

---

# What's New in Python

*Release 3.6.0rc1*

**A. M. Kuchling**

**December 06, 2016**

**Python Software Foundation**  
**Email: [docs@python.org](mailto:docs@python.org)**

## Contents

<b>1</b>	<b>Summary – Release highlights</b>	<b>3</b>
<b>2</b>	<b>New Features</b>	<b>5</b>
2.1	PEP 498: Formatted string literals . . . . .	5
2.2	PEP 526: Syntax for variable annotations . . . . .	5
2.3	PEP 515: Underscores in Numeric Literals . . . . .	5
2.4	PEP 525: Asynchronous Generators . . . . .	6
2.5	PEP 530: Asynchronous Comprehensions . . . . .	6
2.6	PEP 487: Simpler customization of class creation . . . . .	6
2.7	PEP 487: Descriptor Protocol Enhancements . . . . .	7
2.8	PEP 519: Adding a file system path protocol . . . . .	7
2.9	PEP 495: Local Time Disambiguation . . . . .	8
2.10	PEP 529: Change Windows filesystem encoding to UTF-8 . . . . .	8
2.11	PEP 528: Change Windows console encoding to UTF-8 . . . . .	9
2.12	PEP 520: Preserving Class Attribute Definition Order . . . . .	9
2.13	PEP 468: Preserving Keyword Argument Order . . . . .	9
2.14	New dict implementation . . . . .	9
2.15	PEP 523: Adding a frame evaluation API to CPython . . . . .	10
2.16	PYTHONMALLOC environment variable . . . . .	10
2.17	DTrace and SystemTap probing support . . . . .	11
<b>3</b>	<b>Other Language Changes</b>	<b>11</b>
<b>4</b>	<b>New Modules</b>	<b>12</b>
4.1	secrets . . . . .	12
<b>5</b>	<b>Improved Modules</b>	<b>12</b>
5.1	array . . . . .	12
5.2	ast . . . . .	12
5.3	asyncio . . . . .	12
5.4	binascii . . . . .	13
5.5	cmath . . . . .	13
5.6	collections . . . . .	13
5.7	concurrent.futures . . . . .	14
5.8	contextlib . . . . .	14
5.9	datetime . . . . .	14
5.10	decimal . . . . .	14
5.11	distutils . . . . .	14

5.12	email	14
5.13	encodings	15
5.14	enum	15
5.15	faulthandler	15
5.16	fileinput	15
5.17	hashlib	15
5.18	http.client	16
5.19	idlelib and IDLE	16
5.20	importlib	16
5.21	inspect	16
5.22	json	16
5.23	logging	16
5.24	math	17
5.25	multiprocessing	17
5.26	os	17
5.27	pathlib	17
5.28	pdb	17
5.29	pickle	17
5.30	pickletools	17
5.31	pydoc	17
5.32	random	18
5.33	re	18
5.34	readline	18
5.35	rlcompleter	18
5.36	shlex	18
5.37	site	18
5.38	sqlite3	18
5.39	socket	18
5.40	socketserver	19
5.41	ssl	19
5.42	statistics	19
5.43	struct	19
5.44	subprocess	19
5.45	sys	20
5.46	telnetlib	20
5.47	time	20
5.48	timeit	20
5.49	tkinter	20
5.50	traceback	20
5.51	tracemalloc	20
5.52	typing	21
5.53	unicodedata	21
5.54	unittest.mock	21
5.55	urllib.request	21
5.56	urllib.robotparser	21
5.57	venv	22
5.58	warnings	22
5.59	winreg	22
5.60	winsound	22
5.61	xmlrpc.client	22
5.62	zipfile	22
5.63	zlib	23
<b>6</b>	<b>Optimizations</b>	<b>23</b>
<b>7</b>	<b>Build and C API Changes</b>	<b>24</b>
<b>8</b>	<b>Other Improvements</b>	<b>24</b>

<b>9</b>	<b>Deprecated</b>	<b>24</b>
9.1	New Keywords	24
9.2	Deprecated Python behavior	24
9.3	Deprecated Python modules, functions and methods	25
	asynchat	25
	asyncore	25
	dbm	25
	distutils	25
	grp	25
	importlib	25
	os	25
	re	26
	ssl	26
	tkinter	26
	venv	26
9.4	Deprecated functions and types of the C API	26
9.5	Deprecated Build Options	26
<b>10</b>	<b>Removed</b>	<b>26</b>
10.1	API and Feature Removals	26
<b>11</b>	<b>Porting to Python 3.6</b>	<b>27</b>
11.1	Changes in ‘python’ Command Behavior	27
11.2	Changes in the Python API	27
11.3	Changes in the C API	29
11.4	CPython bytecode changes	29
	<b>Index</b>	<b>31</b>

---

**Release** 3.6.0rc1

**Date** December 06, 2016

**Editors** Elvis Pranskevichus <elvis@magic.io>, Yury Selivanov <yury@magic.io>

---

**Note:** Prerelease users should be aware that this document is currently in draft form. It will be updated substantially as Python 3.6 moves towards release, so it’s worth checking back even after reading earlier versions.

---

This article explains the new features in Python 3.6, compared to 3.5.

**See also:**

**PEP 494** - Python 3.6 Release Schedule

## 1 Summary – Release highlights

New syntax features:

- *PEP 498*, formatted string literals.
- *PEP 515*, underscores in numeric literals.
- *PEP 526*, syntax for variable annotations.
- *PEP 525*, asynchronous generators.
- *PEP 530*: asynchronous comprehensions.

New library modules:

- `secrets`: [PEP 506 – Adding A Secrets Module To The Standard Library](#).

CPython implementation improvements:

- The dict type has been reimplemented to use a *faster, more compact representation* similar to the [PyPy dict implementation](#). This resulted in dictionaries using 20% to 25% less memory when compared to Python 3.5.
- Customization of class creation has been simplified with the *new protocol*.
- The class attribute definition order is *now preserved*.
- The order of elements in `**kwargs` now *corresponds to the order* in which keyword arguments were passed to the function.
- DTrace and SystemTap *probing support* has been added.
- The new [PYTHONMALLOC](#) environment variable can now be used to debug the interpreter memory allocation and access errors.

Significant improvements in the standard library:

- The `asyncio` module has received new features, significant usability and performance improvements, and a fair amount of bug fixes. Starting with Python 3.6 the `asyncio` module is no longer provisional and its API is considered stable.
- A new *file system path protocol* has been implemented to support path-like objects. All standard library functions operating on paths have been updated to work with the new protocol.
- The `datetime` module has gained support for *Local Time Disambiguation*.
- The `typing` module received a number of *improvements* and is no longer provisional.
- The `tracemalloc` module has been significantly reworked and is now used to provide better output for `ResourceWarning` as well as provide better diagnostics for memory allocation errors. See the [PYTHONMALLOC section](#) for more information.

Security improvements:

- The new `secrets` module has been added to simplify the generation of cryptographically strong pseudo-random numbers suitable for managing secrets such as account authentication, tokens, and similar.
- On Linux, `os.urandom()` now blocks until the system urandom entropy pool is initialized to increase the security. See the [PEP 524](#) for the rationale.
- The `hashlib` and `ssl` modules now support OpenSSL 1.1.0.
- The default settings and feature set of the `ssl` module have been improved.
- The `hashlib` module received support for the BLAKE2, SHA-3 and SHAKE hash algorithms and the `script()` key derivation function.

