib.machinery.FrozenImporter.find_module()`
- `importlib.machinery.FrozenLoader.module_repr()`
- `importlib.machinery.PathFinder.find_module()`
- `importlib.machinery.WindowsRegistryFinder.find_module()`
- `importlib.util.module_for_loader()`
- `importlib.util.set_loader_wrapper()`
- `importlib.util.set_package_wrapper()`
- `pkgutil.ImpImporter`

- `pkgutil.ImpLoader`
- `pathlib.Path.link_to()`
- `sqlite3.enable_shared_cache()`
- `sqlite3.OptimizedUnicode()`
- `PYTHONTHREADDEBUG` environment variable
- The following deprecated aliases in `unittest`:

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

13 Removed

This section lists Python APIs that have been removed in Python 3.11.

Removed C APIs are *listed separately*.

- Removed the `@asyncio.coroutine()` decorator enabling legacy generator-based coroutines to be compatible with `async` / `await` code. The function has been deprecated since Python 3.8 and the removal was initially scheduled for Python 3.10. Use `async def` instead. (Contributed by Illia Volochii in bpo-43216.)
- Removed `asyncio.coroutines.CoroWrapper` used for wrapping legacy generator-based coroutine objects in the debug mode. (Contributed by Illia Volochii in bpo-43216.)
- Due to significant security concerns, the *reuse_address* parameter of `asyncio.loop.create_datagram_endpoint()`, disabled in Python 3.9, is now entirely removed. This is because of the behavior of the socket option `SO_REUSEADDR` in UDP. (Contributed by Hugo van Kemenade in bpo-45129.)
- Removed the `binhex` module, deprecated in Python 3.9. Also removed the related, similarly-deprecated `binascii` functions:
  - `binascii.a2b_hqx()`
  - `binascii.b2a_hqx()`
  - `binascii.rlecode_hqx()`
  - `binascii.rldecode_hqx()`

  The `binascii.crc_hqx()` function remains available.

  (Contributed by Victor Stinner in bpo-45085.)
- Removed the `distutils` `bdist_msi` command deprecated in Python 3.9. Use `bdist_wheel` (wheel packages) instead. (Contributed by Hugo van Kemenade in bpo-45124.)

- Removed the `__getitem__()` methods of `xml.dom.pulldom.DOMEventStream`, `wsgiref.util.FileWrapper` and `fileinput.FileInput`, deprecated since Python 3.9. (Contributed by Hugo van Kemenade in bpo-45132.)
- Removed the deprecated `gettext` functions `lgettext()`, `ldgettext()`, `lngettext()` and `ldngettext()`. Also removed the `bind_textdomain_codeset()` function, the `NullTranslations.output_charset()` and `NullTranslations.set_output_charset()` methods, and the *codeset* parameter of `translation()` and `install()`, since they are only used for the `l*gettext()` functions. (Contributed by Dong-hee Na and Serhiy Storchaka in bpo-44235.)
- Removed from the `inspect` module:
  - The `getargspec()` function, deprecated since Python 3.0; use `inspect.signature()` or `inspect.getfullargspec()` instead.
  - The `formatargspec()` function, deprecated since Python 3.5; use the `inspect.signature()` function or the `inspect.Signature` object directly.
  - The undocumented `Signature.from_builtin()` and `Signature.from_function()` methods, deprecated since Python 3.5; use the `Signature.from_callable()` method instead.

  (Contributed by Hugo van Kemenade in bpo-45320.)
- Removed the `__class_getitem__()` method from `pathlib.PurePath`, because it was not used and added by mistake in previous versions. (Contributed by Nikita Sobolev in bpo-46483.)
- Removed the `MailmanProxy` class in the `smtpd` module, as it is unusable without the external `mailman` package. (Contributed by Dong-hee Na in bpo-35800.)
- Removed the deprecated `split()` method of `_tkinter.TkappType`. (Contributed by Erlend E. Aasland in bpo-38371.)
- Removed namespace package support from `unittest` discovery. It was introduced in Python 3.4 but has been broken since Python 3.7. (Contributed by Inada Naoki in bpo-23882.)
- Removed the undocumented private `float.__set_format__()` method, previously known as `float.__setformat__()` in Python 3.7. Its docstring said: "You probably don't want to use this function. It exists mainly to be used in Python's test suite." (Contributed by Victor Stinner in bpo-46852.)
- The `--experimental-isolated-subinterpreters` configure flag (and corresponding `EXPERIMENTAL_ISOLATED_SUBINTERPRETERS` macro) have been removed.
- Pynche — The Pythonically Natural Color and Hue Editor — has been moved out of `Tools/scripts` and is being developed independently from the Python source tree.

