What's New in Python

Release 3.5.0

A. M. Kuchling

September 13, 2015

Python Software Foundation
Email: docs@python.org

Contents

1 Summary – Release highlights **3**

2 New Features **5**
2.1 PEP 492 - Coroutines with async and await syntax . . . 5
2.2 PEP 465 - A dedicated infix operator for matrix multiplication . . . 6
2.3 PEP 448 - Additional Unpacking Generalizations . . . 7
2.4 PEP 461 - % formatting support for bytes and bytearray . . . 7
2.5 PEP 484 - Type Hints . . . 8
2.6 PEP 471 - os.scandir() function – a better and faster directory iterator . . . 8
2.7 PEP 475: Retry system calls failing with EINTR . . . 8
2.8 PEP 479: Change StopIteration handling inside generators . . . 9
2.9 PEP 486: Make the Python Launcher aware of virtual environments . . . 9
2.10 PEP 488: Elimination of PYO files . . . 9
2.11 PEP 489: Multi-phase extension module initialization . . . 10
2.12 PEP 485: A function for testing approximate equality . . . 10

3 Other Language Changes **10**

4 New Modules **10**
4.1 zipapp . . . 10

5 Improved Modules **11**
5.1 argparse . . . 11
5.2 bz2 . . . 11
5.3 cgi . . . 11
5.4 cmath . . . 11
5.5 code . . . 11
5.6 collections . . . 11
5.7 collections.abc . . . 12
5.8 compileall . . . 12
5.9 concurrent.futures . . . 12
5.10 contextlib . . . 12
5.11 curses . . . 12
5.12 difflib . . . 12

5.13 distutils . . . 13
5.14 doctest . . . 13
5.15 email . . . 13
5.16 enum . . . 13
5.17 faulthandler . . . 13
5.18 functools . . . 13
5.19 glob . . . 14
5.20 heapq . . . 14
5.21 http . . . 14
5.22 idlelib and IDLE . . . 14
5.23 imaplib . . . 14
5.24 imghdr . . . 14
5.25 importlib . . . 14
5.26 inspect . . . 15
5.27 io . . . 15
5.28 ipaddress . . . 15
5.29 json . . . 15
5.30 locale . . . 15
5.31 logging . . . 16
5.32 lzma . . . 16
5.33 math . . . 16
5.34 operator . . . 16
5.35 os . . . 16
5.36 pathlib . . . 17
5.37 pickle . . . 17
5.38 poplib . . . 17
5.39 re . . . 17
5.40 readline . . . 17
5.41 selectors . . . 18
5.42 shutil . . . 18
5.43 signal . . . 18
5.44 smtpd . . . 18
5.45 smtplib . . . 18
5.46 sndhdr . . . 19
5.47 ssl . . . 19
Memory BIO Support . . . 19
Application-Layer Protocol Negotiation Support . . . 19
Other Changes . . . 19
5.48 socket . . . 20
5.49 sqlite3 . . . 20
5.50 subprocess . . . 20
5.51 sys . . . 20
5.52 sysconfig . . . 20
5.53 tarfile . . . 20
5.54 threading . . . 21
5.55 time . . . 21
5.56 timeit . . . 21
5.57 tkinter . . . 21
5.58 traceback . . . 21
5.59 types . . . 21
5.60 urllib . . . 21
5.61 unicodedata . . . 22
5.62 unittest . . . 22
5.63 unittest.mock . . . 22

5.64 wsgiref . . . 22
5.65 xmlrpc . . . 22
5.66 xml.sax . . . 22
5.67 zipfile . . . 22

6 Other module-level changes **23**

7 Optimizations **23**

8 Build and C API Changes **23**

9 Deprecated **25**
9.1 New Keywords . . . 25
9.2 Unsupported Operating Systems . . . 25
9.3 Deprecated Python modules, functions and methods . . . 25

10 Removed **25**
10.1 API and Feature Removals . . . 25

11 Porting to Python 3.5 **26**
11.1 Changes in the Python API . . . 26
11.2 Changes in the C API . . . 27

Index **28**

Release 3.5.0

Date September 13, 2015

Editors Elvis Pranskevichus <elprans@gmail.com>, Yury Selivanov <yselivanov@gmail.com>

Python 3.5 was released on September 13, 2015.

This article explains the new features in Python 3.5, compared to 3.4. For full details, see the changelog.

See also:

PEP 478 - Python 3.5 Release Schedule

1 Summary – Release highlights

New syntax features:

- **PEP 492**, coroutines with async and await syntax.
- **PEP 465**, a new matrix multiplication operator: `a @ b`.
- **PEP 448**, additional unpacking generalizations.

New library modules:

- `zipapp`: *Improving Python ZIP Application Support* (**PEP 441**).

New built-in features:

- `bytes % args`, `bytearray % args`: **PEP 461** - Adding `%` formatting to bytes and bytearray.

- `b'\xf0\x9f\x90\x8d'.hex()`, `bytearray(b'\xf0\x9f\x90\x8d').hex()`, `memoryview(b'\xf0\x9f\x90\x8d').hex()`: issue 9951 - A `hex` method has been added to bytes, bytearray, and memoryview.
- `memoryview` (including multi-dimensional) now supports tuple indexing. (Contributed by Antoine Pitrou in issue 23632.)
- Generators have new `gi_yieldfrom` attribute, which returns the object being iterated by `yield from` expressions. (Contributed by Benno Leslie and Yury Selivanov in issue 24450.)
- New `RecursionError` exception. (Contributed by Georg Brandl in issue 19235.)
- New `StopAsyncIteration` exception. (Contributed by Yury Selivanov in issue 24017. See also **PEP 492**.)

CPython implementation improvements:

- When the `LC_TYPE` locale is the POSIX locale (`C` locale), `sys.stdin` and `sys.stdout` are now using the `surrogateescape` error handler, instead of the `strict` error handler. (Contributed by Victor Stinner in issue 19977.)
- `.pyo` files are no longer used and have been replaced by a more flexible scheme that inclides the optimization level explicitly in `.pyc` name. (**PEP 488**)
- Builtin and extension modules are now initialized in a multi-phase process, which is similar to how Python modules are loaded. (**PEP 489**).

Significantly Improved Library Modules:

- `collections.OrderedDict` is now implemented in C, which makes it 4 to 100 times faster. (Contributed by Eric Snow in issue 16991.)
- You may now pass bytes to the `tempfile` module's APIs and it will return the temporary pathname as `bytes` instead of `str`. It also accepts a value of `None` on parameters where only str was accepted in the past to do the right thing based on the types of the other inputs. Two functions, `gettempdirb()` and `gettempprefixb()`, have been added to go along with this. This behavior matches that of the `os` APIs. (Contributed by Gregory P. Smith in issue 24230.)
- `ssl` module gained support for Memory BIO, which decouples SSL protocol handling from network IO. (Contributed by Geert Jansen in issue 21965.)
- `traceback` has new lightweight and convenient to work with classes `TracebackException`, `StackSummary`, and `FrameSummary`. (Contributed by Robert Collins in issue 17911.)
- Most of `functools.lru_cache()` machinery is now implemented in C. (Contributed by Matt Joiner, Alexey Kachayev, and Serhiy Storchaka in issue 14373.)

Security improvements:

- SSLv3 is now disabled throughout the standard library. It can still be enabled by instantiating a `ssl.SSLContext` manually. (See issue 22638 for more details; this change was backported to CPython 3.4 and 2.7.)
- HTTP cookie parsing is now stricter, in order to protect against potential injection attacks. (Contributed by Antoine Pitrou in issue 22796.)

Windows improvements:

- A new installer for Windows has replaced the old MSI. See *using-on-windows* for more information.
- Windows builds now use Microsoft Visual C++ 14.0, and extension modules should use the same.

Please read on for a comprehensive list of user-facing changes, including many other smaller improvements, CPython optimizations, deprecations, and potential porting issues.