Windows improvements:

- [PEP 528](#) and [PEP 529](#), Windows filesystem and console encoding changed to UTF-8.
- The `py.exe` launcher, when used interactively, no longer prefers Python 2 over Python 3 when the user doesn't specify a version (via command line arguments or a config file). Handling of shebang lines remains unchanged - "python" refers to Python 2 in that case.
- `python.exe` and `pythonw.exe` have been marked as long-path aware, which means that the 260 character path limit may no longer apply. See removing the MAX\_PATH limitation for details.
- A `._pth` file can be added to force isolated mode and fully specify all search paths to avoid registry and environment lookup. See the documentation for more information.
- A `python36.zip` file now works as a landmark to infer `PYTHONHOME`. See the documentation for more information.

## 2 New Features

### 2.1 PEP 498: Formatted string literals

**PEP 498** introduces a new kind of string literals: *f-strings*, or formatted string literals.

Formatted string literals are prefixed with 'f' and are similar to the format strings accepted by `str.format()`. They contain replacement fields surrounded by curly braces. The replacement fields are expressions, which are evaluated at run time, and then formatted using the `format()` protocol:

```
>>> name = "Fred"
>>> f"He said his name is {name}."
'He said his name is Fred.'
>>> width = 10
>>> precision = 4
>>> value = decimal.Decimal("12.34567")
>>> f"result: {value:{width}.{precision}}" # nested fields
'result:      12.35'
```

**See also:**

**PEP 498 – Literal String Interpolation.** PEP written and implemented by Eric V. Smith.

Feature documentation.

### 2.2 PEP 526: Syntax for variable annotations

**PEP 484** introduced the standard for type annotations of function parameters, a.k.a. type hints. This PEP adds syntax to Python for annotating the types of variables including class variables and instance variables:

```
primes: List[int] = []

captain: str # Note: no initial value!

class Starship:
    stats: Dict[str, int] = {}
```

Just as for function annotations, the Python interpreter does not attach any particular meaning to variable annotations and only stores them in the `__annotations__` attribute of a class or module.

In contrast to variable declarations in statically typed languages, the goal of annotation syntax is to provide an easy way to specify structured type metadata for third party tools and libraries via the abstract syntax tree and the `__annotations__` attribute.

**See also:**

**PEP 526 – Syntax for variable annotations.** PEP written by Ryan Gonzalez, Philip House, Ivan Levkivskyi, Lisa Roach, and Guido van Rossum. Implemented by Ivan Levkivskyi.

Tools that use or will use the new syntax: [mypy](#), [pytype](#), [PyCharm](#), etc.

### 2.3 PEP 515: Underscores in Numeric Literals

**PEP 515** adds the ability to use underscores in numeric literals for improved readability. For example:

```
>>> 1_000_000_000_000_000
1000000000000000
>>> 0xFF_FF_FF_FF
4294967295
```

Single underscores are allowed between digits and after any base specifier. Leading, trailing, or multiple underscores in a row are not allowed.

The string formatting language also now has support for the `'_'` option to signal the use of an underscore for a thousands separator for floating point presentation types and for integer presentation type `'d'`. For integer presentation types `'b'`, `'o'`, `'x'`, and `'X'`, underscores will be inserted every 4 digits:

```
>>> '{:_}'.format(1000000)
'1_000_000'
>>> '{:_x}'.format(0xFFFFFFFF)
'ffff_ffff'
```

See also:

**PEP 515 – Underscores in Numeric Literals** PEP written by Georg Brandl and Serhiy Storchaka.

## 2.4 PEP 525: Asynchronous Generators

**PEP 492** introduced support for native coroutines and `async / await` syntax to Python 3.5. A notable limitation of the Python 3.5 implementation is that it was not possible to use `await` and `yield` in the same function body. In Python 3.6 this restriction has been lifted, making it possible to define *asynchronous generators*:

```
async def ticker(delay, to):
    """Yield numbers from 0 to *to* every *delay* seconds."""
    for i in range(to):
        yield i
        await asyncio.sleep(delay)
```

The new syntax allows for faster and more concise code.

See also:

**PEP 525 – Asynchronous Generators** PEP written and implemented by Yuri Selivanov.

## 2.5 PEP 530: Asynchronous Comprehensions

**PEP 530** adds support for using `async for` in list, set, dict comprehensions and generator expressions:

```
result = [i async for i in aiter() if i % 2]
```

Additionally, `await` expressions are supported in all kinds of comprehensions:

```
result = [await fun() for fun in funcs if await condition()]
```

See also:

**PEP 530 – Asynchronous Comprehensions** PEP written and implemented by Yuri Selivanov.

## 2.6 PEP 487: Simpler customization of class creation

It is now possible to customize subclass creation without using a metaclass. The new `__init_subclass__` classmethod will be called on the base class whenever a new subclass is created:

```
class PluginBase:
    subclasses = []

    def __init_subclass__(cls, **kwargs):
        super().__init_subclass__(**kwargs)
        cls.subclasses.append(cls)

class Plugin1(PluginBase):
    pass
```

```
class Plugin2(PluginBase):
    pass
```

In order to allow zero-argument `super()` calls to work correctly from `__init_subclass__()` implementations, custom metaclasses must ensure that the new `__classcell__` namespace entry is propagated to `type.__new__` (as described in [class-object-creation](#)).

See also:

**PEP 487 – Simpler customization of class creation** PEP written and implemented by Martin Teichmann.

[Feature documentation](#)

## 2.7 PEP 487: Descriptor Protocol Enhancements

**PEP 487** extends the descriptor protocol has to include the new optional `__set_name__()` method. Whenever a new class is defined, the new method will be called on all descriptors included in the definition, providing them with a reference to the class being defined and the name given to the descriptor within the class namespace. In other words, instances of descriptors can now know the attribute name of the descriptor in the owner class:

```
class IntField:
    def __get__(self, instance, owner):
        return instance.__dict__[self.name]

    def __set__(self, instance, value):
        if not isinstance(value, int):
            raise ValueError(f'expecting integer in {self.name}')
        instance.__dict__[self.name] = value

    # this is the new initializer:
    def __set_name__(self, owner, name):
        self.name = name

class Model:
    int_field = IntField()
```

See also:

**PEP 487 – Simpler customization of class creation** PEP written and implemented by Martin Teichmann.

[Feature documentation](#)

## 2.8 PEP 519: Adding a file system path protocol

File system paths have historically been represented as `str` or `bytes` objects. This has led to people who write code which operate on file system paths to assume that such objects are only one of those two types (an `int` representing a file descriptor does not count as that is not a file path). Unfortunately that assumption prevents alternative object representations of file system paths like `pathlib` from working with pre-existing code, including Python's standard library.

To fix this situation, a new interface represented by `os.PathLike` has been defined. By implementing the `__fspath__()` method, an object signals that it represents a path. An object can then provide a low-level representation of a file system path as a `str` or `bytes` object. This means an object is considered path-like if it implements `os.PathLike` or is a `str` or `bytes` object which represents a file system path. Code can use `os.fspath()`, `os.fsdecode()`, or `os.fsencode()` to explicitly get a `str` and/or `bytes` representation of a path-like object.

The built-in `open()` function has been updated to accept `os.PathLike` objects, as have all relevant functions in the `os` and `os.path` modules, and most other functions and classes in the standard library. The `os.DirEntry` class and relevant classes in `pathlib` have also been updated to implement `os.PathLike`.

The hope is that updating the fundamental functions for operating on file system paths will lead to third-party code to implicitly support all path-like objects without any code changes, or at least very minimal ones (e.g. calling `os.fspath()` at the beginning of code before operating on a path-like object).

