What's New in Python

Release 3.11.0b3

A. M. Kuchling

July 11, 2022

Python Software Foundation
Email: docs@python.org

Contents

Release 3.11.0b3

Date July 11, 2022

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

For full details, see the changelog.

Note: Prerelease users should be aware that this document is currently in draft form. It will be updated substantially as Python 3.11 moves towards release, so it's worth checking back even after reading earlier versions.

---

1 Summary – Release highlights

- Python 3.11 is up to 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*`. (Contributed by Irit Katriel in bpo-45292.)

New typing features:

- **PEP 646**: Variadic generics.
- **PEP 655**: Marking individual TypedDict items as required or potentially missing.
- **PEP 673**: `Self` type.
- **PEP 675**: Arbitrary literal string type.

Security improvements:

- New `-P` command line option and `PYTHONSAFEPATH` environment variable to not prepend a potentially unsafe path to `sys.path` such as the current directory, the script's directory or an empty string.

2 New Features

2.1 Enhanced 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)
                           ^^^^^^^^^
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 dictionary 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)
```

(continues on next page)

(continued from previous page)

```
                ~~~~~~^~~
ZeroDivisionError: division by zero
```

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 code objects which may result in a small increase of disk usage of compiled Python files or interpreter memory usage. To avoid storing the extra information and/or deactivate printing the extra traceback information, the `-X no_debug_ranges` command line flag or the `PYTHONNODEBUGRANGES` environment variable can be used.

Column information for code objects

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

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

The `-X no_debug_ranges` option and the environment variable `PYTHONNODEBUGRANGES` can be used to disable this feature.

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

Exceptions can be enriched with notes (PEP 678)

The `add_note()` method was added to `BaseException`. It can be used to enrich exceptions with context information which is not available at the time when the exception is raised. The notes added appear in the default traceback. See **PEP 678** for more details. (Contributed by Irit Katriel in bpo-45607.)

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 introduced `TypeVar`, enabling creation of generics parameterised with a single type. **PEP 646** introduces `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.

Fields are still required by default, unless the `total=False` parameter is set. For example, the following specifies a dictionary with one required and one not-required key:

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

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,
    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 `dataclasses.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, like this:
@create_model
class CustomerModel:
    id: int
    name: str

c = CustomerModel(id=327, name="John Smith")
```

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, `__future__.annotations` that was planned for this release has been indefinitely postponed. See this message for more information.

4 Other Language Changes

- Starred expressions can 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. (Contributed by Serhiy Storchaka in bpo-33346.)
- A `TypeError` is now raised instead of an `AttributeError` in `contextlib.ExitStack.enter_context()` and `contextlib.AsyncExitStack.enter_async_context()` for objects which do not support the context manager or asynchronous context manager protocols correspondingly. (Contributed by Serhiy Storchaka in bpo-44471.)
- A `TypeError` is now raised instead of an `AttributeError` in `with` and `async with` statements for objects which do not support the context manager or asynchronous context manager protocols correspondingly. (Contributed by Serhiy Storchaka in bpo-12022.)
- Added `object.__getstate__()` which provides the default implementation of the `__getstate__()` method. `Copying` and `pickling` 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.)
- Add `-P` command line option and `PYTHONSAFEPATH` environment variable to not prepend a potentially unsafe path to `sys.path` such as the current directory, the script's directory or an empty string. (Contributed by Victor Stinner in gh-57684.)
- A `"z"` option was added to the format specification mini-language that coerces negative zero to zero after rounding to the format precision. See **PEP 682** for more details. (Contributed by John Belmonte in gh-90153.)

5 Other CPython Implementation Changes

- Special methods `complex.__complex__()` and `bytes.__bytes__()` are implemented to support `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 algorithms. 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 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 (a.k.a exc_info, or _PyErr_StackItem) now has only the `exc_value` field, `exc_type` and `exc_traceback` have been removed as their values can be derived from `exc_value`. (Contributed by Irit Katriel in bpo-45711.)
- A new command line option for the Windows installer `AppendPath` has been added. It behaves similiar 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`.