2 New Features

2.1 PEP 492 - Coroutines with async and await syntax

PEP 492 greatly improves support for asynchronous programming in Python by adding *awaitable objects*, *coroutine functions*, *asynchronous iteration*, and *asynchronous context managers*.

Coroutine functions are declared using the new `async def` syntax:

```
>>> async def coro():
...     return 'spam'
```

Inside a coroutine function, the new `await` expression can be used to suspend coroutine execution until the result is available. Any object can be *awaited*, as long as it implements the *awaitable* protocol by defining the `__await__()` method.

PEP 492 also adds `async for` statement for convenient iteration over asynchronous iterables.

An example of a simple HTTP client written using the new syntax:

```
import asyncio

async def http_get(domain):
    reader, writer = await asyncio.open_connection(domain, 80)

    writer.write(b'\r\n'.join([
        b'GET / HTTP/1.1',
        b'Host: %b' % domain.encode('latin-1'),
        b'Connection: close',
        b'', b''
    ]))

    async for line in reader:
        print('>>>', line)

    writer.close()

loop = asyncio.get_event_loop()
try:
    loop.run_until_complete(http_get('example.com'))
finally:
    loop.close()
```

Similarly to asynchronous iteration, there is a new syntax for asynchronous context managers. The following script:

```
import asyncio

async def coro(name, lock):
    print('coro {}: waiting for lock'.format(name))
    async with lock:
        print('coro {}: holding the lock'.format(name))
        await asyncio.sleep(1)
        print('coro {}: releasing the lock'.format(name))

loop = asyncio.get_event_loop()
lock = asyncio.Lock()
coros = asyncio.gather(coro(1, lock), coro(2, lock))
```

```
try:
    loop.run_until_complete(coros)
finally:
    loop.close()
```

will print:

```
coro 2: waiting for lock
coro 2: holding the lock
coro 1: waiting for lock
coro 2: releasing the lock
coro 1: holding the lock
coro 1: releasing the lock
```

Note that both `async for` and `async with` can only be used inside a coroutine function declared with `async def`.

Coroutine functions are intended to be run inside a compatible event loop, such as `asyncio.Loop`.

See also:

PEP 492 – Coroutines with async and await syntax PEP written and implemented by Yury Selivanov.

2.2 PEP 465 - A dedicated infix operator for matrix multiplication

PEP 465 adds the `@` infix operator for matrix multiplication. Currently, no builtin Python types implement the new operator, however, it can be implemented by defining `__matmul__()`, `__rmatmul__()`, and `__imatmul__()` for regular, reflected, and in-place matrix multiplication. The semantics of these methods is similar to that of methods defining other infix arithmetic operators.

Matrix multiplication is a notably common operation in many fields of mathematics, science, engineering, and the addition of `@` allows writing cleaner code:

```
S = (H @ beta - r).T @ inv(H @ V @ H.T) @ (H @ beta - r)
```

instead of:

```
S = dot((dot(H, beta) - r).T,
        dot(inv(dot(dot(H, V), H.T)), dot(H, beta) - r))
```

An upcoming release of NumPy 1.10 will add support for the new operator:

```
>>> import numpy

>>> x = numpy.ones(3)
>>> x
array([ 1., 1., 1.])

>>> m = numpy.eye(3)
>>> m
array([[ 1., 0., 0.],
       [ 0., 1., 0.],
       [ 0., 0., 1.]])

>>> x @ m
array([ 1., 1., 1.])
```

See also:

PEP 465 – A dedicated infix operator for matrix multiplication PEP written by Nathaniel J. Smith; implemented by Benjamin Peterson.

2.3 PEP 448 - Additional Unpacking Generalizations

PEP 448 extends the allowed uses of the `*` iterable unpacking operator and `**` dictionary unpacking operator. It is now possible to use an arbitrary number of unpackings in function calls:

```
>>> print(*[1], *[2], 3, *[4, 5])
1 2 3 4 5

>>> def fn(a, b, c, d):
...     print(a, b, c, d)
...

>>> fn(**{'a': 1, 'c': 3}, **{'b': 2, 'd': 4})
1 2 3 4
```

Similarly, tuple, list, set, and dictionary displays allow multiple unpackings:

```
>>> *range(4), 4
(0, 1, 2, 3, 4)

>>> [*range(4), 4]
[0, 1, 2, 3, 4]

>>> {*range(4), 4, *(5, 6, 7)}
{0, 1, 2, 3, 4, 5, 6, 7}

>>> {'x': 1, **{'y': 2}}
{'x': 1, 'y': 2}
```

See also:

PEP 448 – Additional Unpacking Generalizations PEP written by Joshua Landau; implemented by Neil Girdhar, Thomas Wouters, and Joshua Landau.

2.4 PEP 461 - % formatting support for bytes and bytearray

PEP 461 adds % formatting to `bytes` and `bytearray`, aiding in handling data that is a mixture of binary and ASCII compatible text. This feature also eases porting such code from Python 2.

Examples:

```
>>> b'Hello %s!' % b'World'
b'Hello World!'

>>> b'x=%i y=%f' % (1, 2.5)
b'x=1 y=2.500000'
```

Unicode is not allowed for `%s`, but it is accepted by `%a` (equivalent of `repr(obj).encode('ascii', 'backslashreplace')`):

```
>>> b'Hello %s!' % 'World'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: %b requires bytes, or an object that implements __bytes__, not 'str'
```

```
>>> b'price: %a' % '10€'
b"price: '10\\u20ac'"
```

See also:

PEP 461 – Adding % formatting to bytes and bytearray PEP written by Ethan Furman; implemented by Neil Schemenauer and Ethan Furman.

2.5 PEP 484 - Type Hints

This PEP introduces a provisional module to provide these standard definitions and tools, along with some conventions for situations where annotations are not available.

For example, here is a simple function whose argument and return type are declared in the annotations:

```
def greeting(name: str) -> str:
    return 'Hello ' + name
```

The type system supports unions, generic types, and a special type named `Any` which is consistent with (i.e. assignable to and from) all types.

See also:

- `typing` module documentation
- **PEP 484 – Type Hints** PEP written by Guido van Rossum, Jukka Lehtosalo, and Łukasz Langa; implemented by Guido van Rossum.

2.6 PEP 471 - os.scandir() function – a better and faster directory iterator

PEP 471 adds a new directory iteration function, `os.scandir()`, to the standard library. Additionally, `os.walk()` is now implemented using `os.scandir()`, which speeds it up by 3-5 times on POSIX systems and by 7-20 times on Windows systems.

See also:

PEP 471 – os.scandir() function – a better and faster directory iterator PEP written and implemented by Ben Hoyt with the help of Victor Stinner.

2.7 PEP 475: Retry system calls failing with EINTR

PEP 475 adds support for automatic retry of system calls failing with `EINTR`: this means that user code doesn't have to deal with EINTR or `InterruptedError` manually, and should make it more robust against asynchronous signal reception.

Examples of functions which are now retried when interrupted by a signal instead of raising `InterruptedError` if the Python signal handler does not raise an exception:

- `open()`, `os.open()`, `io.open()`;
- functions of the `faulthandler` module;
- `os` functions: `fchdir()`, `fchmod()`, `fchown()`, `fdatasync()`, `fstat()`, `fstatvfs()`, `fsync()`, `ftruncate()`, `mkfifo()`, `mknod()`, `posix_fadvise()`, `posix_fallocate()`, `pread()`, `pwrite()`, `read()`, `readv()`, `sendfile()`, `wait3()`, `wait4()`, `wait()`, `waitid()`, `waitpid()`, `write()`, `writev()`;

- special cases: `os.close()` and `os.dup2()` now ignore `EINTR` error, the syscall is not retried (see the PEP for the rationale);
- `select` functions: `poll()`, `poll()`, `control()`, `poll()`, `select()`;
- `socket.socket()` methods: `accept()`, `connect()` (except for non-blocking sockets), `recv()`, `recvfrom()`, `recvmsg()`, `send()`, `sendall()`, `sendmsg()`, `sendto()`;
- `signal.sigtimedwait()`, `signal.sigwaitinfo()`;
- `time.sleep()`.

