What's New in Python

Release 3.7.14

A. M. Kuchling

September 06, 2022

Python Software Foundation
Email: docs@python.org

Contents

Editor Elvis Pranskevichus <elvis@magic.io>

This article explains the new features in Python 3.7, compared to 3.6. Python 3.7 was released on June 27, 2018. For full details, see the changelog.

1 Summary – Release Highlights

New syntax features:

- *PEP 563*, postponed evaluation of type annotations.

Backwards incompatible syntax changes:

- `async` and `await` are now reserved keywords.

New library modules:

- `contextvars`: *PEP 567 – Context Variables*
- `dataclasses`: *PEP 557 – Data Classes*
- *importlib.resources*

New built-in features:

- *PEP 553*, the new `breakpoint()` function.

Python data model improvements:

- *PEP 562*, customization of access to module attributes.
- *PEP 560*, core support for typing module and generic types.
- the insertion-order preservation nature of dict objects has been declared to be an official part of the Python language spec.

Significant improvements in the standard library:

- The `asyncio` module has received new features, significant *usability and performance improvements*.
- The `time` module gained support for *functions with nanosecond resolution*.

CPython implementation improvements:

- Avoiding the use of ASCII as a default text encoding:
  - *PEP 538*, legacy C locale coercion
  - *PEP 540*, forced UTF-8 runtime mode
- *PEP 552*, deterministic .pycs
- *the new development runtime mode*
- *PEP 565*, improved `DeprecationWarning` handling

C API improvements:

- *PEP 539*, new C API for thread-local storage

Documentation improvements:

- *PEP 545*, Python documentation translations
- New documentation translations: Japanese, French, and Korean.

This release features notable performance improvements in many areas. The *Optimizations* section lists them in detail.

For a list of changes that may affect compatibility with previous Python releases please refer to the *Porting to Python 3.7* section.

2 New Features

2.1 PEP 563: Postponed Evaluation of Annotations

The advent of type hints in Python uncovered two glaring usability issues with the functionality of annotations added in **PEP 3107** and refined further in **PEP 526**:

- annotations could only use names which were already available in the current scope, in other words they didn't support forward references of any kind; and
- annotating source code had adverse effects on startup time of Python programs.

Both of these issues are fixed by postponing the evaluation of annotations. Instead of compiling code which executes expressions in annotations at their definition time, the compiler stores the annotation in a string form equivalent to the AST of the expression in question. If needed, annotations can be resolved at runtime using `typing.get_type_hints()`. In the common case where this is not required, the annotations are cheaper to store (since short strings are interned by the interpreter) and make startup time faster.

Usability-wise, annotations now support forward references, making the following syntax valid:

```
class C:
    @classmethod
    def from_string(cls, source: str) -> C:
        ...

    def validate_b(self, obj: B) -> bool:
        ...

class B:
    ...
```

Since this change breaks compatibility, the new behavior needs to be enabled on a per-module basis in Python 3.7 using a `__future__` import:

```
from __future__ import annotations
```

It will become the default in Python 3.10.

See also:

PEP 563 – Postponed evaluation of annotations PEP written and implemented by Łukasz Langa.

2.2 PEP 538: Legacy C Locale Coercion

An ongoing challenge within the Python 3 series has been determining a sensible default strategy for handling the "7-bit ASCII" text encoding assumption currently implied by the use of the default C or POSIX locale on non-Windows platforms.

PEP 538 updates the default interpreter command line interface to automatically coerce that locale to an available UTF-8 based locale as described in the documentation of the new `PYTHONCOERCECLOCALE` environment variable. Automatically setting `LC_CTYPE` this way means that both the core interpreter and locale-aware C extensions (such as `readline`) will assume the use of UTF-8 as the default text encoding, rather than ASCII.

The platform support definition in **PEP 11** has also been updated to limit full text handling support to suitably configured non-ASCII based locales.

As part of this change, the default error handler for `stdin` and `stdout` is now `surrogateescape` (rather than `strict`) when using any of the defined coercion target locales (currently `C.UTF-8`, `C.utf8`, and `UTF-8`). The default error handler for `stderr` continues to be `backslashreplace`, regardless of locale.

Locale coercion is silent by default, but to assist in debugging potentially locale related integration problems, explicit warnings (emitted directly on `stderr`) can be requested by setting `PYTHONCOERCECLOCALE=warn`. This setting will also cause the Python runtime to emit a warning if the legacy C locale remains active when the core interpreter is initialized.

While **PEP 538**'s locale coercion has the benefit of also affecting extension modules (such as GNU `readline`), as well as child processes (including those running non-Python applications and older versions of Python), it has the downside of requiring that a suitable target locale be present on the running system. To better handle the case where no suitable target locale is available (as occurs on RHEL/CentOS 7, for example), Python 3.7 also implements *PEP 540: Forced UTF-8 Runtime Mode*.

See also:

PEP 538 – Coercing the legacy C locale to a UTF-8 based locale PEP written and implemented by Nick Coghlan.

2.3 PEP 540: Forced UTF-8 Runtime Mode

The new `-X utf8` command line option and `PYTHONUTF8` environment variable can be used to enable the CPython *UTF-8 mode*.

When in UTF-8 mode, CPython ignores the locale settings, and uses the UTF-8 encoding by default. The error handlers for `sys.stdin` and `sys.stdout` streams are set to `surrogateescape`.

The forced UTF-8 mode can be used to change the text handling behavior in an embedded Python interpreter without changing the locale settings of an embedding application.

While **PEP 540**'s UTF-8 mode has the benefit of working regardless of which locales are available on the running system, it has the downside of having no effect on extension modules (such as GNU `readline`), child processes running non-Python applications, and child processes running older versions of Python. To reduce the risk of corrupting text data when communicating with such components, Python 3.7 also implements *PEP 540: Forced UTF-8 Runtime Mode*).

The UTF-8 mode is enabled by default when the locale is `C` or `POSIX`, and the **PEP 538** locale coercion feature fails to change it to a UTF-8 based alternative (whether that failure is due to `PYTHONCOERCECLOCALE=0` being set, `LC_ALL` being set, or the lack of a suitable target locale).

See also:

PEP 540 – Add a new UTF-8 mode PEP written and implemented by Victor Stinner

2.4 PEP 553: Built-in `breakpoint()`

Python 3.7 includes the new built-in `breakpoint()` function as an easy and consistent way to enter the Python debugger.

Built-in `breakpoint()` calls `sys.breakpointhook()`. By default, the latter imports `pdb` and then calls `pdb.set_trace()`, but by binding `sys.breakpointhook()` to the function of your choosing, `breakpoint()` can enter any debugger. Additionally, the environment variable `PYTHONBREAKPOINT` can be set to the callable of your debugger of choice. Set `PYTHONBREAKPOINT=0` to completely disable built-in `breakpoint()`.

See also:

PEP 553 – Built-in breakpoint() PEP written and implemented by Barry Warsaw

2.5 PEP 539: New C API for Thread-Local Storage

While Python provides a C API for thread-local storage support; the existing Thread Local Storage (TLS) API has used `int` to represent TLS keys across all platforms. This has not generally been a problem for officially-support platforms, but that is neither POSIX-compliant, nor portable in any practical sense.

PEP 539 changes this by providing a new Thread Specific Storage (TSS) API to CPython which supersedes use of the existing TLS API within the CPython interpreter, while deprecating the existing API. The TSS API uses a new type `Py_tss_t` instead of `int` to represent TSS keys–an opaque type the definition of which may depend on the underlying TLS implementation. Therefore, this will allow to build CPython on platforms where the native TLS key is defined in a way that cannot be safely cast to `int`.

Note that on platforms where the native TLS key is defined in a way that cannot be safely cast to `int`, all functions of the existing TLS API will be no-op and immediately return failure. This indicates clearly that the old API is not supported on platforms where it cannot be used reliably, and that no effort will be made to add such support.

See also:

PEP 539 – A New C-API for Thread-Local Storage in CPython PEP written by Erik M. Bray; implementation by Masayuki Yamamoto.

2.6 PEP 562: Customization of Access to Module Attributes

Python 3.7 allows defining `__getattr__()` on modules and will call it whenever a module attribute is otherwise not found. Defining `__dir__()` on modules is now also allowed.

A typical example of where this may be useful is module attribute deprecation and lazy loading.

See also:

PEP 562 – Module `__getattr__` and `__dir__` PEP written and implemented by Ivan Levkivskyi

2.7 PEP 564: New Time Functions With Nanosecond Resolution

The resolution of clocks in modern systems can exceed the limited precision of a floating point number returned by the `time.time()` function and its variants. To avoid loss of precision, **PEP 564** adds six new "nanosecond" variants of the existing timer functions to the `time` module:

- `time.clock_gettime_ns()`
- `time.clock_settime_ns()`
- `time.monotonic_ns()`

- `time.perf_counter_ns()`
- `time.process_time_ns()`
- `time.time_ns()`

The new functions return the number of nanoseconds as an integer value.

Measurements show that on Linux and Windows the resolution of `time.time_ns()` is approximately 3 times better than that of `time.time()`.

See also:

PEP 564 – Add new time functions with nanosecond resolution PEP written and implemented by Victor Stinner

2.8 PEP 565: Show DeprecationWarning in `__main__`

The default handling of `DeprecationWarning` has been changed such that these warnings are once more shown by default, but only when the code triggering them is running directly in the `__main__` module. As a result, developers of single file scripts and those using Python interactively should once again start seeing deprecation warnings for the APIs they use, but deprecation warnings triggered by imported application, library and framework modules will continue to be hidden by default.

As a result of this change, the standard library now allows developers to choose between three different deprecation warning behaviours:

- `FutureWarning`: always displayed by default, recommended for warnings intended to be seen by application end users (e.g. for deprecated application configuration settings).
- `DeprecationWarning`: displayed by default only in `__main__` and when running tests, recommended for warnings intended to be seen by other Python developers where a version upgrade may result in changed behaviour or an error.
- `PendingDeprecationWarning`: displayed by default only when running tests, intended for cases where a future version upgrade will change the warning category to `DeprecationWarning` or `FutureWarning`.

Previously both `DeprecationWarning` and `PendingDeprecationWarning` were only visible when running tests, which meant that developers primarily writing single file scripts or using Python interactively could be surprised by breaking changes in the APIs they used.