6 New Modules

- A new module, `tomllib`, was added for parsing TOML. (Contributed by Taneli Hukkinen in bpo-40059.)
- `wsgiref.types`, containing WSGI-specific types for static type checking, was added. (Contributed by Sebastian Rittau in bpo-42012.)

7 Improved Modules

7.1 asyncio

- Add raw datagram socket functions to the event loop: `sock_sendto()`, `sock_recvfrom()` and `sock_recvfrom_into()`. (Contributed by Alex Grönholm in bpo-46805.)
- Add `start_tls()` method for upgrading existing stream-based connections to TLS. (Contributed by Ian Good in bpo-34975.)
- Add `Barrier` class to the synchronization primitives of the asyncio library. (Contributed by Yves Duprat and Andrew Svetlov in gh-87518.)
- Add `TaskGroup` class, an asynchronous context manager holding a group of tasks that will wait for all of them upon exit. (Contributed by Yury Seliganov and others.)

7.2 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.)

7.3 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.)

7.4 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=" ")
```

(continues on next page)

(continued from previous page)

```
...     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.)

7.5 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.)

7.6 IDLE and idlelib

- Apply syntax highlighting to *.pyi* files. (Contributed by Alex Waygood and Terry Jan Reedy in bpo-45447.)

7.7 inspect

- Add `inspect.getmembers_static()`: return all members without triggering dynamic lookup via the descriptor protocol. (Contributed by Weipeng Hong in bpo-30533.)
- Add `inspect.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 a regular object (that is backwards compatible with the old tuple-like interface) that include 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()` and `inspect.trace()`. (Contributed by Pablo Galindo in gh-88116)

7.8 locale

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

7.9 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.)

7.10 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.)

7.11 os

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

7.12 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.)

7.13 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.)

7.14 shutil

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

7.15 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).

7.16 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)

7.17 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.)

7.18 sysconfig

- Two 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.)

7.19 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.)

7.20 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.)

7.21 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.)

7.22 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.)

7.23 unicodedata

- The Unicode database has been updated to version 14.0.0. (bpo-45190).

7.24 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.)

7.25 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.)

7.26 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.)

7.27 zipfile

- Added support for specifying member name encoding for reading metadata in the zipfile's directory and file headers. (Contributed by Stephen J. Turnbull and Serhiy Storchaka in bpo-28080.)

7.28 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.

8 Optimizations

- Compiler now optimizes simple C-style formatting with literal format containing only format codes `%s`, `%r` and `%a` and makes it as fast as corresponding f-string expression. (Contributed by Serhiy Storchaka in bpo-28307.)
- "Zero-cost" exceptions are implemented. The cost of `try` statements is almost eliminated when no exception is raised. (Contributed by Mark Shannon in bpo-40222.)
- Pure ASCII strings are now normalized in constant time by `unicodedata.normalize()`. (Contributed by Dong-hee Na in bpo-44987.)
- `math` functions `comb()` and `perm()` are now up to 10 times or more faster for large arguments (the speed up is larger for larger *k*). (Contributed by Serhiy Storchaka in bpo-37295.)
- Dict don't store hash value when all inserted keys are Unicode objects. This reduces dict size. For example, `sys.getsizeof(dict.fromkeys("abcdefg"))` becomes 272 bytes from 352 bytes on 64bit platform. (Contributed by Inada Naoki in bpo-46845.)
- `re`'s regular expression matching engine has been partially refactored, and now uses computed gotos (or "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.