See also:

PEP 475 – Retry system calls failing with EINTR PEP and implementation written by Charles-François Natali and Victor Stinner, with the help of Antoine Pitrou (the french connection).

2.8 PEP 479: Change StopIteration handling inside generators

PEP 479 changes the behavior of generators: when a `StopIteration` exception is raised inside a generator, it is replaced with a `RuntimeError`. To enable the feature a `__future__` import should be used:

```
from __future__ import generator_stop
```

Without a `__future__` import, a `PendingDeprecationWarning` will be raised.

See also:

PEP 479 – Change StopIteration handling inside generators PEP written by Chris Angelico and Guido van Rossum. Implemented by Chris Angelico, Yury Selivanov and Nick Coghlan.

2.9 PEP 486: Make the Python Launcher aware of virtual environments

PEP 486 makes the Windows launcher (see **PEP 397**) aware of an active virtual environment. When the default interpreter would be used and the `VIRTUAL_ENV` environment variable is set, the interpreter in the virtual environment will be used.

See also:

PEP 486 – Make the Python Launcher aware of virtual environments PEP written and implemented by Paul Moore.

2.10 PEP 488: Elimination of PYO files

PEP 488 does away with the concept of `.pyo` files. This means that `.pyc` files represent both unoptimized and optimized bytecode. To prevent the need to constantly regenerate bytecode files, `.pyc` files now have an optional `opt-` tag in their name when the bytecode is optimized. This has the side-effect of no more bytecode file name clashes when running under either *-O* or *-OO*. Consequently, bytecode files generated from *-O*, and *-OO* may now exist simultaneously. `importlib.util.cache_from_source()` has an updated API to help with this change.

See also:

PEP 488 – Elimination of PYO files PEP written and implemented by Brett Cannon.

2.11 PEP 489: Multi-phase extension module initialization

PEP 489 updates extension module initialization to take advantage of the two step module loading mechanism introduced by PEP 451 in Python 3.4.

This change brings the import semantics of extension modules that opt-in to using the new mechanism much closer to those of Python source and bytecode modules, including the ability to use any valid identifier as a module name, rather than being restricted to ASCII.

See also:

PEP 489 – Multi-phase extension module initialization PEP written by Petr Viktorin, Stefan Behnel, and Nick Coghlan; implemented by Petr Viktorin.

2.12 PEP 485: A function for testing approximate equality

PEP 485 adds the `math.isclose()` and `cmath.isclose()` functions which tell whether two values are approximately equal or "close" to each other. Whether or not two values are considered close is determined according to given absolute and relative tolerances.

See also:

PEP 485 – A function for testing approximate equality PEP written by Christopher Barker; implemented by Chris Barker and Tal Einat.

3 Other Language Changes

Some smaller changes made to the core Python language are:

- Added the `"namereplace"` error handlers. The `"backslashreplace"` error handlers now works with decoding and translating. (Contributed by Serhiy Storchaka in issue 19676 and issue 22286.)
- The *-b* option now affects comparisons of `bytes` with `int`. (Contributed by Serhiy Storchaka in issue 23681.)
- New Kazakh *codec* `kz1048`. (Contributed by Serhiy Storchaka in issue 22682.)
- Property docstrings are now writable. This is especially useful for `collections.namedtuple()` docstrings. (Contributed by Berker Peksag in issue 24064.)
- New Tajik *codec* `koi8_t`. (Contributed by Serhiy Storchaka in issue 22681.)
- Circular imports involving relative imports are now supported. (Contributed by Brett Cannon and Antoine Pitrou in issue 17636.)

4 New Modules

4.1 zipapp

The new `zipapp` module (specified in PEP 441) provides an API and command line tool for creating executable Python Zip Applications, which were introduced in Python 2.6 in issue 1739468, but which were not well publicized, either at the time or since.

With the new module, bundling your application is as simple as putting all the files, including a `__main__.py` file, into a directory `myapp` and running:

```
$ python -m zipapp myapp
$ python myapp.pyz
```

The module implementation has been contributed by Paul Moore in issue 23491.

See also:

PEP 441 – Improving Python ZIP Application Support

5 Improved Modules

5.1 argparse

The `ArgumentParser` class now allows to disable *abbreviated usage* of long options by setting *allow_abbrev* to `False`. (Contributed by Jonathan Paugh, Steven Bethard, paul j3 and Daniel Eriksson in issue 14910.)

5.2 bz2

The `BZ2Decompressor.decompress` method now accepts an optional *max_length* argument to limit the maximum size of decompressed data. (Contributed by Nikolaus Rath in issue 15955.)

5.3 cgi

The `FieldStorage` class now supports the context management protocol. (Contributed by Berker Peksag in issue 20289.)

5.4 cmath

A new function `isclose()` provides a way to test for approximate equality. (Contributed by Chris Barker and Tal Einat in issue 24270.)

5.5 code

The `InteractiveInterpreter.showtraceback` method now prints the full chained traceback, just like the interactive interpreter. (Contributed by Claudiu Popa in issue 17442.)

5.6 collections

The `OrderedDict` class is now implemented in C, which makes it 4 to 100 times faster. (Contributed by Eric Snow in issue 16991.)

`OrderedDict.items`, `OrderedDict.keys`, `OrderedDict.values` views now support `reversed()` iteration. (Contributed by Serhiy Storchaka in issue 19505.)

The `deque` class now defines `index()`, `insert()`, and `copy()`, as well as supports `+` and `*` operators. This allows deques to be recognized as a `MutableSequence` and improves their substitutability for lists. (Contributed by Raymond Hettinger issue 23704.)

Docstrings produced by `namedtuple()` can now be updated:

```
Point = namedtuple('Point', ['x', 'y'])
Point.__doc__ = 'ordered pair'
Point.x.__doc__ = 'abscissa'
Point.y.__doc__ = 'ordinate'
```

(Contributed by Berker Peksag in issue 24064.)

The `UserString` class now implements `__getnewargs__()`, `__rmod__()`, `casefold()`, `format_map()`, `isprintable()`, and `maketrans()` methods to match corresponding methods of `str`. (Contributed by Joe Jevnik in issue 22189.)

5.7 collections.abc

A new `Generator` abstract base class. (Contributed by Stefan Behnel in issue 24018.)

New `Coroutine`, `AsyncIterator`, and `AsyncIterable` abstract base classes. (Contributed by Yury Selivanov in issue 24184.)

5.8 compileall

A new `compileall` option, `-j N`, allows to run `N` workers sumultaneously to perform parallel bytecode compilation. The `compile_dir()` function has a corresponding `workers` parameter. (Contributed by Claudiu Popa in issue 16104.)

The `-q` command line option can now be specified more than once, in which case all output, including errors, will be suppressed. The corresponding `quiet` parameter in `compile_dir()`, `compile_file()`, and `compile_path()` can now accept an integer value indicating the level of output suppression. (Contributed by Thomas Kluyver in issue 21338.)

5.9 concurrent.futures

The `Executor.map` method now accepts a *chunksize* argument to allow batching of tasks to improve performance when `ProcessPoolExecutor()` is used. (Contributed by Dan O'Reilly in issue 11271.)

5.10 contextlib

The new `redirect_stderr()` context manager (similar to `redirect_stdout()`) makes it easier for utility scripts to handle inflexible APIs that write their output to `sys.stderr` and don't provide any options to redirect it. (Contributed by Berker Peksag in issue 22389.)

5.11 curses

The new `update_lines_cols()` function updates `LINES` and `COLS` environment variables. This is useful for detecting manual screen resize. (Contributed by Arnon Yaari in issue 4254.)

5.12 difflib

The charset of HTML documents generated by `HtmlDiff.make_file` can now be customized by using a new *charset* keyword-only argument. The default charset of HTML document changed from `"ISO-8859-1"` to `"utf-8"`. (Contributed by Berker Peksag in issue 2052.)