Here are some examples of how the new interface allows for `pathlib.Path` to be used more easily and transparently with pre-existing code:

```
>>> import pathlib
>>> with open(pathlib.Path("README")) as f:
...     contents = f.read()
...
>>> import os.path
>>> os.path.splitext(pathlib.Path("some_file.txt"))
('some_file', '.txt')
>>> os.path.join("/a/b", pathlib.Path("c"))
'/a/b/c'
>>> import os
>>> os.fspath(pathlib.Path("some_file.txt"))
'some_file.txt'
```

(Implemented by Brett Cannon, Ethan Furman, Dusty Phillips, and Jelle Zijlstra.)

**See also:**

**PEP 519 – Adding a file system path protocol** PEP written by Brett Cannon and Koos Zevenhoven.

## 2.9 PEP 495: Local Time Disambiguation

In most world locations, there have been and will be times when local clocks are moved back. In those times, intervals are introduced in which local clocks show the same time twice in the same day. In these situations, the information displayed on a local clock (or stored in a Python `datetime` instance) is insufficient to identify a particular moment in time.

**PEP 495** adds the new *fold* attribute to instances of `datetime.datetime` and `datetime.time` classes to differentiate between two moments in time for which local times are the same:

```
>>> u0 = datetime(2016, 11, 6, 4, tzinfo=timezone.utc)
>>> for i in range(4):
...     u = u0 + i*HOUR
...     t = u.astimezone(Eastern)
...     print(u.time(), 'UTC =', t.time(), t.tzname(), t.fold)
...
04:00:00 UTC = 00:00:00 EDT 0
05:00:00 UTC = 01:00:00 EDT 0
06:00:00 UTC = 01:00:00 EST 1
07:00:00 UTC = 02:00:00 EST 0
```

The values of the `fold` attribute have the value 0 for all instances except those that represent the second (chronologically) moment in time in an ambiguous case.

**See also:**

**PEP 495 – Local Time Disambiguation** PEP written by Alexander Belopolsky and Tim Peters, implementation by Alexander Belopolsky.

## 2.10 PEP 529: Change Windows filesystem encoding to UTF-8

Representing filesystem paths is best performed with `str` (Unicode) rather than `bytes`. However, there are some situations where using `bytes` is sufficient and correct.



Prior to Python 3.6, data loss could result when using bytes paths on Windows. With this change, using bytes to represent paths is now supported on Windows, provided those bytes are encoded with the encoding returned by `sys.getfilesystemencoding()`, which now defaults to `'utf-8'`.

Applications that do not use `str` to represent paths should use `os.fsencode()` and `os.fsdecode()` to ensure their bytes are correctly encoded. To revert to the previous behaviour, set `PYTHONLEGACYWINDOWSFSENCODING` or call `sys._enablelegacywindowsfsencoding()`.

See [PEP 529](#) for more information and discussion of code modifications that may be required.

## 2.11 PEP 528: Change Windows console encoding to UTF-8

The default console on Windows will now accept all Unicode characters and provide correctly read `str` objects to Python code. `sys.stdin`, `sys.stdout` and `sys.stderr` now default to utf-8 encoding.

This change only applies when using an interactive console, and not when redirecting files or pipes. To revert to the previous behaviour for interactive console use, set `PYTHONLEGACYWINDOWSTOENCODING`.

See also:

**PEP 528 – Change Windows console encoding to UTF-8** PEP written and implemented by Steve Dower.

## 2.12 PEP 520: Preserving Class Attribute Definition Order

Attributes in a class definition body have a natural ordering: the same order in which the names appear in the source. This order is now preserved in the new class's `__dict__` attribute.

Also, the effective default class *execution* namespace (returned from `type.__prepare__()`) is now an insertion-order-preserving mapping.

See also:

**PEP 520 – Preserving Class Attribute Definition Order** PEP written and implemented by Eric Snow.

## 2.13 PEP 468: Preserving Keyword Argument Order

`**kwargs` in a function signature is now guaranteed to be an insertion-order-preserving mapping.

See also:

**PEP 468 – Preserving Keyword Argument Order** PEP written and implemented by Eric Snow.

## 2.14 New dict implementation

The dict type now uses a “compact” representation pioneered by PyPy. The memory usage of the new `dict()` is between 20% and 25% smaller compared to Python 3.5.

The order-preserving aspect of this new implementation is considered an implementation detail and should not be relied upon (this may change in the future, but it is desired to have this new dict implementation in the language for a few releases before changing the language spec to mandate order-preserving semantics for all current and future Python implementations; this also helps preserve backwards-compatibility with older versions of the language where random iteration order is still in effect, e.g. Python 3.5).

(Contributed by INADA Naoki in [issue 27350](#). Idea originally suggested by Raymond Hettinger.)

## 2.15 PEP 523: Adding a frame evaluation API to CPython

While Python provides extensive support to customize how code executes, one place it has not done so is in the evaluation of frame objects. If you wanted some way to intercept frame evaluation in Python there really wasn't any way without directly manipulating function pointers for defined functions.

**PEP 523** changes this by providing an API to make frame evaluation pluggable at the C level. This will allow for tools such as debuggers and JITs to intercept frame evaluation before the execution of Python code begins. This enables the use of alternative evaluation implementations for Python code, tracking frame evaluation, etc.

This API is not part of the limited C API and is marked as private to signal that usage of this API is expected to be limited and only applicable to very select, low-level use-cases. Semantics of the API will change with Python as necessary.

**See also:**

**PEP 523 – Adding a frame evaluation API to CPython** PEP written by Brett Cannon and Dino Viehland.

## 2.16 PYTHONMALLOC environment variable

The new `PYTHONMALLOC` environment variable allows setting the Python memory allocators and installing debug hooks.

It is now possible to install debug hooks on Python memory allocators on Python compiled in release mode using `PYTHONMALLOC=debug`. Effects of debug hooks:

- Newly allocated memory is filled with the byte `0xCB`
- Freed memory is filled with the byte `0xDB`
- Detect violations of the Python memory allocator API. For example, `PyObject_Free()` called on a memory block allocated by `PyMem_Malloc()`.
- Detect writes before the start of a buffer (buffer underflows)
- Detect writes after the end of a buffer (buffer overflows)
- Check that the GIL is held when allocator functions of `PYMEM_DOMAIN_OBJ` (ex: `PyObject_Malloc()`) and `PYMEM_DOMAIN_MEM` (ex: `PyMem_Malloc()`) domains are called.

Checking if the GIL is held is also a new feature of Python 3.6.

See the `PyMem_SetupDebugHooks()` function for debug hooks on Python memory allocators.

It is now also possible to force the usage of the `malloc()` allocator of the C library for all Python memory allocations using `PYTHONMALLOC=malloc`. This is helpful when using external memory debuggers like Valgrind on a Python compiled in release mode.

On error, the debug hooks on Python memory allocators now use the `tracemalloc` module to get the traceback where a memory block was allocated.