14 Porting to Python 3.11

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

Porting notes for the C API are *listed separately*.

- `open()`, `io.open()`, `codecs.open()` and `fileinput.FileInput` no longer accept `'U'` ("universal newline") in the file mode. In Python 3, "universal newline" mode is used by default whenever a file is opened in text mode, and the `'U'` flag has been deprecated since Python 3.3. The newline parameter to these functions controls how universal newlines work. (Contributed by Victor Stinner in bpo-37330.)
- `ast.AST` node positions are now validated when provided to `compile()` and other related functions. If invalid positions are detected, a `ValueError` will be raised. (Contributed by Pablo Galindo in gh-93351)
- Prohibited passing non-`concurrent.futures.ThreadPoolExecutor` executors to `asyncio.loop.set_default_executor()` following a deprecation in Python 3.8. (Contributed by Illia Volochii in bpo-43234.)

- `calendar`: The `calendar.LocaleTextCalendar` and `calendar.LocaleHTMLCalendar` classes now use `locale.getlocale()`, instead of using `locale.getdefaultlocale()`, if no locale is specified. (Contributed by Victor Stinner in bpo-46659.)
- The `pdb` module now reads the `.pdbrc` configuration file with the `'UTF-8'` encoding. (Contributed by Srinivas Reddy Thatiparthy (□□□□□□□□□□□ □□□□□□ □□□□□□□□□□) in bpo-41137.)
- The *population* parameter of `random.sample()` must be a sequence, and automatic conversion of `sets` to `lists` is no longer supported. Also, if the sample size is larger than the population size, a `ValueError` is raised. (Contributed by Raymond Hettinger in bpo-40465.)
- The *random* optional parameter of `random.shuffle()` was removed. It was previously an arbitrary random function to use for the shuffle; now, `random.random()` (its previous default) will always be used.
- In `re` re-syntax, global inline flags (e.g. `(?i)`) can now only be used at the start of regular expressions. Using them elsewhere has been deprecated since Python 3.6. (Contributed by Serhiy Storchaka in bpo-47066.)
- In the `re` module, several long-standing bugs where fixed that, in rare cases, could cause capture groups to get the wrong result. Therefore, this could change the captured output in these cases. (Contributed by Ma Lin in bpo-35859.)

15 Build Changes

- CPython now has **PEP 11 Tier 3 support** for cross compiling to the WebAssembly platforms Emscripten (`wasm32-unknown-emscripten`, i.e. Python in the browser) and WebAssembly System Interface (WASI) (`wasm32-unknown-wasi`). The effort is inspired by previous work like Pyodide. These platforms provide a limited subset of POSIX APIs; Python standard libraries features and modules related to networking, processes, threading, signals, mmap, and users/groups are not available or don't work. (Emscripten contributed by Christian Heimes and Ethan Smith in gh-84461 and WASI contributed by Christian Heimes in gh-90473; platforms promoted in gh-95085)
- Building CPython now requires:
  - A C11 compiler and standard library. Optional C11 features are not required. (Contributed by Victor Stinner in bpo-46656, bpo-45440 and bpo-46640.)
  - Support for IEEE 754 floating point numbers. (Contributed by Victor Stinner in bpo-46917.)
- The `Py_NO_NAN` macro has been removed. Since CPython now requires IEEE 754 floats, NaN values are always available. (Contributed by Victor Stinner in bpo-46656.)
- The `tkinter` package now requires Tcl/Tk version 8.5.12 or newer. (Contributed by Serhiy Storchaka in bpo-46996.)
- Build dependencies, compiler flags, and linker flags for most stdlib extension modules are now detected by **`configure`**. libffi, libnsl, libsqlite3, zlib, bzip2, liblzma, libcrypt, Tcl/Tk, and uuid flags are detected by pkg-config (when available). `tkinter` now requires a pkg-config command to detect development settings for Tcl/Tk headers and libraries. (Contributed by Christian Heimes and Erlend Egeberg Aasland in bpo-45847, bpo-45747, and bpo-45763.)
- libpython is no longer linked against libcrypt. (Contributed by Mike Gilbert in bpo-45433.)
- CPython can now be built with the ThinLTO option via passing `thin` to `--with-lto`, i.e. `--with-lto=thin`. (Contributed by Dong-hee Na and Brett Holman in bpo-44340.)
- Freelists for object structs can now be disabled. A new **`configure`** option `--without-freelists` can be used to disable all freelists except empty tuple singleton. (Contributed by Christian Heimes in bpo-45522.)
- `Modules/Setup` and `Modules/makesetup` have been improved and tied up. Extension modules can now be built through `makesetup`. All except some test modules can be linked statically into a main binary or library. (Contributed by Brett Cannon and Christian Heimes in bpo-45548, bpo-45570, bpo-45571, and bpo-43974.)