The `diff_bytes()` function can now compare lists of byte strings. This fixes a regression from Python 2. (Contributed by Terry J. Reedy and Greg Ward in issue 17445.)

5.13 distutils

Both `build` and `build_ext` commands now accept a `-j` option to enable parallel building of extension modules. (Contributed by Antoine Pitrou in issue 5309.)

The `distutils` module now supports `xz` compression, and can be enabled by passing `xztar` as an argument to `bdist --format`. (Contributed by Serhiy Storchaka in issue 16314.)

5.14 doctest

The `DocTestSuite()` function returns an empty `unittest.TestSuite` if *module* contains no docstrings instead of raising `ValueError`. (Contributed by Glenn Jones in issue 15916.)

5.15 email

A new policy option `Policy.mangle_from_` controls whether or not lines that start with `"From "` in email bodies are prefixed with a `">"` character by generators. The default is `True` for `compat32` and `False` for all other policies. (Contributed by Milan Oberkirch in issue 20098.)

A new `Message.get_content_disposition` method provides easy access to a canonical value for the *`Content-Disposition`* header. (Contributed by Abhilash Raj in issue 21083.)

A new policy option `EmailPolicy.utf8` can be set to `True` to encode email headers using the UTF-8 charset instead of using encoded words. This allows `Messages` to be formatted according to **RFC 6532** and used with an SMTP server that supports the **RFC 6531** `SMTPUTF8` extension. (Contributed by R. David Murray in issue 24211.)

5.16 enum

The `Enum` callable has a new parameter *start* to specify the initial number of enum values if only *names* are provided:

```
>>> Animal = enum.Enum('Animal', 'cat dog', start=10)
>>> Animal.cat
<Animal.cat: 10>
>>> Animal.dog
<Animal.dog: 11>
```

(Contributed by Ethan Furman in issue 21706.)

5.17 faulthandler

`enable()`, `register()`, `dump_traceback()` and `dump_traceback_later()` functions now accept file descriptors in addition to file-like objects. (Contributed by Wei Wu in issue 23566.)

5.18 functools

Most of `lru_cache()` machinery is now implemented in C, making it significantly faster. (Contributed by Matt Joiner, Alexey Kachayev, and Serhiy Storchaka in issue 14373.)

5.19 glob

`iglob()` and `glob()` functions now support recursive search in subdirectories using the `"**"` pattern. (Contributed by Serhiy Storchaka in issue 13968.)

5.20 heapq

Element comparison in `merge()` can now be customized by passing a *key function* in a new optional `key` keyword argument. A new optional `reverse` keyword argument can be used to reverse element comparison. (Contributed by Raymond Hettinger in issue 13742.)

5.21 http

A new `HTTPStatus` enum that defines a set of HTTP status codes, reason phrases and long descriptions written in English. (Contributed by Demian Brecht in issue 21793.)

5.22 idlelib and IDLE

Since idlelib implements the IDLE shell and editor and is not intended for import by other programs, it gets improvements with every release. See `Lib/idlelib/NEWS.txt` for a cumulative list of changes since 3.4.0, as well as changes made in future 3.5.x releases. This file is also available from the IDLE Help -> About Idle dialog.

5.23 imaplib

The `IMAP4` class now supports context manager protocol. When used in a `with` statement, the IMAP4 `LOGOUT` command will be called automatically at the end of the block. (Contributed by Tarek Ziadé and Serhiy Storchaka in issue 4972.)

The `imaplib` module now supports **RFC 5161** (ENABLE Extension) and **RFC 6855** (UTF-8 Support) via the `IMAP4.enable` method. A new `IMAP4.utf8_enabled` attribute, tracks whether or not **RFC 6855** support is enabled. (Contributed by Milan Oberkirch, R. David Murray, and Maciej Szulik in issue 21800.)

The `imaplib` module now automatically encodes non-ASCII string usernames and passwords using UTF-8, as recommended by the RFCs. (Contributed by Milan Oberkirch in issue 21800.)

5.24 imghdr

The `what()` function now recognizes the OpenEXR format (contributed by Martin Vignali and Claudiu Popa in issue 20295), and the WebP format (contributed by Fabrice Aneche and Claudiu Popa in issue 20197.)

5.25 importlib

The `util.LazyLoader` class allows for lazy loading of modules in applications where startup time is important. (Contributed by Brett Cannon in issue 17621.)

The `abc.InspectLoader.source_to_code` method is now a static method. This makes it easier to initialize a module object with code compiled from a string by running `exec(code, module.__dict__)`. (Contributed by Brett Cannon in issue 21156.)

The new `util.module_from_spec` function is now the preferred way to create a new module. As opposed to creating a `types.ModuleType` instance directly, this new function will set the various import-controlled attributes based on the passed-in spec object. (Contributed by Brett Cannon in issue 20383.)

5.26 inspect

Both `Signature` and `Parameter` classes are now picklable and hashable. (Contributed by Yury Selivanov in issue 20726 and issue 20334.)

A new `BoundArguments.apply_defaults` method provides a way to set default values for missing arguments. (Contributed by Yury Selivanov in issue 24190.)

A new class method `Signature.from_callable` makes subclassing of `Signature` easier. (Contributed by Yury Selivanov and Eric Snow in issue 17373.)

The `signature()` function now accepts a `follow_wrapped` optional keyword argument, which, when set to `False`, disables automatic following of `__wrapped__` links. (Contributed by Yury Selivanov in issue 20691.)

A set of new functions to inspect *coroutine functions* and *coroutine objects* has been added: `iscoroutine()`, `iscoroutinefunction()`, `isawaitable()`, `getcoroutinelocals()`, and `getcoroutinestate()`. (Contributed by Yury Selivanov in issue 24017 and issue 24400.)

`stack()`, `trace()`, `getouterframes()`, and `getinnerframes()` functions now return a list of named tuples. (Contributed by Daniel Shahaf in issue 16808.)

5.27 io

A new `BufferedIOBase.readinto1` method, that uses at most one call to the underlying raw stream's `RawIOBase.read` (or `RawIOBase.readinto`) method. (Contributed by Nikolaus Rath in issue 20578.)

5.28 ipaddress

Both `IPv4Network` and `IPv6Network` classes now accept an `(address, netmask)` tuple argument, so as to easily construct network objects from existing addresses. (Contributed by Peter Moody and Antoine Pitrou in issue 16531.)

A new `reverse_pointer>` attribute for `IPv4Network` and `IPv6Network` classes returns the name of the reverse DNS PTR record. (Contributed by Leon Weber in issue 20480.)

5.29 json

The `json.tool` command line interface now preserves the order of keys in JSON objects passed in input. The new `--sort-keys` option can be used to sort the keys alphabetically. (Contributed by Berker Peksag in issue 21650.)

JSON decoder now raises `json.JSONDecodeError` instead of `ValueError`. (Contributed by Serhiy Storchaka in issue 19361.)

5.30 locale

A new `delocalize()` function can be used to convert a string into a normalized number string, taking the `LC_NUMERIC` settings into account. (Contributed by Cédric Krier in issue 13918.)

5.31 logging

All logging methods (`Logger log()`, `exception()`, `critical()`, `debug()`, etc.), now accept exception instances as an `exc_info` argument, in addition to boolean values and exception tuples. (Contributed by Yury Selivanov in issue 20537.)

The `handlers.HTTPHandler` class now accepts an optional `ssl.SSLContext` instance to configure SSL settings used in an HTTP connection. (Contributed by Alex Gaynor in issue 22788.)

The `handlers.QueueListener` class now takes a *respect_handler_level* keyword argument which, if set to `True`, will pass messages to handlers taking handler levels into account. (Contributed by Vinay Sajip.)

5.32 lzma

The `LZMADecompressor.decompress` method now accepts an optional *max_length* argument to limit the maximum size of decompressed data. (Contributed by Martin Panter in issue 15955.)

5.33 math

Two new constants have been added to the `math` module: `inf` and `nan`. (Contributed by Mark Dickinson in issue 23185.)