Example of fatal error on buffer overflow using `python3.6 -X tracemalloc=5` (store 5 frames in traces):

```
Debug memory block at address p=0x7fbcd41666f8: API 'o'
  4 bytes originally requested
The 7 pad bytes at p-7 are FORBIDDENBYTE, as expected.
The 8 pad bytes at tail=0x7fbcd41666fc are not all FORBIDDENBYTE (0xfb):
  at tail+0: 0x02 *** OUCH
  at tail+1: 0xfb
  at tail+2: 0xfb
  at tail+3: 0xfb
  at tail+4: 0xfb
  at tail+5: 0xfb
  at tail+6: 0xfb
  at tail+7: 0xfb
```

```
The block was made by call #1233329 to debug malloc/realloc.  
Data at p: 1a 2b 30 00
```

Memory block allocated at (most recent call first):

```
File "test/test_bytes.py", line 323  
File "unittest/case.py", line 600  
File "unittest/case.py", line 648  
File "unittest/suite.py", line 122  
File "unittest/suite.py", line 84
```

Fatal Python error: bad trailing pad byte

Current thread 0x00007fbcd32700 (most recent call first):

```
File "test/test_bytes.py", line 323 in test_hex  
File "unittest/case.py", line 600 in run  
File "unittest/case.py", line 648 in __call__  
File "unittest/suite.py", line 122 in run  
File "unittest/suite.py", line 84 in __call__  
File "unittest/suite.py", line 122 in run  
File "unittest/suite.py", line 84 in __call__  
...
```

(Contributed by Victor Stinner in [issue 26516](#) and [issue 26564](#).)

## 2.17 DTrace and SystemTap probing support

Python can now be built `--with-dtrace` which enables static markers for the following events in the interpreter:

- function call/return
- garbage collection started/finished
- line of code executed.

This can be used to instrument running interpreters in production, without the need to recompile specific debug builds or providing application-specific profiling/debugging code.

More details in instrumentation.

The current implementation is tested on Linux and macOS. Additional markers may be added in the future.

(Contributed by Łukasz Langa in [issue 21590](#), based on patches by Jesús Cea Avi3n, David Malcolm, and Nikhil Benesch.)

## 3 Other Language Changes

Some smaller changes made to the core Python language are:

- A `global` or `nonlocal` statement must now textually appear before the first use of the affected name in the same scope. Previously this was a `SyntaxWarning`.
- It is now possible to set a special method to `None` to indicate that the corresponding operation is not available. For example, if a class sets `__iter__()` to `None`, the class is not iterable. (Contributed by Andrew Barnert and Ivan Levkivskyi in [issue 25958](#).)
- Long sequences of repeated traceback lines are now abbreviated as "[Previous line repeated {count} more times]" (see [traceback](#) for an example). (Contributed by Emanuel Barry in [issue 26823](#).)

- `Import` now raises the new exception `ModuleNotFoundError` (subclass of `ImportError`) when it cannot find a module. Code that currently checks for `ImportError` (in `try-except`) will still work. (Contributed by Eric Snow in [issue 15767](#).)
- Class methods relying on zero-argument `super()` will now work correctly when called from metaclass methods during class creation. (Contributed by Martin Teichmann in [issue 23722](#).)

## 4 New Modules

### 4.1 secrets

The main purpose of the new `secrets` module is to provide an obvious way to reliably generate cryptographically strong pseudo-random values suitable for managing secrets, such as account authentication, tokens, and similar.

**Warning:** Note that the pseudo-random generators in the `random` module should *NOT* be used for security purposes. Use `secrets` on Python 3.6+ and `os.urandom()` on Python 3.5 and earlier.

See also:

**PEP 506 – Adding A Secrets Module To The Standard Library** PEP written and implemented by Steven D’Aprano.

## 5 Improved Modules

### 5.1 array

Exhausted iterators of `array.array` will now stay exhausted even if the iterated array is extended. This is consistent with the behavior of other mutable sequences.

Contributed by Serhiy Storchaka in [issue 26492](#).

### 5.2 ast

The new `ast.Constant` AST node has been added. It can be used by external AST optimizers for the purposes of constant folding.

Contributed by Victor Stinner in [issue 26146](#).

### 5.3 asyncio

Starting with Python 3.6 the `asyncio` module is no longer provisional and its API is considered stable.

Notable changes in the `asyncio` module since Python 3.5.0 (all backported to 3.5.x due to the provisional status):

- The `get_event_loop()` function has been changed to always return the currently running loop when called from couroutines and callbacks. (Contributed by Yury Selivanov in [issue 28613](#).)
- The `ensure_future()` function and all functions that use it, such as `loop.run_until_complete()`, now accept all kinds of awaitable objects. (Contributed by Yury Selivanov.)
- New `run_coroutine_threadsafe()` function to submit coroutines to event loops from other threads. (Contributed by Vincent Michel.)

- New `Transport.is_closing()` method to check if the transport is closing or closed. (Contributed by Yury Selivanov.)
- The `loop.create_server()` method can now accept a list of hosts. (Contributed by Yann Sionneau.)
- New `loop.create_future()` method to create `Future` objects. This allows alternative event loop implementations, such as `uvloop`, to provide a faster `asyncio.Future` implementation. (Contributed by Yury Selivanov in [issue 27041](#).)
- New `loop.get_exception_handler()` method to get the current exception handler. (Contributed by Yury Selivanov in [issue 27040](#).)
- New `StreamReader.readuntil()` method to read data from the stream until a separator bytes sequence appears. (Contributed by Mark Korenberg.)
- The performance of `StreamReader.readexactly()` has been improved. (Contributed by Mark Korenberg in [issue 28370](#).)
- The `loop.getaddrinfo()` method is optimized to avoid calling the system `getaddrinfo` function if the address is already resolved. (Contributed by A. Jesse Jiryu Davis.)
- The `BaseEventLoop.stop()` method has been changed to stop the loop immediately after the current iteration. Any new callbacks scheduled as a result of the last iteration will be discarded. (Contributed by Guido van Rossum in [issue 25593](#).)
- `Future.set_exception` will now raise `TypeError` when passed an instance of the `StopIteration` exception. (Contributed by Chris Angelico in [issue 26221](#).)
- New `Loop.connect_accepted_socket()` method to be used by servers that accept connections outside of `asyncio`, but that use `asyncio` to handle them. (Contributed by Jim Fulton in [issue 27392](#).)

## 5.4 binascii

The `b2a_base64()` function now accepts an optional `newline` keyword argument to control whether the new-line character is appended to the return value. (Contributed by Victor Stinner in [issue 25357](#).)

## 5.5 cmath

The new `cmath.tau` ( $\tau$ ) constant has been added. (Contributed by Lisa Roach in [issue 12345](#), see [PEP 628](#) for details.)

New constants: `cmath.inf` and `cmath.nan` to match `math.inf` and `math.nan`, and also `cmath.infj` and `cmath.nanj` to match the format used by complex repr. (Contributed by Mark Dickinson in [issue 23229](#).)

## 5.6 collections

The new `Collection` abstract base class has been added to represent sized iterable container classes. (Contributed by Ivan Levkivskyi, docs by Neil Girdhar in [issue 27598](#).)

The new `Reversible` abstract base class represents iterable classes that also provide the `__reversed__()` method. (Contributed by Ivan Levkivskyi in [issue 25987](#).)

The new `AsyncGenerator` abstract base class represents asynchronous generators. (Contributed by Yury Selivanov in [issue 28720](#).)

The `namedtuple()` function now accepts an optional keyword argument `module`, which, when specified, is used for the `__module__` attribute of the returned named tuple class. (Contributed by Raymond Hettinger in [issue 17941](#).)

The `verbose` and `rename` arguments for `namedtuple()` are now keyword-only. (Contributed by Raymond Hettinger in [issue 25628](#).)

Recursive `collections.deque` instances can now be pickled. (Contributed by Serhiy Storchaka in [issue 26482](#).)

## 5.7 concurrent.futures

The `ThreadPoolExecutor` class constructor now accepts an optional `thread_name_prefix` argument to make it possible to customize the names of the threads created by the pool. (Contributed by Gregory P. Smith in [issue 27664](#).)