9 Faster CPython

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

This project focuses on two major areas in Python: faster startup and faster runtime. Other optimizations not under this project are listed in *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 code objects (and bytecode) are statically allocated by the interpreter. This reduces the steps in module execution process to this:

```
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 numerous issues.)

9.2 Faster Runtime

Cheaper, lazy Python frames

Python frames are created whenever Python calls a Python function. This frame holds execution information. 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` or `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. This speeds up most of such calls. In simple recursive functions like fibonacci or factorial, a 1.7x speedup was observed. This also means recursive functions can recurse significantly deeper (if the user increases the recursion limit). 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. This reduces the overhead during execution.

Python will only specialize when it sees code that is "hot" (executed multiple times). This prevents Python from wasting time for 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. This allows 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

[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.

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.)
- 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.)

9.4 FAQ

Q: How should I write my code to utilize these speedups?

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

Q: Will CPython 3.11 use more memory?

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

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

A: 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 speedup. 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!

Q: Is there a JIT compiler?

A: 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

- Replaced all numeric `BINARY_*` and `INPLACE_*` instructions with a single `BINARY_OP` implementation.
- Replaced the three call instructions: `CALL_FUNCTION`, `CALL_FUNCTION_KW` and `CALL_METHOD` with `PUSH_NULL`, `PRECALL`, `CALL`, and `KW_NAMES`. This decouples the argument shifting for methods from the handling of keyword arguments and allows better specialization of calls.
- Removed `COPY_DICT_WITHOUT_KEYS` and `GEN_START`.
- `MATCH_CLASS` and `MATCH_KEYS` no longer push an additional boolean value indicating whether the match succeeded or failed. Instead, they indicate failure with `None` (where a tuple of extracted values would otherwise be).
- Replace several stack manipulation instructions (`DUP_TOP`, `DUP_TOP_TWO`, `ROT_TWO`, `ROT_THREE`, `ROT_FOUR`, and `ROT_N`) with new `COPY` and `SWAP` instructions.
- Replaced `JUMP_IF_NOT_EXC_MATCH` by `CHECK_EXC_MATCH` which performs the check but does not jump.
- Replaced `JUMP_IF_NOT_EG_MATCH` by `CHECK_EG_MATCH` which performs the check but does not jump.
- Replaced `JUMP_ABSOLUTE` by the relative `JUMP_BACKWARD`.
- Added `JUMP_BACKWARD_NO_INTERRUPT`, which is used in certain loops where it is undesirable to handle interrupts.
- Replaced `POP_JUMP_IF_TRUE` and `POP_JUMP_IF_FALSE` by the relative `POP_JUMP_FORWARD_IF_TRUE`, `POP_JUMP_BACKWARD_IF_TRUE`, `POP_JUMP_FORWARD_IF_FALSE` and `POP_JUMP_BACKWARD_IF_FALSE`.
- Added `POP_JUMP_FORWARD_IF_NOT_NONE`, `POP_JUMP_BACKWARD_IF_NOT_NONE`, `POP_JUMP_FORWARD_IF_NONE` and `POP_JUMP_BACKWARD_IF_NONE` opcodes to speed up conditional jumps.
- `JUMP_IF_TRUE_OR_POP` and `JUMP_IF_FALSE_OR_POP` are now relative rather than absolute.

11 Deprecated

- 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 value larger than `0o377` now produce `DeprecationWarning`. In a future Python version they will be a `SyntaxWarning` and eventually a `SyntaxError`. (Contributed by Serhiy Storchaka in gh-81548.)
- The `lib2to3` package and `2to3` tool are now deprecated and may not be able to parse Python 3.10 or newer. See the **PEP 617** (New PEG parser for CPython). (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.)
- `webbrowser.MacOSX` is deprecated and will be removed in Python 3.13. It is untested and undocumented and also 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 `unittest` functions, scheduled for removal in Python 3.13:
  - `unittest.findTestCases()`
  - `unittest.makeSuite()`
  - `unittest.getTestCaseNames()`

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

  (Contributed by Erlend E. Aasland in bpo-5846.)