Note: Use the environment variables `TCLTK_CFLAGS` and `TCLTK_LIBS` to manually specify the location of Tcl/Tk headers and libraries. The **`configure`** options `--with-tcltk-includes` and `--with-tcltk-libs` have been removed.

On RHEL 7 and CentOS 7 the development packages do not provide `tcl.pc` and `tk.pc`; use `TCLTK_LIBS="-ltk8.5 -ltkstub8.5 -ltcl8.5"`. The directory `Misc/rhel7` contains `.pc` files and instructions on how to build Python with RHEL 7's and CentOS 7's Tcl/Tk and OpenSSL.

- CPython will now use 30-bit digits by default for the Python `int` implementation. Previously, the default was to use 30-bit digits on platforms with `SIZEOF_VOID_P >= 8`, and 15-bit digits otherwise. It's still possible to explicitly request use of 15-bit digits via either the `--enable-big-digits` option to the configure script or (for Windows) the `PYLONG_BITS_IN_DIGIT` variable in `PC/pyconfig.h`, but this option may be removed at some point in the future. (Contributed by Mark Dickinson in bpo-45569.)

16 C API Changes

16.1 New Features

- Add a new `PyType_GetName()` function to get type's short name. (Contributed by Hai Shi in bpo-42035.)
- Add a new `PyType_GetQualName()` function to get type's qualified name. (Contributed by Hai Shi in bpo-42035.)
- Add new `PyThreadState_EnterTracing()` and `PyThreadState_LeaveTracing()` functions to the limited C API to suspend and resume tracing and profiling. (Contributed by Victor Stinner in bpo-43760.)
- Added the `Py_Version` constant which bears the same value as `PY_VERSION_HEX`. (Contributed by Gabriele N. Tornetta in bpo-43931.)
- `Py_buffer` and APIs are now part of the limited API and the stable ABI:
  - `PyObject_CheckBuffer()`
  - `PyObject_GetBuffer()`
  - `PyBuffer_GetPointer()`
  - `PyBuffer_SizeFromFormat()`
  - `PyBuffer_ToContiguous()`
  - `PyBuffer_FromContiguous()`
  - `PyBuffer_CopyData()`
  - `PyBuffer_IsContiguous()`
  - `PyBuffer_FillContiguousStrides()`
  - `PyBuffer_FillInfo()`
  - `PyBuffer_Release()`
  - `PyMemoryView_FromBuffer()`
  - `bf_getbuffer` and `bf_releasebuffer` type slots

  (Contributed by Christian Heimes in bpo-45459.)
- Added the `PyType_GetModuleByDef` function, used to get the module in which a method was defined, in cases where this information is not available directly (via `PyCMethod`). (Contributed by Petr Viktorin in bpo-46613.)