## 5.8 contextlib

The `contextlib.AbstractContextManager` class has been added to provide an abstract base class for context managers. It provides a sensible default implementation for `__enter__()` which returns `self` and leaves `__exit__()` an abstract method. A matching class has been added to the `typing` module as `typing.ContextManager`. (Contributed by Brett Cannon in [issue 25609](#).)

## 5.9 datetime

The `datetime` and `time` classes have the new `fold` attribute used to disambiguate local time when necessary. Many functions in the `datetime` have been updated to support local time disambiguation. See [Local Time Disambiguation](#) section for more information. (Contributed by Alexander Belopolsky in [issue 24773](#).)

The `datetime.strptime()` and `date.strptime()` methods now support ISO 8601 date directives `%G`, `%u` and `%V`. (Contributed by Ashley Anderson in [issue 12006](#).)

The `datetime.isoformat()` function now accepts an optional `timespec` argument that specifies the number of additional components of the time value to include. (Contributed by Alessandro Cucci and Alexander Belopolsky in [issue 19475](#).)

The `datetime.combine()` now accepts an optional `tzinfo` argument. (Contributed by Alexander Belopolsky in [issue 27661](#).)

## 5.10 decimal

New `Decimal.as_integer_ratio()` method that returns a pair  $(n, d)$  of integers that represent the given `Decimal` instance as a fraction, in lowest terms and with a positive denominator:

```
>>> Decimal('-3.14').as_integer_ratio()
(-157, 50)
```

(Contributed by Stefan Krah and Mark Dickinson in [issue 25928](#).)

## 5.11 distutils

The `default_format` attribute has been removed from `distutils.command.sdist.sdist` and the `formats` attribute defaults to `['gztar']`. Although not anticipated, any code relying on the presence of `default_format` may need to be adapted. See [issue 27819](#) for more details.

## 5.12 email

The new email API, enabled via the `policy` keyword to various constructors, is no longer provisional. The email documentation has been reorganized and rewritten to focus on the new API, while retaining the old documentation for the legacy API. (Contributed by R. David Murray in [issue 24277](#).)

The `email.mime` classes now all accept an optional `policy` keyword. (Contributed by Berker Peksag in [issue 27331](#).)

The `DecodedGenerator` now supports the *policy* keyword.

There is a new `policy` attribute, `message_factory`, that controls what class is used by default when the parser creates new message objects. For the `email.policy.compat32` policy this is `Message`, for the new policies it is `EmailMessage`. (Contributed by R. David Murray in [issue 20476](#).)

## 5.13 encodings

On Windows, added the `'oem'` encoding to use `CP_OEMCP`, and the `'ansi'` alias for the existing `'mbcs'` encoding, which uses the `CP_ACP` code page. (Contributed by Steve Dower in [issue 27959](#).)

## 5.14 enum

Two new enumeration base classes have been added to the `enum` module: `Flag` and `IntFlags`. Both are used to define constants that can be combined using the bitwise operators. (Contributed by Ethan Furman in [issue 23591](#).)

Many standard library modules have been updated to use the `IntFlags` class for their constants.

The new `enum.auto` value can be used to assign values to enum members automatically:

```
>>> from enum import Enum, auto
>>> class Color(Enum):
...     red = auto()
...     blue = auto()
...     green = auto()
...
>>> list(Color)
[<Color.red: 1>, <Color.blue: 2>, <Color.green: 3>]
```

## 5.15 faulthandler

On Windows, the `faulthandler` module now installs a handler for Windows exceptions: see `faulthandler.enable()`. (Contributed by Victor Stinner in [issue 23848](#).)

## 5.16 fileinput

`hook_encoded()` now supports the *errors* argument. (Contributed by Joseph Hackman in [issue 25788](#).)

## 5.17 hashlib

`hashlib` supports OpenSSL 1.1.0. The minimum recommend version is 1.0.2. (Contributed by Christian Heimes in [issue 26470](#).)

BLAKE2 hash functions were added to the module. `blake2b()` and `blake2s()` are always available and support the full feature set of BLAKE2. (Contributed by Christian Heimes in [issue 26798](#) based on code by Dmitry Chestnykh and Samuel Neves. Documentation written by Dmitry Chestnykh.)

The SHA-3 hash functions `sha3_224()`, `sha3_256()`, `sha3_384()`, `sha3_512()`, and SHAKE hash functions `shake_128()` and `shake_256()` were added. (Contributed by Christian Heimes in [issue 16113](#). Keccak Code Package by Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer.)

The password-based key derivation function `scrypt()` is now available with OpenSSL 1.1.0 and newer. (Contributed by Christian Heimes in [issue 27928](#).)

## 5.18 http.client

`HTTPConnection.request()` and `endheaders()` both now support chunked encoding request bodies. (Contributed by Demian Brecht and Rolf Krahle in [issue 12319](#).)

## 5.19 idlelib and IDLE

The `idlelib` package is being modernized and refactored to make IDLE look and work better and to make the code easier to understand, test, and improve. Part of making IDLE look better, especially on Linux and Mac, is using `ttk` widgets, mostly in the dialogs. As a result, IDLE no longer runs with `tcl/tk` 8.4. It now requires `tcl/tk` 8.5 or 8.6. We recommend running the latest release of either.

‘Modernizing’ includes renaming and consolidation of `idlelib` modules. The renaming of files with partial upper-case names is similar to the renaming of, for instance, `Tkinter` and `TkFont` to `tkinter` and `tkinter.font` in 3.0. As a result, imports of `idlelib` files that worked in 3.5 will usually not work in 3.6. At least a module name change will be needed (see `idlelib/README.txt`), sometimes more. (Name changes contributed by Al Swiegart and Terry Reedy in [issue 24225](#). Most `idlelib` patches since have been and will be part of the process.)

In compensation, the eventual result will be that some `idlelib` classes will be easier to use, with better APIs and docstrings explaining them. Additional useful information will be added to `idlelib` when available.

## 5.20 importlib

`Import` now raises the new exception `ModuleNotFoundError` (subclass of `ImportError`) when it cannot find a module. Code that currently checks for `ImportError` (in `try-except`) will still work. (Contributed by Eric Snow in [issue 15767](#).)

`importlib.util.LazyLoader` now calls `create_module()` on the wrapped loader, removing the restriction that `importlib.machinery.BuiltinImporter` and `importlib.machinery.ExtensionFileLoader` couldn’t be used with `importlib.util.LazyLoader`.

`importlib.util.cache_from_source()`, `importlib.util.source_from_cache()`, and `importlib.util.spec_from_file_location()` now accept a path-like object.

## 5.21 inspect

The `inspect.signature()` function now reports the implicit `.0` parameters generated by the compiler for comprehension and generator expression scopes as if they were positional-only parameters called `implicit0`. (Contributed by Jelle Zijlstra in [issue 19611](#).)

To reduce code churn when upgrading from Python 2.7 and the legacy `inspect.getargspec()` API, the previously documented deprecation of `inspect.getfullargspec()` has been reversed. While this function is convenient for single/source Python 2/3 code bases, the richer `inspect.signature()` interface remains the recommended approach for new code. (Contributed by Nick Coghlan in [issue 27172](#).)

## 5.22 json

`json.load()` and `json.loads()` now support binary input. Encoded JSON should be represented using either UTF-8, UTF-16, or UTF-32. (Contributed by Serhiy Storchaka in [issue 17909](#).)

## 5.23 logging

The new `WatchedFileHandler.reopenIfNeeded()` method has been added to add the ability to check if the log file needs to be reopened. (Contributed by Marian Horban in [issue 24884](#).)