- The `turtle.RawTurtle.settiltangle()` is deprecated since Python 3.1, it now emits a deprecation warning and will be removed in Python 3.13. Use `turtle.RawTurtle.tiltangle()` instead (it was earlier incorrectly marked as deprecated, its docstring is now corrected). (Contributed by Hugo van Kemenade in bpo-45837.)
- 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.)
- The following have been deprecated in `configparser` since Python 3.2. Their deprecation warnings have now been updated to note they will 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. 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 `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.)
- 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 removed in Python 3.12 (**PEP 594**). (Contributed by Hugo van Kemenade in bpo-47022.)
- **PEP 594** led to the deprecations of the following modules which are slated for removal in Python 3.13:
  - `aifc`
  - `audioop`
  - `cgi`
  - `cgitb`

  - `chunk`
  - `crypt`
  - `imghdr`
  - `mailcap`
  - `msilib`
  - `nis`
  - `nntplib`
  - `ossaudiodev`
  - `pipes`
  - `sndhdr`
  - `spwd`
  - `sunau`
  - `telnetlib`
  - `uu`
  - `xdrlib`

  (Contributed by Brett Cannon in bpo-47061 and Victor Stinner in gh-68966.)

- More strict rules will be applied now applied for numerical group references and group names in regular expressions in future Python versions. Only sequence of ASCII digits will be now accepted as a numerical reference. The group name in bytes patterns and replacement strings could only contain ASCII letters and digits and underscore. For now, a deprecation warning is raised for such syntax. (Contributed by Serhiy Storchaka in gh-91760.)
- `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 `TypedDict` types is now deprecated. Support will be removed in Python 3.13. (Contributed by Jingchen Ye in gh-90224.)
- 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.)

12 Pending Removal in Python 3.12

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

Python API:

- `pkgutil.ImpImporter`
- `pkgutil.ImpLoader`
- `PYTHONTHREADDEBUG`
- `importlib.find_loader()`
- `importlib.util.module_for_loader()`
- `importlib.util.set_loader_wrapper()`
- `importlib.util.set_package_wrapper()`
- `importlib.abc.Loader.module_repr()`

- importlib.abc.Loadermodule_repr()
- importlib.abc.MetaPathFinder.find_module()
- 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()
- pathlib.Path.link_to()
- The entire distutils namespace
- cgi.log()
- sqlite3.OptimizedUnicode()
- sqlite3.enable_shared_cache()

C API:

- 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()

13 Removed

- `smtpd.MailmanProxy` is now removed as it is unusable without an external module, `mailman`. (Contributed by Dong-hee Na in bpo-35800.)
- The `binhex` module, deprecated in Python 3.9, is now removed. The following `binascii` functions, deprecated in Python 3.9, are now also removed:
  - `a2b_hqx()`, `b2a_hqx()`;
  - `rlecode_hqx()`, `rledecode_hqx()`.

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

  (Contributed by Victor Stinner in bpo-45085.)
- The distutils `bdist_msi` command, deprecated in Python 3.9, is now removed. Use `bdist_wheel` (wheel packages) instead. (Contributed by Hugo van Kemenade in bpo-45124.)
- 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 `__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.)
- The following deprecated functions and methods are removed in the `gettext` module: `lgettext()`, `ldgettext()`, `lngettext()` and `ldngettext()`.

  Function `bind_textdomain_codeset()`, methods `output_charset()` and `set_output_charset()`, and the *codeset* parameter of functions `translation()` and `install()` are also removed, since they are only used for the `l*gettext()` functions. (Contributed by Dong-hee Na and Serhiy Storchaka in bpo-44235.)
- 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.)
- `asyncio.coroutines.CoroWrapper` used for wrapping legacy generator-based coroutine objects in the debug mode. (Contributed by Illia Volochii in bpo-43216.)
- Removed the deprecated `split()` method of `_tkinter.TkappType`. (Contributed by Erlend E. Aasland in bpo-38371.)
- 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 and `Signature` object directly.
  - the undocumented `Signature.from_builtin` and `Signature.from_function` functions, deprecated since Python 3.5; use the `Signature.from_callable()` method instead.

  (Contributed by Hugo van Kemenade in bpo-45320.)
- Remove 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.)
- Remove `__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.)
- Remove 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`) have been removed.