See also:

PEP 565 – Show DeprecationWarning in `__main__` PEP written and implemented by Nick Coghlan

2.9 PEP 560: Core Support for `typing` module and Generic Types

Initially **PEP 484** was designed in such way that it would not introduce *any* changes to the core CPython interpreter. Now type hints and the `typing` module are extensively used by the community, so this restriction is removed. The PEP introduces two special methods `__class_getitem__()` and `__mro_entries__`, these methods are now used by most classes and special constructs in `typing`. As a result, the speed of various operations with types increased up to 7 times, the generic types can be used without metaclass conflicts, and several long standing bugs in `typing` module are fixed.

See also:

PEP 560 – Core support for typing module and generic types PEP written and implemented by Ivan Levkivskyi

2.10 PEP 552: Hash-based .pyc Files

Python has traditionally checked the up-to-dateness of bytecode cache files (i.e., `.pyc` files) by comparing the source metadata (last-modified timestamp and size) with source metadata saved in the cache file header when it was generated. While effective, this invalidation method has its drawbacks. When filesystem timestamps are too coarse, Python can miss source updates, leading to user confusion. Additionally, having a timestamp in the cache file is problematic for build reproducibility and content-based build systems.

PEP 552 extends the pyc format to allow the hash of the source file to be used for invalidation instead of the source timestamp. Such `.pyc` files are called "hash-based". By default, Python still uses timestamp-based invalidation and does not generate hash-based `.pyc` files at runtime. Hash-based `.pyc` files may be generated with `py_compile` or `compileall`.

Hash-based `.pyc` files come in two variants: checked and unchecked. Python validates checked hash-based `.pyc` files against the corresponding source files at runtime but doesn't do so for unchecked hash-based pycs. Unchecked hash-based `.pyc` files are a useful performance optimization for environments where a system external to Python (e.g., the build system) is responsible for keeping `.pyc` files up-to-date.

See pyc-invalidation for more information.

See also:

PEP 552 – Deterministic pycs PEP written and implemented by Benjamin Peterson

2.11 PEP 545: Python Documentation Translations

PEP 545 describes the process of creating and maintaining Python documentation translations.

Three new translations have been added:

- Japanese: https://docs.python.org/ja/
- French: https://docs.python.org/fr/
- Korean: https://docs.python.org/ko/

See also:

PEP 545 – Python Documentation Translations PEP written and implemented by Julien Palard, Inada Naoki, and Victor Stinner.

2.12 Development Runtime Mode: -X dev

The new `-X dev` command line option or the new `PYTHONDEVMODE` environment variable can be used to enable CPython's *development mode*. When in development mode, CPython performs additional runtime checks that are too expensive to be enabled by default. See `-X dev` documentation for the full description of the effects of this mode.

3 Other Language Changes

- An `await` expression and comprehensions containing an `async for` clause were illegal in the expressions in formatted string literals due to a problem with the implementation. In Python 3.7 this restriction was lifted.
- More than 255 arguments can now be passed to a function, and a function can now have more than 255 parameters. (Contributed by Serhiy Storchaka in bpo-12844 and bpo-18896.)
- `bytes.fromhex()` and `bytearray.fromhex()` now ignore all ASCII whitespace, not only spaces. (Contributed by Robert Xiao in bpo-28927.)
- `str`, `bytes`, and `bytearray` gained support for the new `isascii()` method, which can be used to test if a string or bytes contain only the ASCII characters. (Contributed by INADA Naoki in bpo-32677.)
- `ImportError` now displays module name and module `__file__` path when `from ... import ...` fails. (Contributed by Matthias Bussonnier in bpo-29546.)
- Circular imports involving absolute imports with binding a submodule to a name are now supported. (Contributed by Serhiy Storchaka in bpo-30024.)
- `object.__format__(x, '')` is now equivalent to `str(x)` rather than `format(str(self), '')`. (Contributed by Serhiy Storchaka in bpo-28974.)
- In order to better support dynamic creation of stack traces, `types.TracebackType` can now be instantiated from Python code, and the `tb_next` attribute on tracebacks is now writable. (Contributed by Nathaniel J. Smith in bpo-30579.)
- When using the `-m` switch, `sys.path[0]` is now eagerly expanded to the full starting directory path, rather than being left as the empty directory (which allows imports from the *current* working directory at the time when an import occurs) (Contributed by Nick Coghlan in bpo-33053.)
- The new `-X importtime` option or the `PYTHONPROFILEIMPORTTIME` environment variable can be used to show the timing of each module import. (Contributed by Victor Stinner in bpo-31415.)

4 New Modules

4.1 contextvars

The new `contextvars` module and a set of new C APIs introduce support for *context variables*. Context variables are conceptually similar to thread-local variables. Unlike TLS, context variables support asynchronous code correctly.

The `asyncio` and `decimal` modules have been updated to use and support context variables out of the box. Particularly the active decimal context is now stored in a context variable, which allows decimal operations to work with the correct context in asynchronous code.

See also:

PEP 567 – Context Variables PEP written and implemented by Yury Selivanov

4.2 dataclasses

The new `dataclass()` decorator provides a way to declare *data classes*. A data class describes its attributes using class variable annotations. Its constructor and other magic methods, such as `__repr__()`, `__eq__()`, and `__hash__()` are generated automatically.

Example:

```
@dataclass
class Point:
    x: float
    y: float
    z: float = 0.0

p = Point(1.5, 2.5)
print(p)    # produces "Point(x=1.5, y=2.5, z=0.0)"
```

See also:

PEP 557 – Data Classes PEP written and implemented by Eric V. Smith

4.3 importlib.resources

The new `importlib.resources` module provides several new APIs and one new ABC for access to, opening, and reading *resources* inside packages. Resources are roughly similar to files inside packages, but they needn't be actual files on the physical file system. Module loaders can provide a `get_resource_reader()` function which returns a `importlib.abc.ResourceReader` instance to support this new API. Built-in file path loaders and zip file loaders both support this.

Contributed by Barry Warsaw and Brett Cannon in bpo-32248.

See also:

importlib_resources – a PyPI backport for earlier Python versions.

5 Improved Modules

5.1 argparse

The new `ArgumentParser.parse_intermixed_args()` method allows intermixing options and positional arguments. (Contributed by paul.j3 in bpo-14191.)

5.2 asyncio

The `asyncio` module has received many new features, usability and *performance improvements*. Notable changes include:

- The new provisional `asyncio.run()` function can be used to run a coroutine from synchronous code by automatically creating and destroying the event loop. (Contributed by Yury Selivanov in bpo-32314.)
- asyncio gained support for `contextvars`. `loop.call_soon()`, `loop.call_soon_threadsafe()`, `loop.call_later()`, `loop.call_at()`, and `Future.add_done_callback()` have a new optional keyword-only *context* parameter. `Tasks` now track their context automatically. See **PEP 567** for more details. (Contributed by Yury Selivanov in bpo-32436.)

- The new `asyncio.create_task()` function has been added as a shortcut to `asyncio.get_event_loop().create_task()`. (Contributed by Andrew Svetlov in bpo-32311.)
- The new `loop.start_tls()` method can be used to upgrade an existing connection to TLS. (Contributed by Yury Selivanov in bpo-23749.)
- The new `loop.sock_recv_into()` method allows reading data from a socket directly into a provided buffer making it possible to reduce data copies. (Contributed by Antoine Pitrou in bpo-31819.)
- The new `asyncio.current_task()` function returns the currently running `Task` instance, and the new `asyncio.all_tasks()` function returns a set of all existing `Task` instances in a given loop. The `Task.current_task()` and `Task.all_tasks()` methods have been deprecated. (Contributed by Andrew Svetlov in bpo-32250.)
- The new *provisional* `BufferedProtocol` class allows implementing streaming protocols with manual control over the receive buffer. (Contributed by Yury Selivanov in bpo-32251.)
- The new `asyncio.get_running_loop()` function returns the currently running loop, and raises a `RuntimeError` if no loop is running. This is in contrast with `asyncio.get_event_loop()`, which will *create* a new event loop if none is running. (Contributed by Yury Selivanov in bpo-32269.)
- The new `StreamWriter.wait_closed()` coroutine method allows waiting until the stream writer is closed. The new `StreamWriter.is_closing()` method can be used to determine if the writer is closing. (Contributed by Andrew Svetlov in bpo-32391.)
- The new `loop.sock_sendfile()` coroutine method allows sending files using `os.sendfile` when possible. (Contributed by Andrew Svetlov in bpo-32410.)
- The new `Future.get_loop()` and `Task.get_loop()` methods return the instance of the loop on which a task or a future were created. `Server.get_loop()` allows doing the same for `asyncio.Server` objects. (Contributed by Yury Selivanov in bpo-32415 and Srinivas Reddy Thatiparthy in bpo-32418.)
- It is now possible to control how instances of `asyncio.Server` begin serving. Previously, the server would start serving immediately when created. The new *start_serving* keyword argument to `loop.create_server()` and `loop.create_unix_server()`, as well as `Server.start_serving()`, and `Server.serve_forever()` can be used to decouple server instantiation and serving. The new `Server.is_serving()` method returns `True` if the server is serving. `Server` objects are now asynchronous context managers:

```
srv = await loop.create_server(...)

async with srv:
    # some code

# At this point, srv is closed and no longer accepts new connections.
```

  (Contributed by Yury Selivanov in bpo-32662.)
- Callback objects returned by `loop.call_later()` gained the new `when()` method which returns an absolute scheduled callback timestamp. (Contributed by Andrew Svetlov in bpo-32741.)
- The `loop.create_datagram_endpoint()` method gained support for Unix sockets. (Contributed by Quentin Dawans in bpo-31245.)
- The `asyncio.open_connection()`, `asyncio.start_server()` functions, `loop.create_connection()`, `loop.create_server()`, `loop.create_accepted_socket()` methods and their corresponding UNIX socket variants now accept the *ssl_handshake_timeout* keyword argument. (Contributed by Neil Aspinall in bpo-29970.)
- The new `Handle.cancelled()` method returns `True` if the callback was cancelled. (Contributed by Marat Sharafutdinov in bpo-31943.)