## 5.24 math

The tau ( $\tau$ ) constant has been added to the `math` and `cmath` modules. (Contributed by Lisa Roach in [issue 12345](#), see [PEP 628](#) for details.)

## 5.25 multiprocessing

Proxy Objects returned by `multiprocessing.Manager()` can now be nested. (Contributed by Davin Potts in [issue 6766](#).)

## 5.26 os

See the summary of [PEP 519](#) for details on how the `os` and `os.path` modules now support path-like objects. `scandir()` now supports `bytes` paths on Windows.

A new `close()` method allows explicitly closing a `scandir()` iterator. The `scandir()` iterator now supports the context manager protocol. If a `scandir()` iterator is neither exhausted nor explicitly closed a `ResourceWarning` will be emitted in its destructor. (Contributed by Serhiy Storchaka in [issue 25994](#).)

On Linux, `os.urandom()` now blocks until the system urandom entropy pool is initialized to increase the security. See the [PEP 524](#) for the rationale.

The Linux `getrandom()` syscall (get random bytes) is now exposed as the new `os.getrandom()` function. (Contributed by Victor Stinner, part of the [PEP 524](#))

## 5.27 pathlib

`pathlib` now supports path-like objects. (Contributed by Brett Cannon in [issue 27186](#).)

See the summary of [PEP 519](#) for details.

## 5.28 pdb

The `Pdb` class constructor has a new optional `readrc` argument to control whether `.pdbrc` files should be read.

## 5.29 pickle

Objects that need `__new__` called with keyword arguments can now be pickled using pickle protocols older than protocol version 4. Protocol version 4 already supports this case. (Contributed by Serhiy Storchaka in [issue 24164](#).)

## 5.30 pickletools

`pickletools.dis()` now outputs the implicit memo index for the `MEMOIZE` opcode. (Contributed by Serhiy Storchaka in [issue 25382](#).)

## 5.31 pydoc

The `pydoc` module has learned to respect the `MANPAGER` environment variable. (Contributed by Matthias Klose in [issue 8637](#).)

`help()` and `pydoc` can now list named tuple fields in the order they were defined rather than alphabetically. (Contributed by Raymond Hettinger in [issue 24879](#).)

## 5.32 random

The new `choices()` function returns a list of elements of specified size from the given population with optional weights. (Contributed by Raymond Hettinger in [issue 18844](#).)

## 5.33 re

Added support of modifier spans in regular expressions. Examples: `'(?:p)ython'` matches `'python'` and `'Python'`, but not `'PYTHON'`; `'(?:g)(?-i:v)r'` matches `'GvR'` and `'gvr'`, but not `'GVR'`. (Contributed by Serhiy Storchaka in [issue 433028](#).)

Match object groups can be accessed by `__getitem__`, which is equivalent to `group()`. So `mo['name']` is now equivalent to `mo.group('name')`. (Contributed by Eric Smith in [issue 24454](#).)

Match objects now support index-like objects as group indices. (Contributed by Jeroen Demeyer and Xiang Zhang in [issue 27177](#).)

## 5.34 readline

Added `set_auto_history()` to enable or disable automatic addition of input to the history list. (Contributed by Tyler Crompton in [issue 26870](#).)

## 5.35 rlcompleter

Private and special attribute names now are omitted unless the prefix starts with underscores. A space or a colon is added after some completed keywords. (Contributed by Serhiy Storchaka in [issue 25011](#) and [issue 25209](#).)

## 5.36 shlex

The `shlex` has much improved shell compatibility through the new `punctuation_chars` argument to control which characters are treated as punctuation. (Contributed by Vinay Sajip in [issue 1521950](#).)

## 5.37 site

When specifying paths to add to `sys.path` in a `.pth` file, you may now specify file paths on top of directories (e.g. zip files). (Contributed by Wolfgang Langner in [issue 26587](#).)

## 5.38 sqlite3

`sqlite3.Cursor.lastrowid` now supports the `REPLACE` statement. (Contributed by Alex LordThorsen in [issue 16864](#).)

## 5.39 socket

The `ioctl()` function now supports the `SIO_LOOPBACK_FAST_PATH` control code. (Contributed by Daniel Stokes in [issue 26536](#).)

The `getsockopt()` constants `SO_DOMAIN`, `SO_PROTOCOL`, `SO_PEERSEC`, and `SO_PASSSEC` are now supported. (Contributed by Christian Heimes in [issue 26907](#).)

The `setsockopt()` now supports the `setsockopt(level, optname, None, optlen: int)` form. (Contributed by Christian Heimes in [issue 27744](#).)

The `socket` module now supports the address family `AF_ALG` to interface with Linux Kernel crypto API. `ALG_*`, `SOL_ALG` and `sendmsg_afalg()` were added. (Contributed by Christian Heimes in [issue 27744](#) with support from Victor Stinner.)

## 5.40 socketserver

Servers based on the `socketserver` module, including those defined in `http.server`, `xmlrpc.server` and `wsgiref.simple_server`, now support the context manager protocol. (Contributed by Aviv Palivoda in [issue 26404](#).)

The `wfile` attribute of `StreamRequestHandler` classes now implements the `io.BufferedIOBase` writable interface. In particular, calling `write()` is now guaranteed to send the data in full. (Contributed by Martin Panter in [issue 26721](#).)

## 5.41 ssl

`ssl` supports OpenSSL 1.1.0. The minimum recommend version is 1.0.2. (Contributed by Christian Heimes in [issue 26470](#).)

3DES has been removed from the default cipher suites and ChaCha20 Poly1305 cipher suites have been added. (Contributed by Christian Heimes in [issue 27850](#) and [issue 27766](#).)

`SSLContext` has better default configuration for options and ciphers. (Contributed by Christian Heimes in [issue 28043](#).)

SSL session can be copied from one client-side connection to another with the new `SSLSession` class. TLS session resumption can speed up the initial handshake, reduce latency and improve performance (Contributed by Christian Heimes in [issue 19500](#) based on a draft by Alex Warhawk.)

The new `get_ciphers()` method can be used to get a list of enabled ciphers in order of cipher priority.

All constants and flags have been converted to `IntEnum` and `IntFlags`. (Contributed by Christian Heimes in [issue 28025](#).)

Server and client-side specific TLS protocols for `SSLContext` were added. (Contributed by Christian Heimes in [issue 28085](#).)

## 5.42 statistics

A new `harmonic_mean()` function has been added. (Contributed by Steven D'Aprano in [issue 27181](#).)

## 5.43 struct

`struct` now supports IEEE 754 half-precision floats via the `'e'` format specifier. (Contributed by Eli Stevens, Mark Dickinson in [issue 11734](#).)

## 5.44 subprocess

`subprocess.Popen` destructor now emits a `ResourceWarning` warning if the child process is still running. Use the context manager protocol (with `proc: ...`) or explicitly call the `wait()` method to read the exit status of the child process. (Contributed by Victor Stinner in [issue 26741](#).)

The `subprocess.Popen` constructor and all functions that pass arguments through to it now accept *encoding* and *errors* arguments. Specifying either of these will enable text mode for the *stdin*, *stdout* and *stderr* streams. (Contributed by Steve Dower in [issue 6135](#).)

## 5.45 sys

The new `getfilesystemencodeerrors()` function returns the name of the error mode used to convert between Unicode filenames and bytes filenames. (Contributed by Steve Dower in [issue 27781](#).)

On Windows the return value of the `getwindowsversion()` function now includes the *platform\_version* field which contains the accurate major version, minor version and build number of the current operating system, rather than the version that is being emulated for the process (Contributed by Steve Dower in [issue 27932](#).)