A new function `isclose()` provides a way to test for approximate equality. (Contributed by Chris Barker and Tal Einat in issue 24270.)

A new `gcd()` function has been added. The `fractions.gcd()` function is now deprecated. (Contributed by Mark Dickinson and Serhiy Storchaka in issue 22486.)

5.34 operator

`attrgetter()`, `itemgetter()`, and `methodcaller()` objects now support pickling. (Contributed by Josh Rosenberg and Serhiy Storchaka in issue 22955.)

New `matmul()` and `imatmul()` functions to perform matrix multiplication. (Contributed by Benjamin Peterson in issue 21176.)

5.35 os

The new `scandir()` function returning an iterator of `DirEntry` objects has been added. If possible, `scandir()` extracts file attributes while scanning a directory, removing the need to perform subsequent system calls to determine file type or attributes, which may significantly improve performance. (Contributed by Ben Hoyt with the help of Victor Stinner in issue 22524.)

On Windows, a new `stat_result.st_file_attributes` attribute is now available. It corresponds to `dwFileAttributes` member of the `BY_HANDLE_FILE_INFORMATION` structure returned by `GetFileInformationByHandle()`. (Contributed by Ben Hoyt in issue 21719.)

The `urandom()` function now uses `getrandom()` syscall on Linux 3.17 or newer, and `getentropy()` on OpenBSD 5.6 and newer, removing the need to use `/dev/urandom` and avoiding failures due to potential file descriptor exhaustion. (Contributed by Victor Stinner in issue 22181.)

New `get_blocking()` and `set_blocking()` functions allow to get and set a file descriptor blocking mode (`O_NONBLOCK`.) (Contributed by Victor Stinner in issue 22054.)

The `truncate()` and `ftruncate()` functions are now supported on Windows. (Contributed by Steve Dower in issue 23668.)

There is a new `os.path.commonpath()` function returning the longest common sub-path of each passed pathname. Unlike the `os.path.commonprefix()` function, it always returns a valid path. (Contributed by Rafik Draoui and Serhiy Storchaka in issue 10395.)

5.36 pathlib

The new `Path.samefile` method can be used to check whether the path points to the same file as other path, which can be either an another `Path` object, or a string. (Contributed by Vajrasky Kok and Antoine Pitrou in issue 19775.)

The `Path.mkdir` method how accepts a new optional `exist_ok` argument to match `mkdir -p` and `os.makrdirs()` functionality. (Contributed by Berker Peksag in issue 21539.)

There is a new `Path.expanduser` method to expand ~ and `~user` prefixes. (Contributed by Serhiy Storchaka and Claudiu Popa in issue 19776.)

A new `Path.home` class method can be used to get an instance of `Path` object representing the user's home directory. (Contributed by Victor Salgado and Mayank Tripathi in issue 19777.)

New `Path.write_text`, `Path.read_text`, `Path.write_bytes`, `Path.read_bytes` methods to simplify read/write operations on files. (Contributed by Christopher Welborn in issue 20218.)

5.37 pickle

Nested objects, such as unbound methods or nested classes, can now be pickled using *pickle protocols* older than protocol version 4. Protocol version 4 already supports these cases. (Contributed by Serhiy Storchaka in issue 23611.)

5.38 poplib

A new `POP3.utf8` command enables **RFC 6856** (Internationalized Email) support, if a POP server supports it. (Contributed by Milan OberKirch in issue 21804.)

5.39 re

The number of capturing groups in regular expression is no longer limited by 100. (Contributed by Serhiy Storchaka in issue 22437.)

The `sub()` and `subn()` functions now replace unmatched groups with empty strings instead of raising an exception. (Contributed by Serhiy Storchaka in issue 1519638.)

The `re.error` exceptions have new attributes: `msg`, `pattern`, `pos`, `lineno`, and `colno` that provide better context information about the error. (Contributed by Serhiy Storchaka in issue 22578.)

5.40 readline

A new `append_history_file()` function can be used to append the specified number of trailing elements in history to the given file. (Contributed by Bruno Cauet in issue 22940.)

5.41 selectors

The new `DevpollSelector` supports efficient `/dev/poll` polling on Solaris. (Contributed by Giampaolo Rodola' in issue 18931.)

5.42 shutil

The `move()` function now accepts a *copy_function* argument, allowing, for example, the `copy()` function to be used instead of the default `copy2()` if there is a need to ignore file metadata when moving. (Contributed by Claudiu Popa in issue 19840.)

The `make_archive()` function now supports the *xztar* format. (Contributed by Serhiy Storchaka in issue 5411.)

5.43 signal

On Windows, the `set_wakeup_fd()` function now also supports socket handles. (Contributed by Victor Stinner in issue 22018.)

Various `SIG*` constants in the `signal` module have been converted into `Enums`. This allows meaningful names to be printed during debugging, instead of integer "magic numbers". (Contributed by Giampaolo Rodola' in issue 21076.)

5.44 smtpd

Both `SMTPServer` and `SMTPChannel` classes now accept a *decode_data* keyword argument to determine if the `DATA` portion of the SMTP transaction is decoded using the `"utf-8"` codec or is instead provided to the `SMTPServer.process_message` method as a byte string. The default is `True` for backward compatibility reasons, but will change to `False` in Python 3.6. If *decode_data* is set to `False`, the `process_message()` method must be prepared to accept keyword arguments. (Contributed by Maciej Szulik in issue 19662.)

The `SMTPServer` class now advertises the `8BITMIME` extension (**RFC 6152**) if *decode_data* has been set `True`. If the client specifies `BODY=8BITMIME` on the `MAIL` command, it is passed to `SMTPServer.process_message` via the `mail_options` keyword. (Contributed by Milan Oberkirch and R. David Murray in issue 21795.)

The `SMTPServer` class now also supports the `SMTPUTF8` extension (**RFC 6531**: Internationalized Email). If the client specified `SMTPUTF8 BODY=8BITMIME` on the `MAIL` command, they are passed to `SMTPServer.process_message` via the `mail_options` keyword. It is the responsibility of the `process_message()` method to correctly handle the `SMTPUTF8` data. (Contributed by Milan Oberkirch in issue 21725.)

It is now possible to provide, directly or via name resolution, IPv6 addresses in the `SMTPServer` constructor, and have it successfully connect. (Contributed by Milan Oberkirch in issue 14758.)

5.45 smtplib

A new `SMTP.auth` method provides a convenient way to implement custom authentication mechanisms. (Contributed by Milan Oberkirch in issue 15014.)

The `SMTP.set_debuglevel` method now accepts an additional debuglevel (2), which enables timestamps in debug messages. (Contributed by Gavin Chappell and Maciej Szulik in issue 16914.)

Both `SMTP.sendmail` and `SMTP.send_message` methods now support support **RFC 6531** (SMTPUTF8). (Contributed by Milan Oberkirch and R. David Murray in issue 22027.)

5.46 sndhdr

`what()` and `whathdr()` functions now return a `namedtuple()`. (Contributed by Claudiu Popa in issue 18615.)

5.47 ssl

Memory BIO Support

(Contributed by Geert Jansen in issue 21965.)

The new `SSLObject` class has been added to provide SSL protocol support for cases when the network I/O capabilities of `SSLSocket` are not necessary or suboptimal. `SSLObject` represents an SSL protocol instance, but does not implement any network I/O methods, and instead provides a memory buffer interface. The new `MemoryBIO` class can be used to pass data between Python and an SSL protocol instance.

The memory BIO SSL support is primarily intended to be used in frameworks implementing asynchronous I/O for which `SSLSocket`'s readiness model ("select/poll") is inefficient.

A new `SSLContext.wrap_bio` method can be used to create a new `SSLObject` instance.

Application-Layer Protocol Negotiation Support

(Contributed by Benjamin Peterson in issue 20188.)

Where OpenSSL support is present, `ssl` module now implements *Application-Layer Protocol Negotiation* TLS extension as described in **RFC 7301**.