- The asyncio source has been converted to use the `async`/`await` syntax. (Contributed by Andrew Svetlov in bpo-32193.)
- The new `ReadTransport.is_reading()` method can be used to determine the reading state of the transport. Additionally, calls to `ReadTransport.resume_reading()` and `ReadTransport.pause_reading()` are now idempotent. (Contributed by Yury Selivanov in bpo-32356.)
- Loop methods which accept socket paths now support passing path-like objects. (Contributed by Yury Selivanov in bpo-32066.)
- In `asyncio` TCP sockets on Linux are now created with `TCP_NODELAY` flag set by default. (Contributed by Yury Selivanov and Victor Stinner in bpo-27456.)
- Exceptions occurring in cancelled tasks are no longer logged. (Contributed by Yury Selivanov in bpo-30508.)
- New `WindowsSelectorEventLoopPolicy` and `WindowsProactorEventLoopPolicy` classes. (Contributed by Yury Selivanov in bpo-33792.)

Several `asyncio` APIs have been *deprecated*.

5.3 binascii

The `b2a_uu()` function now accepts an optional *backtick* keyword argument. When it's true, zeros are represented by `'`'` instead of spaces. (Contributed by Xiang Zhang in bpo-30103.)

5.4 calendar

The `HTMLCalendar` class has new class attributes which ease the customization of CSS classes in the produced HTML calendar. (Contributed by Oz Tiram in bpo-30095.)

5.5 collections

`collections.namedtuple()` now supports default values. (Contributed by Raymond Hettinger in bpo-32320.)

5.6 compileall

`compileall.compile_dir()` learned the new *invalidation_mode* parameter, which can be used to enable *hash-based .pyc invalidation*. The invalidation mode can also be specified on the command line using the new `--invalidation-mode` argument. (Contributed by Benjamin Peterson in bpo-31650.)

5.7 concurrent.futures

`ProcessPoolExecutor` and `ThreadPoolExecutor` now support the new *initializer* and *initargs* constructor arguments. (Contributed by Antoine Pitrou in bpo-21423.)

The `ProcessPoolExecutor` can now take the multiprocessing context via the new *mp_context* argument. (Contributed by Thomas Moreau in bpo-31540.)

5.8 contextlib

The new `nullcontext()` is a simpler and faster no-op context manager than `ExitStack`. (Contributed by Jesse-Bakker in bpo-10049.)

The new `asynccontextmanager()`, `AbstractAsyncContextManager`, and `AsyncExitStack` have been added to complement their synchronous counterparts. (Contributed by Jelle Zijlstra in bpo-29679 and bpo-30241, and by Alexander Mohr and Ilya Kulakov in bpo-29302.)

5.9 cProfile

The `cProfile` command line now accepts `-m module_name` as an alternative to script path. (Contributed by Sanyam Khurana in bpo-21862.)

5.10 crypt

The `crypt` module now supports the Blowfish hashing method. (Contributed by Serhiy Storchaka in bpo-31664.)

The `mksalt()` function now allows specifying the number of rounds for hashing. (Contributed by Serhiy Storchaka in bpo-31702.)

5.11 datetime

The new `datetime.fromisoformat()` method constructs a `datetime` object from a string in one of the formats output by `datetime.isoformat()`. (Contributed by Paul Ganssle in bpo-15873.)

The `tzinfo` class now supports sub-minute offsets. (Contributed by Alexander Belopolsky in bpo-5288.)

5.12 dbm

`dbm.dumb` now supports reading read-only files and no longer writes the index file when it is not changed.

5.13 decimal

The `decimal` module now uses *context variables* to store the decimal context. (Contributed by Yury Selivanov in bpo-32630.)

5.14 dis

The `dis()` function is now able to disassemble nested code objects (the code of comprehensions, generator expressions and nested functions, and the code used for building nested classes). The maximum depth of disassembly recursion is controlled by the new *depth* parameter. (Contributed by Serhiy Storchaka in bpo-11822.)

5.15 distutils

`README.rst` is now included in the list of distutils standard READMEs and therefore included in source distributions. (Contributed by Ryan Gonzalez in bpo-11913.)

5.16 enum

The `Enum` learned the new `_ignore_` class property, which allows listing the names of properties which should not become enum members. (Contributed by Ethan Furman in bpo-31801.)

In Python 3.8, attempting to check for non-Enum objects in `Enum` classes will raise a `TypeError` (e.g. `1 in Color`); similarly, attempting to check for non-Flag objects in a `Flag` member will raise `TypeError` (e.g. `1 in Perm.RW`); currently, both operations return `False` instead and are deprecated. (Contributed by Ethan Furman in bpo-33217.)

5.17 functools

`functools.singledispatch()` now supports registering implementations using type annotations. (Contributed by Łukasz Langa in bpo-32227.)

5.18 gc

The new `gc.freeze()` function allows freezing all objects tracked by the garbage collector and excluding them from future collections. This can be used before a POSIX `fork()` call to make the GC copy-on-write friendly or to speed up collection. The new `gc.unfreeze()` functions reverses this operation. Additionally, `gc.get_freeze_count()` can be used to obtain the number of frozen objects. (Contributed by Li Zekun in bpo-31558.)

5.19 hmac

The `hmac` module now has an optimized one-shot `digest()` function, which is up to three times faster than `HMAC()`. (Contributed by Christian Heimes in bpo-32433.)

5.20 http.client

`HTTPConnection` and `HTTPSConnection` now support the new *blocksize* argument for improved upload throughput. (Contributed by Nir Soffer in bpo-31945.)

5.21 http.server

`SimpleHTTPRequestHandler` now supports the HTTP `If-Modified-Since` header. The server returns the 304 response status if the target file was not modified after the time specified in the header. (Contributed by Pierre Quentel in bpo-29654.)

`SimpleHTTPRequestHandler` accepts the new *directory* argument, in addition to the new `--directory` command line argument. With this parameter, the server serves the specified directory, by default it uses the current working directory. (Contributed by Stéphane Wirtel and Julien Palard in bpo-28707.)

The new `ThreadingHTTPServer` class uses threads to handle requests using `ThreadingMixin`. It is used when `http.server` is run with `-m`. (Contributed by Julien Palard in bpo-31639.)

5.22 idlelib and IDLE

Multiple fixes for autocompletion. (Contributed by Louie Lu in bpo-15786.)

Module Browser (on the File menu, formerly called Class Browser), now displays nested functions and classes in addition to top-level functions and classes. (Contributed by Guilherme Polo, Cheryl Sabella, and Terry Jan Reedy in bpo-1612262.)

The Settings dialog (Options, Configure IDLE) has been partly rewritten to improve both appearance and function. (Contributed by Cheryl Sabella and Terry Jan Reedy in multiple issues.)

The font sample now includes a selection of non-Latin characters so that users can better see the effect of selecting a particular font. (Contributed by Terry Jan Reedy in bpo-13802.) The sample can be edited to include other characters. (Contributed by Serhiy Storchaka in bpo-31860.)

The IDLE features formerly implemented as extensions have been reimplemented as normal features. Their settings have been moved from the Extensions tab to other dialog tabs. (Contributed by Charles Wohlganger and Terry Jan Reedy in bpo-27099.)

Editor code context option revised. Box displays all context lines up to maxlines. Clicking on a context line jumps the editor to that line. Context colors for custom themes is added to Highlights tab of Settings dialog. (Contributed by Cheryl Sabella and Terry Jan Reedy in bpo-33642, bpo-33768, and bpo-33679.)

On Windows, a new API call tells Windows that tk scales for DPI. On Windows 8.1+ or 10, with DPI compatibility properties of the Python binary unchanged, and a monitor resolution greater than 96 DPI, this should make text and lines sharper. It should otherwise have no effect. (Contributed by Terry Jan Reedy in bpo-33656.)

New in 3.7.1:

Output over N lines (50 by default) is squeezed down to a button. N can be changed in the PyShell section of the General page of the Settings dialog. Fewer, but possibly extra long, lines can be squeezed by right clicking on the output. Squeezed output can be expanded in place by double-clicking the button or into the clipboard or a separate window by right-clicking the button. (Contributed by Tal Einat in bpo-1529353.)

The changes above have been backported to 3.6 maintenance releases.

NEW in 3.7.4:

Add "Run Customized" to the Run menu to run a module with customized settings. Any command line arguments entered are added to sys.argv. They re-appear in the box for the next customized run. One can also suppress the normal Shell main module restart. (Contributed by Cheryl Sabella, Terry Jan Reedy, and others in bpo-5680 and bpo-37627.)

New in 3.7.5:

Add optional line numbers for IDLE editor windows. Windows open without line numbers unless set otherwise in the General tab of the configuration dialog. Line numbers for an existing window are shown and hidden in the Options menu. (Contributed by Tal Einat and Saimadhav Heblikar in bpo-17535.)

5.23 importlib

The `importlib.abc.ResourceReader` ABC was introduced to support the loading of resources from packages. See also *importlib.resources*. (Contributed by Barry Warsaw, Brett Cannon in bpo-32248.)

`importlib.reload()` now raises `ModuleNotFoundError` if the module lacks a spec. (Contributed by Garvit Khatri in bpo-29851.)

`importlib.find_spec()` now raises `ModuleNotFoundError` instead of `AttributeError` if the specified parent module is not a package (i.e. lacks a `__path__` attribute). (Contributed by Milan Oberkirch in bpo-30436.)

The new `importlib.source_hash()` can be used to compute the hash of the passed source. A *hash-based .pyc file* embeds the value returned by this function.

5.24 io

The new `TextIOWrapper.reconfigure()` method can be used to reconfigure the text stream with the new settings. (Contributed by Antoine Pitrou in bpo-30526 and INADA Naoki in bpo-15216.)

5.25 ipaddress

The new `subnet_of()` and `supernet_of()` methods of `ipaddress.IPv6Network` and `ipaddress.IPv4Network` can be used for network containment tests. (Contributed by Michel Albert and Cheryl Sabella in bpo-20825.)

5.26 itertools

`itertools.islice()` now accepts `integer-like objects` as start, stop, and slice arguments. (Contributed by Will Roberts in bpo-30537.)

5.27 locale

The new *monetary* argument to `locale.format_string()` can be used to make the conversion use monetary thousands separators and grouping strings. (Contributed by Garvit in bpo-10379.)

The `locale.getpreferredencoding()` function now always returns `'UTF-8'` on Android or when in the *forced UTF-8 mode*.

5.28 logging

`Logger` instances can now be pickled. (Contributed by Vinay Sajip in bpo-30520.)

The new `StreamHandler.setStream()` method can be used to replace the logger stream after handler creation. (Contributed by Vinay Sajip in bpo-30522.)