- Add new functions to pack and unpack C double (serialize and deserialize): `PyFloat_Pack2()`, `PyFloat_Pack4()`, `PyFloat_Pack8()`, `PyFloat_Unpack2()`, `PyFloat_Unpack4()` and `PyFloat_Unpack8()`. (Contributed by Victor Stinner in bpo-46906.)
- Add new functions to get frame object attributes: `PyFrame_GetBuiltins()`, `PyFrame_GetGenerator()`, `PyFrame_GetGlobals()`, `PyFrame_GetLasti()`.
- Added two new functions to get and set the active exception instance: `PyErr_GetHandledException()` and `PyErr_SetHandledException()`. These are alternatives to `PyErr_SetExcInfo()` and `PyErr_GetExcInfo()` which work with the legacy 3-tuple representation of exceptions. (Contributed by Irit Katriel in bpo-46343.)
- Added the `PyConfig.safe_path` member. (Contributed by Victor Stinner in gh-57684.)

16.2 Porting to Python 3.11

- Some macros have been converted to static inline functions to avoid macro pitfalls. The change should be mostly transparent to users, as the replacement functions will cast their arguments to the expected types to avoid compiler warnings due to static type checks. However, when the limited C API is set to >=3.11, these casts are not done, and callers will need to cast arguments to their expected types. See **PEP 670** for more details. (Contributed by Victor Stinner and Erlend E. Aasland in gh-89653.)
- `PyErr_SetExcInfo()` no longer uses the `type` and `traceback` arguments, the interpreter now derives those values from the exception instance (the `value` argument). The function still steals references of all three arguments. (Contributed by Irit Katriel in bpo-45711.)
- `PyErr_GetExcInfo()` now derives the `type` and `traceback` fields of the result from the exception instance (the `value` field). (Contributed by Irit Katriel in bpo-45711.)
- `_frozen` has a new `is_package` field to indicate whether or not the frozen module is a package. Previously, a negative value in the `size` field was the indicator. Now only non-negative values be used for `size`. (Contributed by Kumar Aditya in bpo-46608.)
- `_PyFrameEvalFunction()` now takes `_PyInterpreterFrame*` as its second parameter, instead of `PyFrameObject*`. See **PEP 523** for more details of how to use this function pointer type.
- `PyCode_New()` and `PyCode_NewWithPosOnlyArgs()` now take an additional `exception_table` argument. Using these functions should be avoided, if at all possible. To get a custom code object: create a code object using the compiler, then get a modified version with the `replace` method.
- `PyCodeObject` no longer has the `co_code`, `co_varnames`, `co_cellvars` and `co_freevars` fields. Instead, use `PyCode_GetCode()`, `PyCode_GetVarnames()`, `PyCode_GetCellvars()` and `PyCode_GetFreevars()` respectively to access them via the C API. (Contributed by Brandt Bucher in bpo-46841 and Ken Jin in gh-92154 and gh-94936.)
- The old trashcan macros (`Py_TRASHCAN_SAFE_BEGIN`/`Py_TRASHCAN_SAFE_END`) are now deprecated. 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.

To support older Python versions in the same codebase, you can define the following macros and use them throughout the code (credit: these were copied from the `mypy` codebase):

```
#if PY_VERSION_HEX >= 0x03080000
#  define CPy_TRASHCAN_BEGIN(op, dealloc) Py_TRASHCAN_BEGIN(op, dealloc)
#  define CPy_TRASHCAN_END(op) Py_TRASHCAN_END
#else
#  define CPy_TRASHCAN_BEGIN(op, dealloc) Py_TRASHCAN_SAFE_BEGIN(op)
#  define CPy_TRASHCAN_END(op) Py_TRASHCAN_SAFE_END(op)
#endif
```

- The `PyType_Ready()` function now raises an error if a type is defined with the `Py_TPFLAGS_HAVE_GC` flag set but has no traverse function (`PyTypeObject.tp_traverse`). (Contributed by Victor Stinner in bpo-44263.)
- Heap types with the `Py_TPFLAGS_IMMUTABLETYPE` flag can now inherit the **PEP 590** vectorcall protocol. Previously, this was only possible for static types. (Contributed by Erlend E. Aasland in bpo-43908)
- Since `Py_TYPE()` is changed to a inline static function, `Py_TYPE(obj) = new_type` must be replaced with `Py_SET_TYPE(obj, new_type)`: see the `Py_SET_TYPE()` function (available since Python 3.9). For backward compatibility, this macro can be used:

```
#if PY_VERSION_HEX < 0x030900A4 && !defined(Py_SET_TYPE)
static inline void _Py_SET_TYPE(PyObject *ob, PyTypeObject *type)
{ ob->ob_type = type; }
#define Py_SET_TYPE(ob, type) _Py_SET_TYPE((PyObject*)(ob), type)
#endif
```

(Contributed by Victor Stinner in bpo-39573.)