The new `SSLContext.set_alpn_protocols` can be used to specify which protocols a socket should advertise during the TLS handshake.

The new `SSLSocket.selected_alpn_protocol` returns the protocol that was selected during the TLS handshake. `HAS_ALPN` flag indicates whether APLN support is present.

Other Changes

There is a new `SSLSocket.version` method to query the actual protocol version in use. (Contributed by Antoine Pitrou in issue 20421.)

The `SSLSocket` class now implements a `SSLSocket.sendfile` method. (Contributed by Giampaolo Rodola' in issue 17552.)

The `SSLSocket.send` method now raises either `ssl.SSLWantReadError` or `ssl.SSLWantWriteError` exception on a non-blocking socket if the operation would block. Previously, it would return `0`. (Contributed by Nikolaus Rath in issue 20951.)

The `cert_time_to_seconds()` function now interprets the input time as UTC and not as local time, per **RFC 5280**. Additionally, the return value is always an `int`. (Contributed by Akira Li in issue 19940.)

New `SSLObject.shared_ciphers` and `SSLSocket.shared_ciphers` methods return the list of ciphers sent by the client during the handshake. (Contributed by Benjamin Peterson in issue 23186.)

The `SSLSocket.do_handshake`, `SSLSocket.read`, `SSLSocket.shutdown`, and `SSLSocket.write` methods of `ssl.SSLSocket` class no longer reset the socket timeout every time bytes are received or sent. The socket timeout is now the maximum total duration of the method. (Contributed by Victor Stinner in issue 23853.)

The `match_hostname()` function now supports matching of IP addresses. (Contributed by Antoine Pitrou in issue 23239.)

5.48 socket

Functions with timeouts now use a monotonic clock, instead of a system clock. (Contributed by Victor Stinner in issue 22043.)

A new `socket.sendfile` method allows to send a file over a socket by using the high-performance `os.sendfile()` function on UNIX resulting in uploads being from 2 to 3 times faster than when using plain `socket.send`. (Contributed by Giampaolo Rodola' in issue 17552.)

The `socket.sendall` method no longer resets the socket timeout every time bytes are received or sent. The socket timeout is now the maximum total duration to send all data. (Contributed by Victor Stinner in issue 23853.)

The *backlog* argument of the `socket.listen` method is now optional. By default it is set to `SOMAXCONN` or to `128` whichever is less. (Contributed by Charles-François Natali in issue 21455.)

5.49 sqlite3

The `Row` class now fully supports sequence protocol, in particular `reversed()` iteration and slice indexing. (Contributed by Claudiu Popa in issue 10203; by Lucas Sinclair, Jessica McKellar, and Serhiy Storchaka in issue 13583.)

5.50 subprocess

The new `run()` function has been added. It runs the specified command and and returns a `CompletedProcess` object, which describes a finished process. The new API is more consistent and is the recommended approach to invoking subprocesses in Python code that does not need to maintain compatibility with earlier Python versions. (Contributed by Thomas Kluyver in issue 23342.)

5.51 sys

A new `set_coroutine_wrapper()` function allows setting a global hook that will be called whenever a *coroutine object* is created by an `async def` function. A corresponding `get_coroutine_wrapper()` can be used to obtain a currently set wrapper. Both functions are provisional, and are intended for debugging purposes only. (Contributed by Yury Selivanov in issue 24017.)

A new `is_finalizing()` function can be used to check if the Python interpreter is *shutting down*. (Contributed by Antoine Pitrou in issue 22696.)

5.52 sysconfig

The name of the user scripts directory on Windows now includes the first two components of Python version. (Contributed by Paul Moore in issue 23437.)

5.53 tarfile

The *mode* argument of the `open()` function now accepts `"x"` to request exclusive creation. (Contributed by Berker Peksag in issue 21717.)

`TarFile.extractall` and `TarFile.extract` methods now take a keyword argument *numeric_only*. If set to `True`, the extracted files and directories will be owned by the numeric `uid` and `gid` from the tarfile. If set to `False` (the default, and the behavior in versions prior to 3.5), they will be owned by the named user and group in the tarfile. (Contributed by Michael Vogt and Eric Smith in issue 23193.)

5.54 threading

Both `Lock.acquire` and `RLock.acquire` methods now use a monotonic clock for timeout management. (Contributed by Victor Stinner in issue 22043.)

5.55 time

The `monotonic()` function is now always available. (Contributed by Victor Stinner in issue 22043.)

5.56 timeit

A new command line option `-u` or `--unit=U` can be used to specify the time unit for the timer output. Supported options are `usec`, `msec`, or `sec`. (Contributed by Julian Gindi in issue 18983.)

The `timeit()` function has a new *globals* parameter for specifying the namespace in which the code will be running. (Contributed by Ben Roberts in issue 2527.)

5.57 tkinter

The `tkinter._fix` module used for setting up the Tcl/Tk environment on Windows has been replaced by a private function in the `_tkinter` module which makes no permanent changes to environment variables. (Contributed by Zachary Ware in issue 20035.)

5.58 traceback

New `walk_stack()` and `walk_tb()` functions to conveniently traverse frame and traceback objects. (Contributed by Robert Collins in issue 17911.)

New lightweight classes: `TracebackException`, `StackSummary`, and `traceback.FrameSummary`. (Contributed by Robert Collins in issue 17911.)

Both `print_tb()` and `print_stack()` functions now support negative values for the *limit* argument. (Contributed by Dmitry Kazakov in issue 22619.)

5.59 types

A new `coroutine()` function to transform *generator* and `generator-like` objects into *awaitables*. (Contributed by Yury Selivanov in issue 24017.)

A new `CoroutineType` is the type of *coroutine* objects created by `async def` functions. (Contributed by Yury Selivanov in issue 24400.)

5.60 urllib

A new `request.HTTPPasswordMgrWithPriorAuth` class allows HTTP Basic Authentication credentials to be managed so as to eliminate unnecessary `401` response handling, or to unconditionally send credentials on the first request in order to communicate with servers that return a `404` response instead of a `401` if the `Authorization` header is not sent. (Contributed by Matej Cepl in issue 19494 and Akshit Khurana in issue 7159.)

A new *quote_via* argument for the `parse.urlencode` function provides a way to control the encoding of query parts if needed. (Contributed by Samwyse and Arnon Yaari in issue 13866.)

The `request.urlopen` function accepts an `ssl.SSLContext` object as a *context* argument, which will be used for the HTTPS connection. (Contributed by Alex Gaynor in issue 22366.)

The `parse.urljoin` was updated to use the **RFC 3986** semantics for the resolution of relative URLs, rather than **RFC 1808** and **RFC 2396**. (Contributed by Demian Brecht and Senthil Kumaran in issue 22118.)

5.61 unicodedata

The `unicodedata` module now uses data from Unicode 8.0.0.

5.62 unittest

A new command line option `--locals` to show local variables in tracebacks. (Contributed by Robert Collins in issue 22936.)

5.63 unittest.mock

The `Mock` has the following improvements:

- Class constructor has a new *unsafe* parameter, which causes mock objects to raise `AttributeError` on attribute names starting with `"assert"`. (Contributed by Kushal Das in issue 21238.)
- A new `Mock.assert_not_called` method to check if the mock object was called. (Contributed by Kushal Das in issue 21262.)

The `MagicMock` class now supports `__truediv__()`, `__divmod__()` and `__matmul__()` operators. (Contributed by Johannes Baiter in issue 20968, and Håkan Lövdahl in issue 23581 and issue 23568.)

5.64 wsgiref

The *headers* argument of the `headers.Headers` class constructor is now optional. (Contributed by Pablo Torres Navarrete and SilentGhost in issue 5800.)

5.65 xmlrpc

The `client.ServerProxy` class is now a *context manager*. (Contributed by Claudiu Popa in issue 20627.)

`client.ServerProxy` constructor now accepts an optional `ssl.SSLContext` instance. (Contributed by Alex Gaynor in issue 22960.)