It is now possible to specify keyword arguments to handler constructors in configuration passed to `logging.config.fileConfig()`. (Contributed by Preston Landers in bpo-31080.)

5.29 math

The new `math.remainder()` function implements the IEEE 754-style remainder operation. (Contributed by Mark Dickinson in bpo-29962.)

5.30 mimetypes

The MIME type of .bmp has been changed from `'image/x-ms-bmp'` to `'image/bmp'`. (Contributed by Nitish Chandra in bpo-22589.)

5.31 msilib

The new `Database.Close()` method can be used to close the MSI database. (Contributed by Berker Peksag in bpo-20486.)

5.32 multiprocessing

The new `Process.close()` method explicitly closes the process object and releases all resources associated with it. `ValueError` is raised if the underlying process is still running. (Contributed by Antoine Pitrou in bpo-30596.)

The new `Process.kill()` method can be used to terminate the process using the `SIGKILL` signal on Unix. (Contributed by Vitor Pereira in bpo-30794.)

Non-daemonic threads created by `Process` are now joined on process exit. (Contributed by Antoine Pitrou in bpo-18966.)

5.33 os

`os.fwalk()` now accepts the *path* argument as `bytes`. (Contributed by Serhiy Storchaka in bpo-28682.)

`os.scandir()` gained support for file descriptors. (Contributed by Serhiy Storchaka in bpo-25996.)

The new `register_at_fork()` function allows registering Python callbacks to be executed at process fork. (Contributed by Antoine Pitrou in bpo-16500.)

Added `os.preadv()` (combine the functionality of `os.readv()` and `os.pread()`) and `os.pwritev()` functions (combine the functionality of `os.writev()` and `os.pwrite()`). (Contributed by Pablo Galindo in bpo-31368.)

The mode argument of `os.makedirs()` no longer affects the file permission bits of newly-created intermediate-level directories. (Contributed by Serhiy Storchaka in bpo-19930.)

`os.dup2()` now returns the new file descriptor. Previously, `None` was always returned. (Contributed by Benjamin Peterson in bpo-32441.)

The structure returned by `os.stat()` now contains the `st_fstype` attribute on Solaris and its derivatives. (Contributed by Jesús Cea Avión in bpo-32659.)

5.34 pathlib

The new `Path.is_mount()` method is now available on POSIX systems and can be used to determine whether a path is a mount point. (Contributed by Cooper Ry Lees in bpo-30897.)

5.35 pdb

`pdb.set_trace()` now takes an optional *header* keyword-only argument. If given, it is printed to the console just before debugging begins. (Contributed by Barry Warsaw in bpo-31389.)

`pdb` command line now accepts `-m module_name` as an alternative to script file. (Contributed by Mario Corchero in bpo-32206.)

5.36 py_compile

`py_compile.compile()` – and by extension, `compileall` – now respects the `SOURCE_DATE_EPOCH` environment variable by unconditionally creating `.pyc` files for hash-based validation. This allows for guaranteeing reproducible builds of `.pyc` files when they are created eagerly. (Contributed by Bernhard M. Wiedemann in bpo-29708.)

5.37 pydoc

The pydoc server can now bind to an arbitrary hostname specified by the new `-n` command-line argument. (Contributed by Feanil Patel in bpo-31128.)

5.38 queue

The new `SimpleQueue` class is an unbounded FIFO queue. (Contributed by Antoine Pitrou in bpo-14976.)

5.39 re

The flags `re.ASCII`, `re.LOCALE` and `re.UNICODE` can be set within the scope of a group. (Contributed by Serhiy Storchaka in bpo-31690.)

`re.split()` now supports splitting on a pattern like `r'\b'`, `'^$'` or `(?=-)` that matches an empty string. (Contributed by Serhiy Storchaka in bpo-25054.)

Regular expressions compiled with the `re.LOCALE` flag no longer depend on the locale at compile time. Locale settings are applied only when the compiled regular expression is used. (Contributed by Serhiy Storchaka in bpo-30215.)

`FutureWarning` is now emitted if a regular expression contains character set constructs that will change semantically in the future, such as nested sets and set operations. (Contributed by Serhiy Storchaka in bpo-30349.)

Compiled regular expression and match objects can now be copied using `copy.copy()` and `copy.deepcopy()`. (Contributed by Serhiy Storchaka in bpo-10076.)

5.40 signal

The new *warn_on_full_buffer* argument to the `signal.set_wakeup_fd()` function makes it possible to specify whether Python prints a warning on stderr when the wakeup buffer overflows. (Contributed by Nathaniel J. Smith in bpo-30050.)

5.41 socket

The new `socket.getblocking()` method returns `True` if the socket is in blocking mode and `False` otherwise. (Contributed by Yury Selivanov in bpo-32373.)

The new `socket.close()` function closes the passed socket file descriptor. This function should be used instead of `os.close()` for better compatibility across platforms. (Contributed by Christian Heimes in bpo-32454.)

The `socket` module now exposes the `socket.TCP_CONGESTION` (Linux 2.6.13), `socket.TCP_USER_TIMEOUT` (Linux 2.6.37), and `socket.TCP_NOTSENT_LOWAT` (Linux 3.12) constants. (Contributed by Omar Sandoval in bpo-26273 and Nathaniel J. Smith in bpo-29728.)

Support for `socket.AF_VSOCK` sockets has been added to allow communication between virtual machines and their hosts. (Contributed by Cathy Avery in bpo-27584.)

Sockets now auto-detect family, type and protocol from file descriptor by default. (Contributed by Christian Heimes in bpo-28134.)

5.42 socketserver

`socketserver.ThreadingMixIn.server_close()` now waits until all non-daemon threads complete. `socketserver.ForkingMixIn.server_close()` now waits until all child processes complete.

Add a new `socketserver.ForkingMixIn.block_on_close` class attribute to `socketserver.ForkingMixIn` and `socketserver.ThreadingMixIn` classes. Set the class attribute to `False` to get the pre-3.7 behaviour.

5.43 sqlite3

`sqlite3.Connection` now exposes the `backup()` method when the underlying SQLite library is at version 3.6.11 or higher. (Contributed by Lele Gaifax in bpo-27645.)

The *database* argument of `sqlite3.connect()` now accepts any path-like object, instead of just a string. (Contributed by Anders Lorentsen in bpo-31843.)

5.44 ssl

The `ssl` module now uses OpenSSL's builtin API instead of `match_hostname()` to check a host name or an IP address. Values are validated during TLS handshake. Any certificate validation error including failing the host name check now raises `SSLCertVerificationError` and aborts the handshake with a proper TLS Alert message. The new exception contains additional information. Host name validation can be customized with `SSLContext.hostname_checks_common_name`. (Contributed by Christian Heimes in bpo-31399.)

Note: The improved host name check requires a *libssl* implementation compatible with OpenSSL 1.0.2 or 1.1. Consequently, OpenSSL 0.9.8 and 1.0.1 are no longer supported (see *Platform Support Removals* for more details). The ssl module is mostly compatible with LibreSSL 2.7.2 and newer.

The `ssl` module no longer sends IP addresses in SNI TLS extension. (Contributed by Christian Heimes in bpo-32185.)

`match_hostname()` no longer supports partial wildcards like `www*.example.org`. (Contributed by Mandeep Singh in bpo-23033 and Christian Heimes in bpo-31399.)

The default cipher suite selection of the `ssl` module now uses a blacklist approach rather than a hard-coded whitelist. Python no longer re-enables ciphers that have been blocked by OpenSSL security updates. Default cipher suite selection can be configured at compile time. (Contributed by Christian Heimes in bpo-31429.)

Validation of server certificates containing internationalized domain names (IDNs) is now supported. As part of this change, the `SSLSocket.server_hostname` attribute now stores the expected hostname in A-label form (`"xn--pythn-mua.org"`), rather than the U-label form (`"pythön.org"`). (Contributed by Nathaniel J. Smith and Christian Heimes in bpo-28414.)

The `ssl` module has preliminary and experimental support for TLS 1.3 and OpenSSL 1.1.1. At the time of Python 3.7.0 release, OpenSSL 1.1.1 is still under development and TLS 1.3 hasn't been finalized yet. The TLS 1.3 handshake and protocol behaves slightly differently than TLS 1.2 and earlier, see ssl-tlsv1_3. (Contributed by Christian Heimes in bpo-32947, bpo-20995, bpo-29136, bpo-30622 and bpo-33618)

`SSLSocket` and `SSLObject` no longer have a public constructor. Direct instantiation was never a documented and supported feature. Instances must be created with `SSLContext` methods `wrap_socket()` and `wrap_bio()`. (Contributed by Christian Heimes in bpo-32951)

OpenSSL 1.1 APIs for setting the minimum and maximum TLS protocol version are available as `SSLContext.minimum_version` and `SSLContext.maximum_version`. Supported protocols are indicated by several new flags, such as `HAS_TLSv1_1`. (Contributed by Christian Heimes in bpo-32609.)

Added `SSLContext.post_handshake_auth` to enable and `ssl.SSLSocket.verify_client_post_handshake()` to initiate TLS 1.3 post-handshake authentication. (Contributed by Christian Heimes in bpo-34670.)

5.45 string

`string.Template` now lets you to optionally modify the regular expression pattern for braced placeholders and non-braced placeholders separately. (Contributed by Barry Warsaw in bpo-1198569.)

5.46 subprocess

The `subprocess.run()` function accepts the new *capture_output* keyword argument. When true, stdout and stderr will be captured. This is equivalent to passing `subprocess.PIPE` as *stdout* and *stderr* arguments. (Contributed by Bo Bayles in bpo-32102.)

The `subprocess.run` function and the `subprocess.Popen` constructor now accept the *text* keyword argument as an alias to *universal_newlines*. (Contributed by Andrew Clegg in bpo-31756.)

On Windows the default for *close_fds* was changed from `False` to `True` when redirecting the standard handles. It's now possible to set *close_fds* to true when redirecting the standard handles. See `subprocess.Popen`. This means that *close_fds* now defaults to `True` on all supported platforms. (Contributed by Segev Finer in bpo-19764.)

The subprocess module is now more graceful when handling `KeyboardInterrupt` during `subprocess.call()`, `subprocess.run()`, or in a `Popen` context manager. It now waits a short amount of time for the child to exit, before continuing the handling of the `KeyboardInterrupt` exception. (Contributed by Gregory P. Smith in bpo-25942.)