- Since `Py_SIZE()` is changed to a inline static function, `Py_SIZE(obj) = new_size` must be replaced with `Py_SET_SIZE(obj, new_size)`: see the `Py_SET_SIZE()` function (available since Python 3.9). For backward compatibility, this macro can be used:

```
#if PY_VERSION_HEX < 0x030900A4 && !defined(Py_SET_SIZE)
static inline void _Py_SET_SIZE(PyVarObject *ob, Py_ssize_t size)
{ ob->ob_size = size; }
#define Py_SET_SIZE(ob, size) _Py_SET_SIZE((PyVarObject*)(ob), size)
#endif
```

(Contributed by Victor Stinner in bpo-39573.)

- `<Python.h>` no longer includes the header files `<stdlib.h>`, `<stdio.h>`, `<errno.h>` and `<string.h>` when the `Py_LIMITED_API` macro is set to `0x030b0000` (Python 3.11) or higher. C extensions should explicitly include the header files after `#include <Python.h>`. (Contributed by Victor Stinner in bpo-45434.)
- The non-limited API files `cellobject.h`, `classobject.h`, `code.h`, `context.h`, `funcobject.h`, `genobject.h` and `longintrepr.h` have been moved to the `Include/cpython` directory. Moreover, the `eval.h` header file was removed. These files must not be included directly, as they are already included in `Python.h`: Include Files. If they have been included directly, consider including `Python.h` instead. (Contributed by Victor Stinner in bpo-35134.)

- The `PyUnicode_CHECK_INTERNED()` macro has been excluded from the limited C API. It was never usable there, because it used internal structures which are not available in the limited C API. (Contributed by Victor Stinner in bpo-46007.)
- The following frame functions and type are now directly available with `#include <Python.h>`, it's no longer needed to add `#include <frameobject.h>`:
  - `PyFrame_Check()`
  - `PyFrame_GetBack()`
  - `PyFrame_GetBuiltins()`
  - `PyFrame_GetGenerator()`
  - `PyFrame_GetGlobals()`
  - `PyFrame_GetLasti()`
  - `PyFrame_GetLocals()`
  - `PyFrame_Type`

  (Contributed by Victor Stinner in gh-93937.)
- The `PyFrameObject` structure members have been removed from the public C API.

  While the documentation notes that the `PyFrameObject` fields are subject to change at any time, they have been stable for a long time and were used in several popular extensions.

  In Python 3.11, the frame struct was reorganized to allow performance optimizations. Some fields were removed entirely, as they were details of the old implementation.

  `PyFrameObject` fields:
  - `f_back`: use `PyFrame_GetBack()`.
  - `f_blockstack`: removed.
  - `f_builtins`: use `PyFrame_GetBuiltins()`.
  - `f_code`: use `PyFrame_GetCode()`.
  - `f_gen`: use `PyFrame_GetGenerator()`.
  - `f_globals`: use `PyFrame_GetGlobals()`.
  - `f_iblock`: removed.
  - `f_lasti`: use `PyFrame_GetLasti()`. Code using `f_lasti` with `PyCode_Addr2Line()` should use `PyFrame_GetLineNumber()` instead; it may be faster.
  - `f_lineno`: use `PyFrame_GetLineNumber()`
  - `f_locals`: use `PyFrame_GetLocals()`.
  - `f_stackdepth`: removed.
  - `f_state`: no public API (renamed to `f_frame.f_state`).
  - `f_trace`: no public API.
  - `f_trace_lines`: use `PyObject_GetAttrString((PyObject*)frame, "f_trace_lines")`.
  - `f_trace_opcodes`: use `PyObject_GetAttrString((PyObject*)frame, "f_trace_opcodes")`.
  - `f_localsplus`: no public API (renamed to `f_frame.localsplus`).
  - `f_valuestack`: removed.

The Python frame object is now created lazily. A side effect is that the `f_back` member must not be accessed directly, since its value is now also computed lazily. The `PyFrame_GetBack()` function must be called instead.