5.66 xml.sax

SAX parsers now support a character stream of the `xmlreader.InputSource` object. (Contributed by Serhiy Storchaka in issue 2175.)

5.67 zipfile

ZIP output can now be written to unseekable streams. (Contributed by Serhiy Storchaka in issue 23252.)

The *mode* argument of `ZipFile.open` method now accepts `"x"` to request exclusive creation. (Contributed by Serhiy Storchaka in issue 21717.)

6 Other module-level changes

Many functions in `mmap`, `ossaudiodev`, `socket`, `ssl`, and `codecs` modules now accept writable *bytes-like objects*. (Contributed by Serhiy Storchaka in issue 23001.)

7 Optimizations

The `os.walk()` function has been sped up by 3 to 5 times on POSIX systems, and by 7 to 20 times on Windows. This was done using the new `os.scandir()` function, which exposes file information from the underlying `readdir` or `FindFirstFile`/`FindNextFile` system calls. (Contributed by Ben Hoyt with help from Victor Stinner in issue 23605.)

Construction of `bytes(int)` (filled by zero bytes) is faster and uses less memory for large objects. `calloc()` is used instead of `malloc()` to allocate memory for these objects. (Contributed by Victor Stinner in issue 21233.)

Some operations on `ipaddress` `IPv4Network` and `IPv6Network` have been massively sped up, such as `subnets()`, `supernet()`, `summarize_address_range()`, `collapse_addresses()`. The speed up can range from 3 to 15 times. (Contributed by Antoine Pitrou, Michel Albert, and Markus in issue 21486, issue 21487, issue 20826, issue 23266.)

Pickling of `ipaddress` objects was optimized to produce significantly smaller output. (Contributed by Serhiy Storchaka in issue 23133.)

Many operations on `io.BytesIO` are now 50% to 100% faster. (Contributed by Serhiy Storchaka in issue 15381 and David Wilson in issue 22003.)

The `marshal.dumps()` function is now faster: 65-85% with versions 3 and 4, 20-25% with versions 0 to 2 on typical data, and up to 5 times in best cases. (Contributed by Serhiy Storchaka in issue 20416 and issue 23344.)

The UTF-32 encoder is now 3 to 7 times faster. (Contributed by Serhiy Storchaka in issue 15027.)

Regular expressions are now parsed up to 10% faster. (Contributed by Serhiy Storchaka in issue 19380.)

The `json.dumps()` function was optimized to run with `ensure_ascii=False` as fast as with `ensure_ascii=True`. (Contributed by Naoki Inada in issue 23206.)

The `PyObject_IsInstance()` and `PyObject_IsSubclass()` functions have been sped up in the common case that the second argument has `type` as its metaclass. (Contributed Georg Brandl by in issue 22540.)

Method caching was slightly improved, yielding up to 5% performance improvement in some benchmarks. (Contributed by Antoine Pitrou in issue 22847.)

Objects from `random` module now use two times less memory on 64-bit builds. (Contributed by Serhiy Storchaka in issue 23488.)

The `property()` getter calls are up to 25% faster. (Contributed by Joe Jevnik in issue 23910.)

Instantiation of `fractions.Fraction` is now up to 30% faster. (Contributed by Stefan Behnel in issue 22464.)

String methods `find()`, `rfind()`, `split()`, `partition()` and `in` string operator are now significantly faster for searching 1-character substrings. (Contributed by Serhiy Storchaka in issue 23573.)

8 Build and C API Changes

New `calloc` functions were added:

- `PyMem_RawCalloc()`,

- `PyMem_Calloc()`,
- `PyObject_Calloc()`,
- `_PyObject_GC_Calloc()`.

(Contributed by Victor Stinner in issue 21233.)

New encoding/decoding helper functions:

- `Py_DecodeLocale()` (replaced `_Py_char2wchar()`),
- `Py_EncodeLocale()` (replaced `_Py_wchar2char()`).

(Contributed by Victor Stinner in issue 18395.)

A new `PyCodec_NameReplaceErrors()` function to replace the unicode encode error with `\N{...}` escapes. (Contributed by Serhiy Storchaka in issue 19676.)

A new `PyErr_FormatV()` function similar to `PyErr_Format()`, but accepts a `va_list` argument. (Contributed by Antoine Pitrou in issue 18711.)

A new `PyExc_RecursionError` exception. (Contributed by Georg Brandl in issue 19235.)

New `PyModule_FromDefAndSpec()`, `PyModule_FromDefAndSpec2()`, and `PyModule_ExecDef()` introduced by **PEP 489** – multi-phase extension module initialization. (Contributed by Petr Viktorin in issue 24268.)

New `PyNumber_MatrixMultiply()` and `PyNumber_InPlaceMatrixMultiply()` functions to perform matrix multiplication. (Contributed by Benjamin Peterson in issue 21176. See also **PEP 465** for details.)

The `PyTypeObject.tp_finalize` slot is now part of stable ABI.

Windows builds now require Microsoft Visual C++ 14.0, which is available as part of Visual Studio 2015.

Extension modules now include platform information tag in their filename on some platforms (the tag is optional, and CPython will import extensions without it; although if the tag is present and mismatched, the extension won't be loaded):

- On Linux, extension module filenames end with `.cpython-<major><minor>m-<architecture>-<os>.pyd`:
  - `<major>` is the major number of the Python version; for Python 3.5 this is `3`.
  - `<minor>` is the minor number of the Python version; for Python 3.5 this is `5`.
  - `<architecture>` is the hardware architecture the extension module was built to run on. It's most commonly either `i386` for 32-bit Intel platforms or `x86_64` for 64-bit Intel (and AMD) platforms.
  - `<os>` is always `linux-gnu`, except for extensions built to talk to the 32-bit ABI on 64-bit platforms, in which case it is `linux-gnu32` (and `<architecture>` will be `x86_64`).
- On Windows, extension module filenames end with `<debug>.cp<major><minor>-<platform>.pyd`:
  - `<major>` is the major number of the Python version; for Python 3.5 this is `3`.
  - `<minor>` is the minor number of the Python version; for Python 3.5 this is `5`.
  - `<platform>` is the platform the extension module was built for, either `win32` for Win32, `win_amd64` for Win64, `win_ia64` for Windows Itanium 64, and `win_arm` for Windows on ARM.
  - If built in debug mode, `<debug>` will be `_d`, otherwise it will be blank.
- On OS X platforms, extension module filenames now end with `-darwin.so`.
- On all other platforms, extension module filenames are the same as they were with Python 3.4.

9 Deprecated

9.1 New Keywords

`async` and `await` are not recommended to be used as variable, class, function or module names. Introduced by **PEP 492** in Python 3.5, they will become proper keywords in Python 3.7.

9.2 Unsupported Operating Systems

Windows XP is no longer supported by Microsoft, thus, per **PEP 11**, CPython 3.5 is no longer officially supported on this OS.

9.3 Deprecated Python modules, functions and methods

The `formatter` module has now graduated to full deprecation and is still slated for removal in Python 3.6.

The `asyncio.async()` function is deprecated in favor of `ensure_future()`.

The `smtpd` module has in the past always decoded the DATA portion of email messages using the `utf-8` codec. This can now be controlled by the new *decode_data* keyword to `SMTPServer`. The default value is `True`, but this default is deprecated. Specify the *decode_data* keyword with an appropriate value to avoid the deprecation warning.

Directly assigning values to the `key`, `value` and `coded_value` of `Morsel` objects is deprecated. Use the `set()` method instead. In addition, the undocumented *LegalChars* parameter of `set()` is deprecated, and is now ignored.

Passing a format string as keyword argument *format_string* to the `format()` method of the `string.Formatter` class has been deprecated.

The `platform.dist()` and `platform.linux_distribution()` functions are now deprecated and will be removed in Python 3.7. Linux distributions use too many different ways of describing themselves, so the functionality is left to a package. (Contributed by Vajrasky Kok and Berker Peksag in issue 1322.)

The previously undocumented `from_function` and `from_builtin` methods of `inspect.Signature` are deprecated. Use new `inspect.Signature.from_callable()` instead. (Contributed by Yury Selivanov in issue 24248.)