5.47 sys

The new `sys.breakpointhook()` hook function is called by the built-in `breakpoint()`. (Contributed by Barry Warsaw in bpo-31353.)

On Android, the new `sys.getandroidapilevel()` returns the build-time Android API version. (Contributed by Victor Stinner in bpo-28740.)

The new `sys.get_coroutine_origin_tracking_depth()` function returns the current coroutine origin tracking depth, as set by the new `sys.set_coroutine_origin_tracking_depth()`. `asyncio` has been converted to use this new API instead of the deprecated `sys.set_coroutine_wrapper()`. (Contributed by Nathaniel J. Smith in bpo-32591.)

5.48 time

PEP 564 adds six new functions with nanosecond resolution to the `time` module:

- `time.clock_gettime_ns()`
- `time.clock_settime_ns()`
- `time.monotonic_ns()`
- `time.perf_counter_ns()`
- `time.process_time_ns()`

- `time.time_ns()`

New clock identifiers have been added:

- `time.CLOCK_BOOTTIME` (Linux): Identical to `time.CLOCK_MONOTONIC`, except it also includes any time that the system is suspended.
- `time.CLOCK_PROF` (FreeBSD, NetBSD and OpenBSD): High-resolution per-process CPU timer.
- `time.CLOCK_UPTIME` (FreeBSD, OpenBSD): Time whose absolute value is the time the system has been running and not suspended, providing accurate uptime measurement.

The new `time.thread_time()` and `time.thread_time_ns()` functions can be used to get per-thread CPU time measurements. (Contributed by Antoine Pitrou in bpo-32025.)

The new `time.pthread_getcpuclockid()` function returns the clock ID of the thread-specific CPU-time clock.

5.49 tkinter

The new `tkinter.ttk.Spinbox` class is now available. (Contributed by Alan Moore in bpo-32585.)

5.50 tracemalloc

`tracemalloc.Traceback` behaves more like regular tracebacks, sorting the frames from oldest to most recent. `Traceback.format()` now accepts negative *limit*, truncating the result to the `abs(limit)` oldest frames. To get the old behaviour, use the new *most_recent_first* argument to `Traceback.format()`. (Contributed by Jesse Bakker in bpo-32121.)

5.51 types

The new `WrapperDescriptorType`, `MethodWrapperType`, `MethodDescriptorType`, and `ClassMethodDescriptorType` classes are now available. (Contributed by Manuel Krebber and Guido van Rossum in bpo-29377, and Serhiy Storchaka in bpo-32265.)

The new `types.resolve_bases()` function resolves MRO entries dynamically as specified by **PEP 560**. (Contributed by Ivan Levkivskyi in bpo-32717.)

5.52 unicodedata

The internal `unicodedata` database has been upgraded to use Unicode 11. (Contributed by Benjamin Peterson.)

5.53 unittest

The new `-k` command-line option allows filtering tests by a name substring or a Unix shell-like pattern. For example, `python -m unittest -k foo` runs `foo_tests.SomeTest.test_something`, `bar_tests.SomeTest.test_foo`, but not `bar_tests.FooTest.test_something`. (Contributed by Jonas Haag in bpo-32071.)

5.54 unittest.mock

The `sentinel` attributes now preserve their identity when they are `copied` or `pickled`. (Contributed by Serhiy Storchaka in bpo-20804.)

The new `seal()` function allows sealing `Mock` instances, which will disallow further creation of attribute mocks. The seal is applied recursively to all attributes that are themselves mocks. (Contributed by Mario Corchero in bpo-30541.)

5.55 urllib.parse

`urllib.parse.quote()` has been updated from **RFC 2396** to **RFC 3986**, adding ~ to the set of characters that are never quoted by default. (Contributed by Christian Theune and Ratnadeep Debnath in bpo-16285.)

5.56 uu

The `uu.encode()` function now accepts an optional *backtick* keyword argument. When it's true, zeros are represented by `'`` '` instead of spaces. (Contributed by Xiang Zhang in bpo-30103.)

5.57 uuid

The new `UUID.is_safe` attribute relays information from the platform about whether generated UUIDs are generated with a multiprocessing-safe method. (Contributed by Barry Warsaw in bpo-22807.)

`uuid.getnode()` now prefers universally administered MAC addresses over locally administered MAC addresses. This makes a better guarantee for global uniqueness of UUIDs returned from `uuid.uuid1()`. If only locally administered MAC addresses are available, the first such one found is returned. (Contributed by Barry Warsaw in bpo-32107.)

5.58 warnings

The initialization of the default warnings filters has changed as follows:

- warnings enabled via command line options (including those for `-b` and the new CPython-specific `-X dev` option) are always passed to the warnings machinery via the `sys.warnoptions` attribute.
- warnings filters enabled via the command line or the environment now have the following order of precedence:
  - the `BytesWarning` filter for `-b` (or `-bb`)
  - any filters specified with the `-W` option
  - any filters specified with the `PYTHONWARNINGS` environment variable
  - any other CPython specific filters (e.g. the `default` filter added for the new `-X dev` mode)
  - any implicit filters defined directly by the warnings machinery
- in CPython debug builds, all warnings are now displayed by default (the implicit filter list is empty)

(Contributed by Nick Coghlan and Victor Stinner in bpo-20361, bpo-32043, and bpo-32230.)

Deprecation warnings are once again shown by default in single-file scripts and at the interactive prompt. See *PEP 565: Show DeprecationWarning in __main__* for details. (Contributed by Nick Coghlan in bpo-31975.)

5.59 xml

As mitigation against DTD and external entity retrieval, the `xml.dom.minidom` and `xml.sax` modules no longer process external entities by default. (Contributed by Christian Heimes in bpo-17239.)

5.60 xml.etree

ElementPath predicates in the `find()` methods can now compare text of the current node with `[. = "text"]`, not only text in children. Predicates also allow adding spaces for better readability. (Contributed by Stefan Behnel in bpo-31648.)

5.61 xmlrpc.server

`SimpleXMLRPCDispatcher.register_function` can now be used as a decorator. (Contributed by Xiang Zhang in bpo-7769.)

5.62 zipapp

Function `create_archive()` now accepts an optional *filter* argument to allow the user to select which files should be included in the archive. (Contributed by Irmen de Jong in bpo-31072.)

Function `create_archive()` now accepts an optional *compressed* argument to generate a compressed archive. A command line option `--compress` has also been added to support compression. (Contributed by Zhiming Wang in bpo-31638.)

5.63 zipfile

`ZipFile` now accepts the new *compresslevel* parameter to control the compression level. (Contributed by Bo Bayles in bpo-21417.)

Subdirectories in archives created by `ZipFile` are now stored in alphabetical order. (Contributed by Bernhard M. Wiedemann in bpo-30693.)

6 C API Changes

A new API for thread-local storage has been implemented. See *PEP 539: New C API for Thread-Local Storage* for an overview and thread-specific-storage-api for a complete reference. (Contributed by Masayuki Yamamoto in bpo-25658.)

The new *context variables* functionality exposes a number of new C APIs.

The new `PyImport_GetModule()` function returns the previously imported module with the given name. (Contributed by Eric Snow in bpo-28411.)

The new `Py_RETURN_RICHCOMPARE` macro eases writing rich comparison functions. (Contributed by Petr Victorin in bpo-23699.)

The new `Py_UNREACHABLE` macro can be used to mark unreachable code paths. (Contributed by Barry Warsaw in bpo-31338.)

The `tracemalloc` now exposes a C API through the new `PyTraceMalloc_Track()` and `PyTraceMalloc_Untrack()` functions. (Contributed by Victor Stinner in bpo-30054.)

The new `import__find__load__start()` and `import__find__load__done()` static markers can be used to trace module imports. (Contributed by Christian Heimes in bpo-31574.)

The fields `name` and `doc` of structures `PyMemberDef`, `PyGetSetDef`, `PyStructSequence_Field`, `PyStructSequence_Desc`, and `wrapperbase` are now of type `const char *` rather of `char *`. (Contributed by Serhiy Storchaka in bpo-28761.)

The result of `PyUnicode_AsUTF8AndSize()` and `PyUnicode_AsUTF8()` is now of type `const char *` rather of `char *`. (Contributed by Serhiy Storchaka in bpo-28769.)

The result of `PyMapping_Keys()`, `PyMapping_Values()` and `PyMapping_Items()` is now always a list, rather than a list or a tuple. (Contributed by Oren Milman in bpo-28280.)

Added functions `PySlice_Unpack()` and `PySlice_AdjustIndices()`. (Contributed by Serhiy Storchaka in bpo-27867.)

`PyOS_AfterFork()` is deprecated in favour of the new functions `PyOS_BeforeFork()`, `PyOS_AfterFork_Parent()` and `PyOS_AfterFork_Child()`. (Contributed by Antoine Pitrou in bpo-16500.)

The `PyExc_RecursionErrorInst` singleton that was part of the public API has been removed as its members being never cleared may cause a segfault during finalization of the interpreter. Contributed by Xavier de Gaye in bpo-22898 and bpo-30697.

Added C API support for timezones with timezone constructors `PyTimeZone_FromOffset()` and `PyTimeZone_FromOffsetAndName()`, and access to the UTC singleton with `PyDateTime_TimeZone_UTC`. Contributed by Paul Ganssle in bpo-10381.

The type of results of `PyThread_start_new_thread()` and `PyThread_get_thread_ident()`, and the *id* parameter of `PyThreadState_SetAsyncExc()` changed from `long` to `unsigned long`. (Contributed by Serhiy Storchaka in bpo-6532.)

`PyUnicode_AsWideCharString()` now raises a `ValueError` if the second argument is `NULL` and the `wchar_t*` string contains null characters. (Contributed by Serhiy Storchaka in bpo-30708.)

Changes to the startup sequence and the management of dynamic memory allocators mean that the long documented requirement to call `Py_Initialize()` before calling most C API functions is now relied on more heavily, and failing to abide by it may lead to segfaults in embedding applications. See the *Porting to Python 3.7* section in this document and the pre-init-safe section in the C API documentation for more details.

The new `PyInterpreterState_GetID()` returns the unique ID for a given interpreter. (Contributed by Eric Snow in bpo-29102.)

`Py_DecodeLocale()`, `Py_EncodeLocale()` now use the UTF-8 encoding when the *UTF-8 mode* is enabled. (Contributed by Victor Stinner in bpo-29240.)