14 Porting to Python 3.11

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

14.1 Changes in the Python API

- Prohibited passing non-`concurrent.futures.ThreadPoolExecutor` executors to `loop.set_default_executor()` following a deprecation in Python 3.8. (Contributed by Illia Volochii in bpo-43234.)
- `open()`, `io.open()`, `codecs.open()` and `fileinput.FileInput` no longer accept `'U'` ("universal newline") in the file mode. This flag was deprecated since Python 3.3. In Python 3, the "universal newline" is used by default when a file is open in text mode. The newline parameter of `open()` controls how universal newlines works. (Contributed by Victor Stinner in bpo-37330.)
- The `pdb` module now reads the `.pdbrc` configuration file with the `'utf-8'` encoding. (Contributed by Srinivas Reddy Thatiparthy (□□□□□□□□□□□ □□□□□□ □□□□□□□□□□) in bpo-41137.)
- When sorting using tuples as keys, the order of the result may differ from earlier releases if the tuple elements don't define a total ordering (see expressions-value-comparisons for information on total ordering). It's generally true that the result of sorting simply isn't well-defined in the absence of a total ordering on list elements.
- `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.)
- Global inline flags (e.g. `(?i)`) can now only be used at the start of the regular expressions. Using them not at the start of expression was deprecated since Python 3.6. (Contributed by Serhiy Storchaka in bpo-47066.)
- `re` module: Fix a few long-standing bugs where, in rare cases, capturing group could get wrong result. So the result may be different than before. (Contributed by Ma Lin in bpo-35859.)
- The *population* parameter of `random.sample()` must be a sequence. Automatic conversion of sets to lists is no longer supported. If the sample size is larger than the population size, a `ValueError` is raised. (Contributed by Raymond Hettinger in bpo-40465.)

15 Build Changes

- Building Python now requires a C11 compiler without optional C11 features. (Contributed by Victor Stinner in bpo-46656.)
- Building Python now requires support of IEEE 754 floating point numbers. (Contributed by Victor Stinner in bpo-46917.)
- CPython can now be built with the ThinLTO option via `--with-lto=thin`. (Contributed by Dong-hee Na and Brett Holman in bpo-44340.)
- libpython is no longer linked against libcrypt. (Contributed by Mike Gilbert in bpo-45433.)
- Building Python now requires a C99 `<math.h>` header file providing the following functions: `copysign()`, `hypot()`, `isfinite()`, `isinf()`, `isnan()`, `round()`. (Contributed by Victor Stinner in bpo-45440.)
- Building Python now requires a C99 `<math.h>` header file providing a `NAN` constant, or the `__builtin_nan()` built-in function. (Contributed by Victor Stinner in bpo-46640.)

- Building Python now requires support for floating point Not-a-Number (NaN): remove the `Py_NO_NAN` macro. (Contributed by Victor Stinner in bpo-46656.)
- 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 main binary or library. (Contributed by Brett Cannon and Christian Heimes in bpo-45548, bpo-45570, bpo-45571, and bpo-43974.)
- 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 libs, and uuid flags are detected by `pkg-config` (when available). (Contributed by Christian Heimes and Erlend Egeberg Aasland in bpo-45847, bpo-45747, and bpo-45763.)

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.

- CPython now has experimental support for cross compiling to WebAssembly platform `wasm32-emscripten`. The effort is inspired by previous work like Pyodide. (Contributed by Christian Heimes and Ethan Smith in bpo-40280.)
- 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.)
- The `tkinter` package now requires Tcl/Tk version 8.5.12 or newer. (Contributed by Serhiy Storchaka in bpo-46996.)

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

- `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 a `co_code` field. Instead, use `PyObject_GetAttrString(code_object, "co_code")` or `PyCode_GetCode()` to get the underlying bytes object. (Contributed by Brandt Bucher in bpo-46841 and Ken Jin in gh-92154.)
- 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);
```

(continues on next page)

(continued from previous page)

```
    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_MAJOR_VERSION >= 3 && PY_MINOR_VERSION >= 8
#  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
    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 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.)

Index

E

P