The `inspect.getargspec()` function is deprecated and scheduled to be removed in Python 3.6. (See issue 20438 for details.)

The `inspect getfullargspec()`, `getargvalues()`, `getcallargs()`, `getargvalues()`, `formatargspec()`, and `formatargvalues()` functions are deprecated in favor of `inspect.signature()` API. (Contributed by Yury Selivanov in issue 20438.)

Use of `re.LOCALE` flag with str patterns or `re.ASCII` is now deprecated. (Contributed by Serhiy Storchaka in issue 22407.)

10 Removed

10.1 API and Feature Removals

The following obsolete and previously deprecated APIs and features have been removed:

- The `__version__` attribute has been dropped from the email package. The email code hasn't been shipped separately from the stdlib for a long time, and the `__version__` string was not updated in the last few releases.

- The internal `Netrc` class in the `ftplib` module was deprecated in 3.4, and has now been removed. (Contributed by Matt Chaput in issue 6623.)
- The concept of `.pyo` files has been removed.
- The JoinableQueue class in the provisional asyncio module was deprecated in 3.4.4 and is now removed. (Contributed by A. Jesse Jiryu Davis in issue 23464.)

11 Porting to Python 3.5

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

11.1 Changes in the Python API

- **PEP 475**: System calls are now retried when interrupted by a signal instead of raising `InterruptedError` if the Python signal handler does not raise an exception.
- Before Python 3.5, a `datetime.time` object was considered to be false if it represented midnight in UTC. This behavior was considered obscure and error-prone and has been removed in Python 3.5. See issue 13936 for full details.
- The `ssl.SSLSocket.send()` method now raises either `ssl.SSLWantReadError` or `ssl.SSLWantWriteError` on a non-blocking socket if the operation would block. Previously, it would return `0`. (Contributed by Nikolaus Rath in issue 20951.)
- The `__name__` attribute of generator is now set from the function name, instead of being set from the code name. Use `gen.gi_code.co_name` to retrieve the code name. Generators also have a new `__qualname__` attribute, the qualified name, which is now used for the representation of a generator (`repr(gen)`). (Contributed by Victor Stinner in issue 21205.)
- The deprecated "strict" mode and argument of `HTMLParser`, `HTMLParser.error()`, and the `HTMLParserError` exception have been removed. (Contributed by Ezio Melotti in issue 15114.) The *convert_charrefs* argument of `HTMLParser` is now `True` by default. (Contributed by Berker Peksag in issue 21047.)
- Although it is not formally part of the API, it is worth noting for porting purposes (ie: fixing tests) that error messages that were previously of the form "'sometype' does not support the buffer protocol" are now of the form "a *bytes-like object* is required, not 'sometype'". (Contributed by Ezio Melotti in issue 16518.)
- If the current directory is set to a directory that no longer exists then `FileNotFoundError` will no longer be raised and instead `find_spec()` will return `None` **without** caching `None` in `sys.path_importer_cache` which is different than the typical case (issue 22834).
- HTTP status code and messages from `http.client` and `http.server` were refactored into a common `HTTPStatus` enum. The values in `http.client` and `http.server` remain available for backwards compatibility. (Contributed by Demian Brecht in issue 21793.)
- When an import loader defines `exec_module()` it is now expected to also define `create_module()` (raises a `DeprecationWarning` now, will be an error in Python 3.6). If the loader inherits from `importlib.abc.Loader` then there is nothing to do, else simply define `create_module()` to return `None`. (Contributed by Brett Cannon in issue 23014.)
- The `re.split()` function always ignored empty pattern matches, so the `"x*"` pattern worked the same as `"x+"`, and the `"\b"` pattern never worked. Now `re.split()` raises a warning if the pattern could match an empty string. For compatibility use patterns that never match an empty string (e.g. `"x+"` instead of `"x*"`). Patterns that could only match an empty string (such as `"\b"`) now raise an error.

- The `Morsel` dict-like interface has been made self consistent: morsel comparison now takes the `key` and `value` into account, `copy()` now results in a `Morsel` instance rather than a `dict`, and `update()` will now raise an exception if any of the keys in the update dictionary are invalid. In addition, the undocumented *LegalChars* parameter of `set()` is deprecated and is now ignored. (Contributed by Demian Brecht in issue 2211.)
- **PEP 488** has removed `.pyo` files from Python and introduced the optional `opt-` tag in `.pyc` file names. The `importlib.util.cache_from_source()` has gained an *optimization* parameter to help control the `opt-` tag. Because of this, the *debug_override* parameter of the function is now deprecated. *.pyo* files are also no longer supported as a file argument to the Python interpreter and thus serve no purpose when distributed on their own (i.e. sourceless code distribution). Due to the fact that the magic number for bytecode has changed in Python 3.5, all old *.pyo* files from previous versions of Python are invalid regardless of this PEP.
- The `socket` module now exports the `CAN_RAW_FD_FRAMES` constant on linux 3.6 and greater.
- The `cert_time_to_seconds()` function now interprets the input time as UTC and not as local time, per **RFC 5280**. Additionally, the return value is always an `int`. (Contributed by Akira Li in issue 19940.)
- The `pygettext.py` Tool now uses the standard +NNNN format for timezones in the POT-Creation-Date header.
- The `smtplib` module now uses `sys.stderr` instead of previous module level `stderr` variable for debug output. If your (test) program depends on patching the module level variable to capture the debug output, you will need to update it to capture sys.stderr instead.
- The `str.startswith()` and `str.endswith()` methods no longer return `True` when finding the empty string and the indexes are completely out of range. (Contributed by Serhiy Storchaka in issue 24284.)
- The `inspect.getdoc()` function now returns documentation strings inherited from base classes. Documentation strings no longer need to be duplicated if the inherited documentation is appropriate. To suppress an inherited string, an empty string must be specified (or the documentation may be filled in). This change affects the output of the `pydoc` module and the `help()` function. (Contributed by Serhiy Storchaka in issue 15582.)

11.2 Changes in the C API

- The undocumented `format` member of the (non-public) `PyMemoryViewObject` structure has been removed. All extensions relying on the relevant parts in `memoryobject.h` must be rebuilt.
- The `PyMemAllocator` structure was renamed to `PyMemAllocatorEx` and a new `calloc` field was added.
- Removed non-documented macro `PyObject_REPR` which leaked references. Use format character `%R` in `PyUnicode_FromFormat()`-like functions to format the `repr()` of the object. (Contributed by Serhiy Storchaka in issue 22453.)
- Because the lack of the `__module__` attribute breaks pickling and introspection, a deprecation warning now is raised for builtin type without the `__module__` attribute. Would be an AttributeError in future. (Contributed by Serhiy Storchaka in issue 20204.)
- As part of **PEP 492** implementation, `tp_reserved` slot of `PyTypeObject` was replaced with a `tp_as_async` slot. Refer to *coro-objects* for new types, structures and functions.

Index

C

COLS, 12

E

environment variable
- COLS, 12
- LINES, 12

L

LINES, 12

P

Python Enhancement Proposals
- PEP 11, 25
- PEP 397, 9
- PEP 441, 3, 10, 11
- PEP 448, 3, 7
- PEP 451, 10
- PEP 461, 3, 8
- PEP 465, 3, 6, 7, 24
- PEP 471, 8
- PEP 475, 8, 9, 26
- PEP 478, 3
- PEP 479, 9
- PEP 484, 8
- PEP 485, 10
- PEP 486, 9
- PEP 488, 4, 9, 27
- PEP 489, 4, 10, 24
- PEP 492, 3–6, 25, 27

R

RFC
- RFC 1808, 22
- RFC 2396, 22
- RFC 3986, 22
- RFC 5161, 14
- RFC 5280, 19, 27
- RFC 6152, 18
- RFC 6531, 13, 18
- RFC 6532, 13
- RFC 6855, 14
- RFC 6856, 17
- RFC 7301, 19