`PyUnicode_DecodeLocaleAndSize()` and `PyUnicode_EncodeLocale()` now use the current locale encoding for `surrogateescape` error handler. (Contributed by Victor Stinner in bpo-29240.)

The *start* and *end* parameters of `PyUnicode_FindChar()` are now adjusted to behave like string slices. (Contributed by Xiang Zhang in bpo-28822.)

7 Build Changes

Support for building `--without-threads` has been removed. The `threading` module is now always available. (Contributed by Antoine Pitrou in bpo-31370.).

A full copy of libffi is no longer bundled for use when building the `_ctypes` module on non-OSX UNIX platforms. An installed copy of libffi is now required when building `_ctypes` on such platforms. (Contributed by Zachary Ware in bpo-27979.)

The Windows build process no longer depends on Subversion to pull in external sources, a Python script is used to download zipfiles from GitHub instead. If Python 3.6 is not found on the system (via `py -3.6`), NuGet is used to download a copy of 32-bit Python for this purpose. (Contributed by Zachary Ware in bpo-30450.)

The `ssl` module requires OpenSSL 1.0.2 or 1.1 compatible libssl. OpenSSL 1.0.1 has reached end of lifetime on 2016-12-31 and is no longer supported. LibreSSL is temporarily not supported as well. LibreSSL releases up to version 2.6.4 are missing required OpenSSL 1.0.2 APIs.

8 Optimizations

The overhead of calling many methods of various standard library classes implemented in C has been significantly reduced by porting more code to use the `METH_FASTCALL` convention. (Contributed by Victor Stinner in bpo-29300, bpo-29507, bpo-29452, and bpo-29286.)

Various optimizations have reduced Python startup time by 10% on Linux and up to 30% on macOS. (Contributed by Victor Stinner, INADA Naoki in bpo-29585, and Ivan Levkivskyi in bpo-31333.)

Method calls are now up to 20% faster due to the bytecode changes which avoid creating bound method instances. (Contributed by Yury Selivanov and INADA Naoki in bpo-26110.)

The `asyncio` module received a number of notable optimizations for commonly used functions:

- The `asyncio.get_event_loop()` function has been reimplemented in C to make it up to 15 times faster. (Contributed by Yury Selivanov in bpo-32296.)
- `asyncio.Future` callback management has been optimized. (Contributed by Yury Selivanov in bpo-32348.)
- `asyncio.gather()` is now up to 15% faster. (Contributed by Yury Selivanov in bpo-32355.)
- `asyncio.sleep()` is now up to 2 times faster when the *delay* argument is zero or negative. (Contributed by Andrew Svetlov in bpo-32351.)
- The performance overhead of asyncio debug mode has been reduced. (Contributed by Antoine Pitrou in bpo-31970.)

As a result of *PEP 560 work*, the import time of `typing` has been reduced by a factor of 7, and many typing operations are now faster. (Contributed by Ivan Levkivskyi in bpo-32226.)

`sorted()` and `list.sort()` have been optimized for common cases to be up to 40-75% faster. (Contributed by Elliot Gorokhovsky in bpo-28685.)

`dict.copy()` is now up to 5.5 times faster. (Contributed by Yury Selivanov in bpo-31179.)

`hasattr()` and `getattr()` are now about 4 times faster when *name* is not found and *obj* does not override `object.__getattr__()` or `object.__getattribute__()`. (Contributed by INADA Naoki in bpo-32544.)

Searching for certain Unicode characters (like Ukrainian capital "Є") in a string was up to 25 times slower than searching for other characters. It is now only 3 times slower in the worst case. (Contributed by Serhiy Storchaka in bpo-24821.)

The `collections.namedtuple()` factory has been reimplemented to make the creation of named tuples 4 to 6 times faster. (Contributed by Jelle Zijlstra with further improvements by INADA Naoki, Serhiy Storchaka, and Raymond Hettinger in bpo-28638.)

`date.fromordinal()` and `date.fromtimestamp()` are now up to 30% faster in the common case. (Contributed by Paul Ganssle in bpo-32403.)

The `os.fwalk()` function is now up to 2 times faster thanks to the use of `os.scandir()`. (Contributed by Serhiy Storchaka in bpo-25996.)

The speed of the `shutil.rmtree()` function has been improved by 20–40% thanks to the use of the `os.scandir()` function. (Contributed by Serhiy Storchaka in bpo-28564.)

Optimized case-insensitive matching and searching of `regular expressions`. Searching some patterns can now be up to 20 times faster. (Contributed by Serhiy Storchaka in bpo-30285.)

`re.compile()` now converts `flags` parameter to int object if it is `RegexFlag`. It is now as fast as Python 3.5, and faster than Python 3.6 by about 10% depending on the pattern. (Contributed by INADA Naoki in bpo-31671.)

The `modify()` methods of classes `selectors.EpollSelector`, `selectors.PollSelector` and `selectors.DevpollSelector` may be around 10% faster under heavy loads. (Contributed by Giampaolo Rodola' in bpo-30014)

Constant folding has been moved from the peephole optimizer to the new AST optimizer, which is able perform optimizations more consistently. (Contributed by Eugene Toder and INADA Naoki in bpo-29469 and bpo-11549.)

Most functions and methods in `abc` have been rewritten in C. This makes creation of abstract base classes, and calling `isinstance()` and `issubclass()` on them 1.5x faster. This also reduces Python start-up time by up to 10%. (Contributed by Ivan Levkivskyi and INADA Naoki in bpo-31333)

Significant speed improvements to alternate constructors for `datetime.date` and `datetime.datetime` by using fast-path constructors when not constructing subclasses. (Contributed by Paul Ganssle in bpo-32403)

The speed of comparison of `array.array` instances has been improved considerably in certain cases. It is now from 10x to 70x faster when comparing arrays holding values of the same integer type. (Contributed by Adrian Wielgosik in bpo-24700.)

The `math.erf()` and `math.erfc()` functions now use the (faster) C library implementation on most platforms. (Contributed by Serhiy Storchaka in bpo-26121.)

9 Other CPython Implementation Changes

- Trace hooks may now opt out of receiving the `line` and opt into receiving the `opcode` events from the interpreter by setting the corresponding new `f_trace_lines` and `f_trace_opcodes` attributes on the frame being traced. (Contributed by Nick Coghlan in bpo-31344.)
- Fixed some consistency problems with namespace package module attributes. Namespace module objects now have an `__file__` that is set to `None` (previously unset), and their `__spec__.origin` is also set to `None` (previously the string `"namespace"`). See bpo-32305. Also, the namespace module object's `__spec__.loader` is set to the same value as `__loader__` (previously, the former was set to `None`). See bpo-32303.
- The `locals()` dictionary now displays in the lexical order that variables were defined. Previously, the order was undefined. (Contributed by Raymond Hettinger in bpo-32690.)
- The `distutils upload` command no longer tries to change CR end-of-line characters to CRLF. This fixes a corruption issue with sdists that ended with a byte equivalent to CR. (Contributed by Bo Bayles in bpo-32304.)

10 Deprecated Python Behavior

Yield expressions (both `yield` and `yield from` clauses) are now deprecated in comprehensions and generator expressions (aside from the iterable expression in the leftmost `for` clause). This ensures that comprehensions always immediately return a container of the appropriate type (rather than potentially returning a generator iterator object), while generator expressions won't attempt to interleave their implicit output with the output from any explicit yield expressions. In Python 3.7, such expressions emit `DeprecationWarning` when compiled, in Python 3.8 this will be a `SyntaxError`. (Contributed by Serhiy Storchaka in bpo-10544.)

Returning a subclass of `complex` from `object.__complex__()` is deprecated and will be an error in future Python versions. This makes `__complex__()` consistent with `object.__int__()` and `object.__float__()`. (Contributed by Serhiy Storchaka in bpo-28894.)

11 Deprecated Python modules, functions and methods

11.1 aifc

`aifc.openfp()` has been deprecated and will be removed in Python 3.9. Use `aifc.open()` instead. (Contributed by Brian Curtin in bpo-31985.)

11.2 asyncio

Support for directly `await`-ing instances of `asyncio.Lock` and other asyncio synchronization primitives has been deprecated. An asynchronous context manager must be used in order to acquire and release the synchronization resource. (Contributed by Andrew Svetlov in bpo-32253.)

The `asyncio.Task.current_task()` and `asyncio.Task.all_tasks()` methods have been deprecated. (Contributed by Andrew Svetlov in bpo-32250.)

11.3 collections

In Python 3.8, the abstract base classes in `collections.abc` will no longer be exposed in the regular `collections` module. This will help create a clearer distinction between the concrete classes and the abstract base classes. (Contributed by Serhiy Storchaka in bpo-25988.)

11.4 dbm

`dbm.dumb` now supports reading read-only files and no longer writes the index file when it is not changed. A deprecation warning is now emitted if the index file is missing and recreated in the `'r'` and `'w'` modes (this will be an error in future Python releases). (Contributed by Serhiy Storchaka in bpo-28847.)

11.5 enum

In Python 3.8, attempting to check for non-Enum objects in `Enum` classes will raise a `TypeError` (e.g. `1 in Color`); similarly, attempting to check for non-Flag objects in a `Flag` member will raise `TypeError` (e.g. `1 in Perm.RW`); currently, both operations return `False` instead. (Contributed by Ethan Furman in bpo-33217.)

11.6 gettext

Using non-integer value for selecting a plural form in `gettext` is now deprecated. It never correctly worked. (Contributed by Serhiy Storchaka in bpo-28692.)

11.7 importlib

Methods `MetaPathFinder.find_module()` (replaced by `MetaPathFinder.find_spec()`) and `PathEntryFinder.find_loader()` (replaced by `PathEntryFinder.find_spec()`) both deprecated in Python 3.4 now emit `DeprecationWarning`. (Contributed by Matthias Bussonnier in bpo-29576)

The `importlib.abc.ResourceLoader` ABC has been deprecated in favour of `importlib.abc.ResourceReader`.

11.8 locale

`locale.format()` has been deprecated, use `locale.format_string()` instead. (Contributed by Garvit in bpo-10379.)

11.9 macpath

The `macpath` is now deprecated and will be removed in Python 3.8. (Contributed by Chi Hsuan Yen in bpo-9850.)