Debuggers that accessed the `f_locals` directly *must* call `PyFrame_GetLocals()` instead. They no longer need to call `PyFrame_FastToLocalsWithError()` or `PyFrame_LocalsToFast()`, in fact they should not call those functions. The necessary updating of the frame is now managed by the virtual machine.

Code defining `PyFrame_GetCode()` on Python 3.8 and older:

```
#if PY_VERSION_HEX < 0x030900B1
static inline PyCodeObject* PyFrame_GetCode(PyFrameObject *frame)
{
    Py_INCREF(frame->f_code);
    return frame->f_code;
}
#endif
```

Code defining `PyFrame_GetBack()` on Python 3.8 and older:

```
#if PY_VERSION_HEX < 0x030900B1
static inline PyFrameObject* PyFrame_GetBack(PyFrameObject *frame)
{
    Py_XINCREF(frame->f_back);
    return frame->f_back;
}
#endif
```

Or use the pythoncapi_compat project to get these two functions on older Python versions.

- Changes of the `PyThreadState` structure members:
  - `frame`: removed, use `PyThreadState_GetFrame()` (function added to Python 3.9 by bpo-40429). Warning: the function returns a strong reference, need to call `Py_XDECREF()`.
  - `tracing`: changed, use `PyThreadState_EnterTracing()` and `PyThreadState_LeaveTracing()` (functions added to Python 3.11 by bpo-43760).
  - `recursion_depth`: removed, use `(tstate->recursion_limit - tstate->recursion_remaining)` instead.
  - `stackcheck_counter`: removed.

Code defining `PyThreadState_GetFrame()` on Python 3.8 and older:

```
#if PY_VERSION_HEX < 0x030900B1
static inline PyFrameObject* PyThreadState_GetFrame(PyThreadState *tstate)
{
    Py_XINCREF(tstate->frame);
    return tstate->frame;
}
#endif
```

Code defining `PyThreadState_EnterTracing()` and `PyThreadState_LeaveTracing()` on Python 3.10 and older:

```
#if PY_VERSION_HEX < 0x030B00A2
static inline void PyThreadState_EnterTracing(PyThreadState *tstate)
{
    tstate->tracing++;
#if PY_VERSION_HEX >= 0x030A00A1
    tstate->cframe->use_tracing = 0;
#else
```

(continues on next page)

(continued from previous page)

```
    tstate->use_tracing = 0;
#endif
}

static inline void PyThreadState_LeaveTracing(PyThreadState *tstate)
{
    int use_tracing = (tstate->c_tracefunc != NULL || tstate->c_profilefunc !=␣
↪NULL);
    tstate->tracing--;
#if PY_VERSION_HEX >= 0x030A00A1
    tstate->cframe->use_tracing = use_tracing;
#else
    tstate->use_tracing = use_tracing;
#endif
}
#endif
```

Or use the pythoncapi_compat project to get these functions on old Python functions.

- Distributors are encouraged to build Python with the optimized Blake2 library libb2.
- The `PyConfig.module_search_paths_set` field must now be set to 1 for initialization to use `PyConfig.module_search_paths` to initialize `sys.path`. Otherwise, initialization will recalculate the path and replace any values added to `module_search_paths`.
- `PyConfig_Read()` no longer calculates the initial search path, and will not fill any values into `PyConfig.module_search_paths`. To calculate default paths and then modify them, finish initialization and use `PySys_GetObject()` to retrieve `sys.path` as a Python list object and modify it directly.

16.3 Deprecated

- Deprecate the following functions to configure the Python initialization:
  - `PySys_AddWarnOptionUnicode()`
  - `PySys_AddWarnOption()`
  - `PySys_AddXOption()`
  - `PySys_HasWarnOptions()`
  - `PySys_SetArgvEx()`
  - `PySys_SetArgv()`
  - `PySys_SetPath()`
  - `Py_SetPath()`
  - `Py_SetProgramName()`
  - `Py_SetPythonHome()`
  - `Py_SetStandardStreamEncoding()`
  - `_Py_SetProgramFullPath()`

  Use the new `PyConfig` API of the Python Initialization Configuration instead (**PEP 587**). (Contributed by Victor Stinner in gh-88279.)
- Deprecate the `ob_shash` member of the `PyBytesObject`. Use `PyObject_Hash()` instead. (Contributed by Inada Naoki in bpo-46864.)