## 5.46 telnetlib

`Telnet` is now a context manager (contributed by Stéphane Wirtel in [issue 25485](#)).

## 5.47 time

The `struct_time` attributes `tm_gmtoff` and `tm_zone` are now available on all platforms.

## 5.48 timeit

The new `Timer.autorange()` convenience method has been added to call `Timer.timeit()` repeatedly so that the total run time is greater or equal to 200 milliseconds. (Contributed by Steven D'Aprano in [issue 6422](#).)

`timeit` now warns when there is substantial (4x) variance between best and worst times. (Contributed by Serhiy Storchaka in [issue 23552](#).)

## 5.49 tkinter

Added methods `trace_add()`, `trace_remove()` and `trace_info()` in the `tkinter.Variable` class. They replace old methods `trace_variable()`, `trace()`, `trace_vdelete()` and `trace_vinfo()` that use obsolete Tcl commands and might not work in future versions of Tcl. (Contributed by Serhiy Storchaka in [issue 22115](#)).

## 5.50 traceback

Both the `traceback` module and the interpreter's builtin exception display now abbreviate long sequences of repeated lines in tracebacks as shown in the following example:

```
>>> def f(): f()
...
>>> f()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 1, in f
  File "<stdin>", line 1, in f
  File "<stdin>", line 1, in f
  [Previous line repeated 995 more times]
RecursionError: maximum recursion depth exceeded
```

(Contributed by Emanuel Barry in [issue 26823](#).)

## 5.51 tracemalloc

The `tracemalloc` module now supports tracing memory allocations in multiple different address spaces.

The new `DomainFilter` filter class has been added to filter block traces by their address space (domain).

(Contributed by Victor Stinner in [issue 26588](#).)

## 5.52 typing

Starting with Python 3.6 the `typing` module is no longer provisional and its API is considered stable.

Since the `typing` module was provisional in Python 3.5, all changes introduced in Python 3.6 have also been backported to Python 3.5.x.

The `typing` module has a much improved support for generic type aliases. For example `Dict[str, Tuple[S, T]]` is now a valid type annotation. (Contributed by Guido van Rossum in [Github #195](#).)

The `typing.ContextManager` class has been added for representing `contextlib.AbstractContextManager`. (Contributed by Brett Cannon in [issue 25609](#).)

The `typing.Collection` class has been added for representing `collections.abc.Collection`. (Contributed by Ivan Levkivskyi in [issue 27598](#).)

The `typing.ClassVar` type construct has been added to mark class variables. As introduced in [PEP 526](#), a variable annotation wrapped in `ClassVar` indicates that a given attribute is intended to be used as a class variable and should not be set on instances of that class. (Contributed by Ivan Levkivskyi in [Github #280](#).)

A new `TYPE_CHECKING` constant that is assumed to be `True` by the static type checkers, but is `False` at runtime. (Contributed by Guido van Rossum in [Github #230](#).)

A new `NewType()` helper function has been added to create lightweight distinct types for annotations:

```
from typing import NewType
```

```
UserId = NewType('UserId', int)
some_id = UserId(524313)
```

The static type checker will treat the new type as if it were a subclass of the original type. (Contributed by Ivan Levkivskyi in [Github #189](#).)

## 5.53 unicodedata

The `unicodedata` module now uses data from [Unicode 9.0.0](#). (Contributed by Benjamin Peterson.)

## 5.54 unittest.mock

The `Mock` class has the following improvements:

- Two new methods, `Mock.assert_called()` and `Mock.assert_called_once()` to check if the mock object was called. (Contributed by Amit Saha in [issue 26323](#).)
- The `Mock.reset_mock()` method now has two optional keyword only arguments: *return\_value* and *side\_effect*. (Contributed by Kushal Das in [issue 21271](#).)

## 5.55 urllib.request

If a HTTP request has a file or iterable body (other than a bytes object) but no `Content-Length` header, rather than throwing an error, `AbstractHTTPHandler` now falls back to use chunked transfer encoding. (Contributed by Demian Brecht and Rolf Krahel in [issue 12319](#).)

## 5.56 urllib.robotparser

`RobotFileParser` now supports the `Crawl-delay` and `Request-rate` extensions. (Contributed by Nikolay Bogoychev in [issue 16099](#).)

## 5.57 venv

`venv` accepts a new parameter `--prompt`. This parameter provides an alternative prefix for the virtual environment. (Proposed by Łukasz Balcerzak and ported to 3.6 by Stéphane Wirtel in [issue 22829](#).)

## 5.58 warnings

A new optional `source` parameter has been added to the `warnings.warn_explicit()` function: the destroyed object which emitted a `ResourceWarning`. A `source` attribute has also been added to `warnings.WarningMessage` (contributed by Victor Stinner in [issue 26568](#) and [issue 26567](#)).

When a `ResourceWarning` warning is logged, the `tracemalloc` module is now used to try to retrieve the traceback where the destroyed object was allocated.

Example with the script `example.py`:

```
import warnings

def func():
    return open(__file__)

f = func()
f = None
```

Output of the command `python3.6 -Wd -X tracemalloc=5 example.py`:

```
example.py:7: ResourceWarning: unclosed file <_io.TextIOWrapper name='example.py' mode='w'>
  f = None
Object allocated at (most recent call first):
  File "example.py", lineno 4
    return open(__file__)
  File "example.py", lineno 6
    f = func()
```

The “Object allocated at” traceback is new and is only displayed if `tracemalloc` is tracing Python memory allocations and if the `warnings` module was already imported.

## 5.59 winreg

Added the 64-bit integer type `REG_QWORD`. (Contributed by Clement Rouault in [issue 23026](#).)

## 5.60 winsound

Allowed keyword arguments to be passed to `Beep`, `MessageBeep`, and `PlaySound` ([issue 27982](#)).

## 5.61 xmlrpc.client

The `xmlrpc.client` module now supports unmarshalling additional data types used by the Apache XML-RPC implementation for numerics and `None`. (Contributed by Serhiy Storchaka in [issue 26885](#).)

## 5.62 zipfile

A new `ZipInfo.from_file()` class method allows making a `ZipInfo` instance from a filesystem file. A new `ZipInfo.is_dir()` method can be used to check if the `ZipInfo` instance represents a directory. (Contributed by Thomas Kluyver in [issue 26039](#).)

The `ZipFile.open()` method can now be used to write data into a ZIP file, as well as for extracting data. (Contributed by Thomas Kluyver in [issue 26039](#).)

## 5.63 zlib

The `compress()` and `decompress()` functions now accept keyword arguments. (Contributed by Aviv Palivoda in [issue 26243](#) and Xiang Zhang in [issue 16764](#) respectively.)