11.10 threading

`dummy_threading` and `_dummy_thread` have been deprecated. It is no longer possible to build Python with threading disabled. Use `threading` instead. (Contributed by Antoine Pitrou in bpo-31370.)

11.11 socket

The silent argument value truncation in `socket.htons()` and `socket.ntohs()` has been deprecated. In future versions of Python, if the passed argument is larger than 16 bits, an exception will be raised. (Contributed by Oren Milman in bpo-28332.)

11.12 ssl

`ssl.wrap_socket()` is deprecated. Use `ssl.SSLContext.wrap_socket()` instead. (Contributed by Christian Heimes in bpo-28124.)

11.13 sunau

`sunau.openfp()` has been deprecated and will be removed in Python 3.9. Use `sunau.open()` instead. (Contributed by Brian Curtin in bpo-31985.)

11.14 sys

Deprecated `sys.set_coroutine_wrapper()` and `sys.get_coroutine_wrapper()`.

The undocumented `sys.callstats()` function has been deprecated and will be removed in a future Python version. (Contributed by Victor Stinner in bpo-28799.)

11.15 wave

`wave.openfp()` has been deprecated and will be removed in Python 3.9. Use `wave.open()` instead. (Contributed by Brian Curtin in bpo-31985.)

12 Deprecated functions and types of the C API

Function `PySlice_GetIndicesEx()` is deprecated and replaced with a macro if `Py_LIMITED_API` is not set or set to a value in the range between `0x03050400` and `0x03060000` (not inclusive), or is `0x03060100` or higher. (Contributed by Serhiy Storchaka in bpo-27867.)

`PyOS_AfterFork()` has been deprecated. Use `PyOS_BeforeFork()`, `PyOS_AfterFork_Parent()` or `PyOS_AfterFork_Child()` instead. (Contributed by Antoine Pitrou in bpo-16500.)

13 Platform Support Removals

- FreeBSD 9 and older are no longer officially supported.
- For full Unicode support, including within extension modules, *nix platforms are now expected to provide at least one of `C.UTF-8` (full locale), `C.utf8` (full locale) or `UTF-8` (`LC_CTYPE`-only locale) as an alternative to the legacy `ASCII`-based `C` locale.
- OpenSSL 0.9.8 and 1.0.1 are no longer supported, which means building CPython 3.7 with SSL/TLS support on older platforms still using these versions requires custom build options that link to a more recent version of OpenSSL.

  Notably, this issue affects the Debian 8 (aka "jessie") and Ubuntu 14.04 (aka "Trusty") LTS Linux distributions, as they still use OpenSSL 1.0.1 by default.

  Debian 9 ("stretch") and Ubuntu 16.04 ("xenial"), as well as recent releases of other LTS Linux releases (e.g. RHEL/CentOS 7.5, SLES 12-SP3), use OpenSSL 1.0.2 or later, and remain supported in the default build configuration.

CPython's own CI configuration file provides an example of using the SSL compatibility testing infrastructure in CPython's test suite to build and link against OpenSSL 1.1.0 rather than an outdated system provided OpenSSL.

14 API and Feature Removals

The following features and APIs have been removed from Python 3.7:

- The `os.stat_float_times()` function has been removed. It was introduced in Python 2.3 for backward compatibility with Python 2.2, and was deprecated since Python 3.1.
- Unknown escapes consisting of `'\'` and an ASCII letter in replacement templates for `re.sub()` were deprecated in Python 3.5, and will now cause an error.
- Removed support of the *exclude* argument in `tarfile.TarFile.add()`. It was deprecated in Python 2.7 and 3.2. Use the *filter* argument instead.
- The `splitunc()` function in the `ntpath` module was deprecated in Python 3.1, and has now been removed. Use the `splitdrive()` function instead.
- `collections.namedtuple()` no longer supports the *verbose* parameter or `_source` attribute which showed the generated source code for the named tuple class. This was part of an optimization designed to speed-up class creation. (Contributed by Jelle Zijlstra with further improvements by INADA Naoki, Serhiy Storchaka, and Raymond Hettinger in bpo-28638.)
- Functions `bool()`, `float()`, `list()` and `tuple()` no longer take keyword arguments. The first argument of `int()` can now be passed only as positional argument.
- Removed previously deprecated in Python 2.4 classes `Plist`, `Dict` and `_InternalDict` in the `plistlib` module. Dict values in the result of functions `readPlist()` and `readPlistFromBytes()` are now normal dicts. You no longer can use attribute access to access items of these dictionaries.
- The `asyncio.windows_utils.socketpair()` function has been removed. Use the `socket.socketpair()` function instead, it is available on all platforms since Python 3.5. `asyncio.windows_utils.socketpair` was just an alias to `socket.socketpair` on Python 3.5 and newer.
- `asyncio` no longer exports the `selectors` and `_overlapped` modules as `asyncio.selectors` and `asyncio._overlapped`. Replace `from asyncio import selectors` with `import selectors`.
- Direct instantiation of `ssl.SSLSocket` and `ssl.SSLObject` objects is now prohibited. The constructors were never documented, tested, or designed as public constructors. Users were supposed to use `ssl.wrap_socket()` or `ssl.SSLContext`. (Contributed by Christian Heimes in bpo-32951.)
- The unused `distutils install_misc` command has been removed. (Contributed by Eric N. Vander Weele in bpo-29218.)

15 Module Removals

The `fpectl` module has been removed. It was never enabled by default, never worked correctly on x86-64, and it changed the Python ABI in ways that caused unexpected breakage of C extensions. (Contributed by Nathaniel J. Smith in bpo-29137.)

16 Windows-only Changes

The python launcher, (py.exe), can accept 32 & 64 bit specifiers **without** having to specify a minor version as well. So `py -3-32` and `py -3-64` become valid as well as `py -3.7-32`, also the *-m*-64 and *-m.n*-64 forms are now accepted to force 64 bit python even if 32 bit would have otherwise been used. If the specified version is not available py.exe will error exit. (Contributed by Steve Barnes in bpo-30291.)

The launcher can be run as `py -0` to produce a list of the installed pythons, *with default marked with an asterisk*. Running `py -0p` will include the paths. If py is run with a version specifier that cannot be matched it will also print the *short form* list of available specifiers. (Contributed by Steve Barnes in bpo-30362.)

17 Porting to Python 3.7

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

17.1 Changes in Python Behavior

- `async` and `await` names are now reserved keywords. Code using these names as identifiers will now raise a `SyntaxError`. (Contributed by Jelle Zijlstra in bpo-30406.)
- **PEP 479** is enabled for all code in Python 3.7, meaning that `StopIteration` exceptions raised directly or indirectly in coroutines and generators are transformed into `RuntimeError` exceptions. (Contributed by Yury Selivanov in bpo-32670.)
- `object.__aiter__()` methods can no longer be declared as asynchronous. (Contributed by Yury Selivanov in bpo-31709.)
- Due to an oversight, earlier Python versions erroneously accepted the following syntax:

```
f(1 for x in [1],)

class C(1 for x in [1]):
    pass
```

  Python 3.7 now correctly raises a `SyntaxError`, as a generator expression always needs to be directly inside a set of parentheses and cannot have a comma on either side, and the duplication of the parentheses can be omitted only on calls. (Contributed by Serhiy Storchaka in bpo-32012 and bpo-32023.)

- When using the `-m` switch, the initial working directory is now added to `sys.path`, rather than an empty string (which dynamically denoted the current working directory at the time of each import). Any programs that are checking for the empty string, or otherwise relying on the previous behaviour, will need to be updated accordingly (e.g. by also checking for `os.getcwd()` or `os.path.dirname(__main__.__file__)`, depending on why the code was checking for the empty string in the first place).

17.2 Changes in the Python API

- `socketserver.ThreadingMixIn.server_close()` now waits until all non-daemon threads complete. Set the new `socketserver.ThreadingMixIn.block_on_close` class attribute to `False` to get the pre-3.7 behaviour. (Contributed by Victor Stinner in bpo-31233 and bpo-33540.)
- `socketserver.ForkingMixIn.server_close()` now waits until all child processes complete. Set the new `socketserver.ForkingMixIn.block_on_close` class attribute to `False` to get the pre-3.7 behaviour. (Contributed by Victor Stinner in bpo-31151 and bpo-33540.)
- The `locale.localeconv()` function now temporarily sets the `LC_CTYPE` locale to the value of `LC_NUMERIC` in some cases. (Contributed by Victor Stinner in bpo-31900.)
- `pkgutil.walk_packages()` now raises a `ValueError` if *path* is a string. Previously an empty list was returned. (Contributed by Sanyam Khurana in bpo-24744.)
- A format string argument for `string.Formatter.format()` is now positional-only. Passing it as a keyword argument was deprecated in Python 3.5. (Contributed by Serhiy Storchaka in bpo-29193.)
- Attributes `key`, `value` and `coded_value` of class `http.cookies.Morsel` are now read-only. Assigning to them was deprecated in Python 3.5. Use the `set()` method for setting them. (Contributed by Serhiy Storchaka in bpo-29192.)
- The *mode* argument of `os.makedirs()` no longer affects the file permission bits of newly-created intermediate-level directories. To set their file permission bits you can set the umask before invoking `makedirs()`. (Contributed by Serhiy Storchaka in bpo-19930.)
- The `struct.Struct.format` type is now `str` instead of `bytes`. (Contributed by Victor Stinner in bpo-21071.)
- `parse_multipart()` now accepts the *encoding* and *errors* arguments and returns the same results as `FieldStorage`: for non-file fields, the value associated to a key is a list of strings, not bytes. (Contributed by Pierre Quentel in bpo-29979.)
- Due to internal changes in `socket`, calling `socket.fromshare()` on a socket created by `socket.share` in older Python versions is not supported.
- `repr` for `BaseException` has changed to not include the trailing comma. Most exceptions are affected by this change. (Contributed by Serhiy Storchaka in bpo-30399.)
- `repr` for `datetime.timedelta` has changed to include the keyword arguments in the output. (Contributed by Utkarsh Upadhyay in bpo-30302.)
- Because `shutil.rmtree()` is now implemented using the `os.scandir()` function, the user specified handler *onerror* is now called with the first argument `os.scandir` instead of `os.listdir` when listing the directory is failed.
- Support for nested sets and set operations in regular expressions as in Unicode Technical Standard #18 might be added in the future. This would change the syntax. To facilitate this future change a `FutureWarning` will be raised in ambiguous cases for the time being. That include sets starting with a literal `'['` or containing literal character sequences `'--'`, `'&&'`, `'~~'`, and `'||'`. To avoid a warning, escape them with a backslash. (Contributed by Serhiy Storchaka in bpo-30349.)
- The result of splitting a string on a `regular expression` that could match an empty string has been changed. For example splitting on `r'\s*'` will now split not only on whitespaces as it did previously, but also on empty strings before all non-whitespace characters and just before the end of the string. The previous behavior can be restored by changing the pattern to `r'\s+'`. A `FutureWarning` was emitted for such patterns since Python 3.5.