16.4 Pending Removal in Python 3.12

The following C APIs have been deprecated in earlier Python releases, and will be removed in Python 3.12.

- `PyUnicode_AS_DATA()`
- `PyUnicode_AS_UNICODE()`
- `PyUnicode_AsUnicodeAndSize()`
- `PyUnicode_AsUnicode()`
- `PyUnicode_FromUnicode()`
- `PyUnicode_GET_DATA_SIZE()`
- `PyUnicode_GET_SIZE()`
- `PyUnicode_GetSize()`
- `PyUnicode_IS_COMPACT()`
- `PyUnicode_IS_READY()`
- `PyUnicode_READY()`
- `Py_UNICODE_WSTR_LENGTH()`
- `_PyUnicode_AsUnicode()`
- `PyUnicode_WCHAR_KIND`
- `PyUnicodeObject`
- `PyUnicode_InternImmortal()`

16.5 Removed

- `PyFrame_BlockSetup()` and `PyFrame_BlockPop()` have been removed. (Contributed by Mark Shannon in bpo-40222.)
- Remove the following math macros using the `errno` variable:
  - `Py_ADJUST_ERANGE1()`
  - `Py_ADJUST_ERANGE2()`
  - `Py_OVERFLOWED()`
  - `Py_SET_ERANGE_IF_OVERFLOW()`
  - `Py_SET_ERRNO_ON_MATH_ERROR()`

  (Contributed by Victor Stinner in bpo-45412.)
- Remove `Py_UNICODE_COPY()` and `Py_UNICODE_FILL()` macros, deprecated since Python 3.3. Use `PyUnicode_CopyCharacters()` or `memcpy()` (`wchar_t*` string), and `PyUnicode_Fill()` functions instead. (Contributed by Victor Stinner in bpo-41123.)
- Remove the `pystrhex.h` header file. It only contains private functions. C extensions should only include the main `<Python.h>` header file. (Contributed by Victor Stinner in bpo-45434.)
- Remove the `Py_FORCE_DOUBLE()` macro. It was used by the `Py_IS_INFINITY()` macro. (Contributed by Victor Stinner in bpo-45440.)
- The following items are no longer available when `Py_LIMITED_API` is defined:
  - `PyMarshal_WriteLongToFile()`
  - `PyMarshal_WriteObjectToFile()`
  - `PyMarshal_ReadObjectFromString()`

  - `PyMarshal_WriteObjectToString()`
  - the `Py_MARSHAL_VERSION` macro

  These are not part of the limited API.

  (Contributed by Victor Stinner in bpo-45474.)

- Exclude `PyWeakref_GET_OBJECT()` from the limited C API. It never worked since the `PyWeakReference` structure is opaque in the limited C API. (Contributed by Victor Stinner in bpo-35134.)
- Remove the `PyHeapType_GET_MEMBERS()` macro. It was exposed in the public C API by mistake, it must only be used by Python internally. Use the `PyTypeObject.tp_members` member instead. (Contributed by Victor Stinner in bpo-40170.)
- Remove the `HAVE_PY_SET_53BIT_PRECISION` macro (moved to the internal C API). (Contributed by Victor Stinner in bpo-45412.)
- Remove the `Py_UNICODE` encoder APIs, as they have been deprecated since Python 3.3, are little used and are inefficient relative to the recommended alternatives.

  The removed functions are:

  - `PyUnicode_Encode()`
  - `PyUnicode_EncodeASCII()`
  - `PyUnicode_EncodeLatin1()`
  - `PyUnicode_EncodeUTF7()`
  - `PyUnicode_EncodeUTF8()`
  - `PyUnicode_EncodeUTF16()`
  - `PyUnicode_EncodeUTF32()`
  - `PyUnicode_EncodeUnicodeEscape()`
  - `PyUnicode_EncodeRawUnicodeEscape()`
  - `PyUnicode_EncodeCharmap()`
  - `PyUnicode_TranslateCharmap()`
  - `PyUnicode_EncodeDecimal()`
  - `PyUnicode_TransformDecimalToASCII()`

  See **PEP 624** for details and **migration guidance**. (Contributed by Inada Naoki in bpo-44029.)

Index

E

P