  For patterns that match both empty and non-empty strings, the result of searching for all matches may also be changed in other cases. For example in the string `'a\n\n'`, the pattern `r'(?m)^\s*?$'` will not only match

empty strings at positions 2 and 3, but also the string `'\n'` at positions 2–3. To match only blank lines, the pattern should be rewritten as `r'(?m)^[^\S\n]*$'`.

`re.sub()` now replaces empty matches adjacent to a previous non-empty match. For example `re.sub('x*', '-', 'abxd')` returns now `'-a-b--d-'` instead of `'-a-b-d-'` (the first minus between 'b' and 'd' replaces 'x', and the second minus replaces an empty string between 'x' and 'd').

(Contributed by Serhiy Storchaka in bpo-25054 and bpo-32308.)

- Change `re.escape()` to only escape regex special characters instead of escaping all characters other than ASCII letters, numbers, and `'_'`. (Contributed by Serhiy Storchaka in bpo-29995.)
- `tracemalloc.Traceback` frames are now sorted from oldest to most recent to be more consistent with `traceback`. (Contributed by Jesse Bakker in bpo-32121.)
- On OSes that support `socket.SOCK_NONBLOCK` or `socket.SOCK_CLOEXEC` bit flags, the `socket.type` no longer has them applied. Therefore, checks like `if sock.type == socket.SOCK_STREAM` work as expected on all platforms. (Contributed by Yury Selivanov in bpo-32331.)
- On Windows the default for the *close_fds* argument of `subprocess.Popen` was changed from `False` to `True` when redirecting the standard handles. If you previously depended on handles being inherited when using `subprocess.Popen` with standard io redirection, you will have to pass `close_fds=False` to preserve the previous behaviour, or use `STARTUPINFO.lpAttributeList`.
- `importlib.machinery.PathFinder.invalidate_caches()` – which implicitly affects `importlib.invalidate_caches()` – now deletes entries in `sys.path_importer_cache` which are set to `None`. (Contributed by Brett Cannon in bpo-33169.)
- In `asyncio`, `loop.sock_recv()`, `loop.sock_sendall()`, `loop.sock_accept()`, `loop.getaddrinfo()`, `loop.getnameinfo()` have been changed to be proper coroutine methods to match their documentation. Previously, these methods returned `asyncio.Future` instances. (Contributed by Yury Selivanov in bpo-32327.)
- `asyncio.Server.sockets` now returns a copy of the internal list of server sockets, instead of returning it directly. (Contributed by Yury Selivanov in bpo-32662.)
- `Struct.format` is now a `str` instance instead of a `bytes` instance. (Contributed by Victor Stinner in bpo-21071.)
- `argparse` subparsers can now be made mandatory by passing `required=True` to `ArgumentParser.add_subparsers()`. (Contributed by Anthony Sottile in bpo-26510.)
- `ast.literal_eval()` is now stricter. Addition and subtraction of arbitrary numbers are no longer allowed. (Contributed by Serhiy Storchaka in bpo-31778.)
- `Calendar.itermonthdates` will now consistently raise an exception when a date falls outside of the `0001-01-01` through `9999-12-31` range. To support applications that cannot tolerate such exceptions, the new `Calendar.itermonthdays3` and `Calendar.itermonthdays4` can be used. The new methods return tuples and are not restricted by the range supported by `datetime.date`. (Contributed by Alexander Belopolsky in bpo-28292.)
- `collections.ChainMap` now preserves the order of the underlying mappings. (Contributed by Raymond Hettinger in bpo-32792.)
- The `submit()` method of `concurrent.futures.ThreadPoolExecutor` and `concurrent.futures.ProcessPoolExecutor` now raises a `RuntimeError` if called during interpreter shutdown. (Contributed by Mark Nemec in bpo-33097.)
- The `configparser.ConfigParser` constructor now uses `read_dict()` to process the default values, making its behavior consistent with the rest of the parser. Non-string keys and values in the defaults dictionary are now being implicitly converted to strings. (Contributed by James Tocknell in bpo-23835.)

- Several undocumented internal imports were removed. One example is that `os.errno` is no longer available; use `import errno` directly instead. Note that such undocumented internal imports may be removed any time without notice, even in micro version releases.

17.3 Changes in the C API

The function `PySlice_GetIndicesEx()` is considered unsafe for resizable sequences. If the slice indices are not instances of `int`, but objects that implement the `__index__()` method, the sequence can be resized after passing its length to `PySlice_GetIndicesEx()`. This can lead to returning indices out of the length of the sequence. For avoiding possible problems use new functions `PySlice_Unpack()` and `PySlice_AdjustIndices()`. (Contributed by Serhiy Storchaka in bpo-27867.)

17.4 CPython bytecode changes

There are two new opcodes: `LOAD_METHOD` and `CALL_METHOD`. (Contributed by Yury Selivanov and INADA Naoki in bpo-26110.)

The `STORE_ANNOTATION` opcode has been removed. (Contributed by Mark Shannon in bpo-32550.)

17.5 Windows-only Changes

The file used to override `sys.path` is now called `<python-executable>._pth` instead of `'sys.path'`. See finding_modules for more information. (Contributed by Steve Dower in bpo-28137.)

17.6 Other CPython implementation changes

In preparation for potential future changes to the public CPython runtime initialization API (see **PEP 432** for an initial, but somewhat outdated, draft), CPython's internal startup and configuration management logic has been significantly refactored. While these updates are intended to be entirely transparent to both embedding applications and users of the regular CPython CLI, they're being mentioned here as the refactoring changes the internal order of various operations during interpreter startup, and hence may uncover previously latent defects, either in embedding applications, or in CPython itself. (Initially contributed by Nick Coghlan and Eric Snow as part of bpo-22257, and further updated by Nick, Eric, and Victor Stinner in a number of other issues). Some known details affected:

- `PySys_AddWarnOptionUnicode()` is not currently usable by embedding applications due to the requirement to create a Unicode object prior to calling *Py_Initialize*. Use `PySys_AddWarnOption()` instead.
- warnings filters added by an embedding application with `PySys_AddWarnOption()` should now more consistently take precedence over the default filters set by the interpreter

Due to changes in the way the default warnings filters are configured, setting `Py_BytesWarningFlag` to a value greater than one is no longer sufficient to both emit `BytesWarning` messages and have them converted to exceptions. Instead, the flag must be set (to cause the warnings to be emitted in the first place), and an explicit `error::BytesWarning` warnings filter added to convert them to exceptions.

Due to a change in the way docstrings are handled by the compiler, the implicit `return None` in a function body consisting solely of a docstring is now marked as occurring on the same line as the docstring, not on the function's header line.

The current exception state has been moved from the frame object to the co-routine. This simplified the interpreter and fixed a couple of obscure bugs caused by having swap exception state when entering or exiting a generator. (Contributed by Mark Shannon in bpo-25612.)

18 Notable changes in Python 3.7.1

Starting in 3.7.1, `Py_Initialize()` now consistently reads and respects all of the same environment settings as `Py_Main()` (in earlier Python versions, it respected an ill-defined subset of those environment variables, while in Python 3.7.0 it didn't read any of them due to bpo-34247). If this behavior is unwanted, set `Py_IgnoreEnvironmentFlag` to 1 before calling `Py_Initialize()`.

In 3.7.1 the C API for Context Variables was updated to use `PyObject` pointers. See also bpo-34762.

`xml.dom.minidom` and `xml.sax` modules no longer process external entities by default. See also bpo-17239.

In 3.7.1 the `tokenize` module now implicitly emits a `NEWLINE` token when provided with input that does not have a trailing new line. This behavior now matches what the C tokenizer does internally. (Contributed by Ammar Askar in bpo-33899.)

19 Notable changes in Python 3.7.2

In 3.7.2, `venv` on Windows no longer copies the original binaries, but creates redirector scripts named `python.exe` and `pythonw.exe` instead. This resolves a long standing issue where all virtual environments would have to be upgraded or recreated with each Python update. However, note that this release will still require recreation of virtual environments in order to get the new scripts.

20 Notable changes in Python 3.7.6

Due to significant security concerns, the *reuse_address* parameter of `asyncio.loop.create_datagram_endpoint()` is no longer supported. This is because of the behavior of the socket option `SO_REUSEADDR` in UDP. For more details, see the documentation for `loop.create_datagram_endpoint()`. (Contributed by Kyle Stanley, Antoine Pitrou, and Yury Selivanov in bpo-37228.)

21 Notable changes in Python 3.7.10

Earlier Python versions allowed using both `;` and `&` as query parameter separators in `urllib.parse.parse_qs()` and `urllib.parse.parse_qsl()`. Due to security concerns, and to conform with newer W3C recommendations, this has been changed to allow only a single separator key, with `&` as the default. This change also affects `cgi.parse()` and `cgi.parse_multipart()` as they use the affected functions internally. For more details, please see their respective documentation. (Contributed by Adam Goldschmidt, Senthil Kumaran and Ken Jin in bpo-42967.)

22 Notable changes in Python 3.7.11

A security fix alters the `ftplib.FTP` behavior to not trust the IPv4 address sent from the remote server when setting up a passive data channel. We reuse the ftp server IP address instead. For unusual code requiring the old behavior, set a `trust_server_pasv_ipv4_address` attribute on your FTP instance to `True`. (See bpo-43285)

The presence of newline or tab characters in parts of a URL allows for some forms of attacks. Following the WHATWG specification that updates RFC 3986, ASCII newline `\n`, `\r` and tab `\t` characters are stripped from the URL by the parser `urllib.parse()` preventing such attacks. The removal characters are controlled by a new module level variable `urllib.parse._UNSAFE_URL_BYTES_TO_REMOVE`. (See bpo-43882)

23 Notable security feature in 3.7.14

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.

Index