## 6 Optimizations

- The Python interpreter now uses a 16-bit wordcode instead of bytecode which made a number of opcode optimizations possible. (Contributed by Demur Rumed with input and reviews from Serhiy Storchaka and Victor Stinner in [issue 26647](#) and [issue 28050](#).)
- The `Future` class now has an optimized C implementation. (Contributed by Yury Selivanov and INADA Naoki in [issue 26801](#).)
- The `Task` class now has an optimized C implementation. (Contributed by Yury Selivanov in [issue 28544](#).)
- Various implementation improvements in the `typing` module (such as caching of generic types) allow up to 30 times performance improvements and reduced memory footprint.
- The ASCII decoder is now up to 60 times as fast for error handlers `surrogateescape`, `ignore` and `replace` (Contributed by Victor Stinner in [issue 24870](#)).
- The ASCII and the Latin1 encoders are now up to 3 times as fast for the error handler `surrogateescape` (Contributed by Victor Stinner in [issue 25227](#)).
- The UTF-8 encoder is now up to 75 times as fast for error handlers `ignore`, `replace`, `surrogateescape`, `surrogatepass` (Contributed by Victor Stinner in [issue 25267](#)).
- The UTF-8 decoder is now up to 15 times as fast for error handlers `ignore`, `replace` and `surrogateescape` (Contributed by Victor Stinner in [issue 25301](#)).
- `bytes % args` is now up to 2 times faster. (Contributed by Victor Stinner in [issue 25349](#)).
- `bytearray % args` is now between 2.5 and 5 times faster. (Contributed by Victor Stinner in [issue 25399](#)).
- Optimize `bytes.fromhex()` and `bytearray.fromhex()` : they are now between 2x and 3.5x faster. (Contributed by Victor Stinner in [issue 25401](#)).
- Optimize `bytes.replace(b'', b'.')` and `bytearray.replace(b'', b'.')` : up to 80% faster. (Contributed by Josh Snider in [issue 26574](#)).
- Allocator functions of the `PyMem_Malloc()` domain (`PYMEM_DOMAIN_MEM`) now use the `pymalloc` memory allocator instead of `malloc()` function of the C library. The `pymalloc` allocator is optimized for objects smaller or equal to 512 bytes with a short lifetime, and use `malloc()` for larger memory blocks. (Contributed by Victor Stinner in [issue 26249](#)).
- `pickle.load()` and `pickle.loads()` are now up to 10% faster when deserializing many small objects (Contributed by Victor Stinner in [issue 27056](#)).
- Passing keyword arguments to a function has an overhead in comparison with passing positional arguments. Now in extension functions implemented with using Argument Clinic this overhead is significantly decreased. (Contributed by Serhiy Storchaka in [issue 27574](#)).
- Optimized `glob()` and `iglob()` functions in the `glob` module; they are now about 3–6 times faster. (Contributed by Serhiy Storchaka in [issue 25596](#)).
- Optimized globbing in `pathlib` by using `os.scandir()` ; it is now about 1.5–4 times faster. (Contributed by Serhiy Storchaka in [issue 26032](#)).
- `xml.etree.ElementTree` parsing, iteration and deepcopy performance has been significantly improved. (Contributed by Serhiy Storchaka in [issue 25638](#), [issue 25873](#), and [issue 25869](#)).
- Creation of `fractions.Fraction` instances from floats and decimals is now 2 to 3 times faster. (Contributed by Serhiy Storchaka in [issue 25971](#).)

## 7 Build and C API Changes

- Python now requires some C99 support in the toolchain to build. Most notably, Python now uses standard integer types and macros in place of custom macros like `PY_LONG_LONG`. For more information, see [PEP 7](#) and [issue 17884](#).
- Cross-compiling CPython with the Android NDK and the Android API level set to 21 (Android 5.0 Lollipop) or greater runs successfully. While Android is not yet a supported platform, the Python test suite runs on the Android emulator with only about 16 tests failures. See the Android meta-issue [issue 26865](#).
- The `--enable-optimizations` configure flag has been added. Turning it on will activate expensive optimizations like PGO. (Original patch by Alecsandru Patrascu of Intel in [issue 26539](#).)
- The GIL must now be held when allocator functions of `PYMEM_DOMAIN_OBJ` (ex: `PyObject_Malloc()`) and `PYMEM_DOMAIN_MEM` (ex: `PyMem_Malloc()`) domains are called.
- New `Py_FinalizeEx()` API which indicates if flushing buffered data failed. (Contributed by Martin Panter in [issue 5319](#).)
- `PyArg_ParseTupleAndKeywords()` now supports positional-only parameters. Positional-only parameters are defined by empty names. (Contributed by Serhiy Storchaka in [issue 26282](#).)
- `PyTraceback_Print` method now abbreviates long sequences of repeated lines as "[Previous line repeated {count} more times]". (Contributed by Emanuel Barry in [issue 26823](#).)
- The new `PyErr_SetImportErrorSubclass()` function allows for specifying a subclass of `ImportError` to raise. (Contributed by Eric Snow in [issue 15767](#).)
- The new `PyErr_ResourceWarning()` function can be used to generate a `ResourceWarning` providing the source of the resource allocation. (Contributed by Victor Stinner in [issue 26567](#).)
- The new `PyOS_FSPath()` function returns the file system representation of a path-like object. (Contributed by Brett Cannon in [issue 27186](#).)
- The `PyUnicode_FSConverter()` and `PyUnicode_FSDecoder()` functions will now accept path-like objects.

## 8 Other Improvements

- When `--version` (short form: `-V`) is supplied twice, Python prints `sys.version` for detailed information.

```
$ ./python -VV
Python 3.6.0b4+ (3.6:223967b49e49+, Nov 21 2016, 20:55:04)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.42.1)]
```

## 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. Starting in Python 3.6, the use of `async` or `await` as names will generate a `DeprecationWarning`.

### 9.2 Deprecated Python behavior

Raising the `StopIteration` exception inside a generator will now generate a `DeprecationWarning`, and will trigger a `RuntimeError` in Python 3.7. See [whatsnew-pep-479](#) for details.



The `__aiter__()` method is now expected to return an asynchronous iterator directly instead of returning an awaitable as previously. Doing the former will trigger a `DeprecationWarning`. Backward compatibility will be removed in Python 3.7. (Contributed by Yury Selivanov in [issue 27243](#).)

A backslash-character pair that is not a valid escape sequence now generates a `DeprecationWarning`. Although this will eventually become a `SyntaxError`, that will not be for several Python releases. (Contributed by Emanuel Barry in [issue 27364](#).)

When performing a relative import, falling back on `__name__` and `__path__` from the calling module when `__spec__` or `__package__` are not defined now raises an `ImportWarning`. (Contributed by Rose Ames in [issue 25791](#).)

## 9.3 Deprecated Python modules, functions and methods

### asynchat

The `asynchat` has been deprecated in favor of `asyncio`. (Contributed by Mariatta in [issue 25002](#).)

### asyncore

The `asyncore` has been deprecated in favor of `asyncio`. (Contributed by Mariatta in [issue 25002](#).)

### dbm

Unlike other `dbm` implementations, the `dbm.dumb` module creates databases with the `'rw'` mode and allows modifying the database opened with the `'r'` mode. This behavior is now deprecated and will be removed in 3.8. (Contributed by Serhiy Storchaka in [issue 21708](#).)

### distutils

The undocumented `extra_path` argument to the `Distribution` constructor is now considered deprecated and will raise a warning if set. Support for this parameter will be removed in a future Python release. See [issue 27919](#) for details.

### grp

The support of non-integer arguments in `getgrgid()` has been deprecated. (Contributed by Serhiy Storchaka in [issue 26129](#).)

### importlib

The `importlib.machinery.SourceFileLoader.load_module()` and `importlib.machinery.SourcelessFileLoader.load_module()` methods are now deprecated. They were the only remaining implementations of `importlib.abc.Loader.load_module()` in `importlib` that had not been deprecated in previous versions of Python in favour of `importlib.abc.Loader.exec_module()`.

### os

Undocumented support of general bytes-like objects as paths in `os` functions, `compile()` and similar functions is now deprecated. (Contributed by Serhiy Storchaka in [issue 25791](#) and [issue 26754](#).)

## re

Support for inline flags (`?letters`) in the middle of the regular expression has been deprecated and will be removed in a future Python version. Flags at the start of a regular expression are still allowed. (Contributed by Serhiy Storchaka in [issue 22493](#).)

## ssl

OpenSSL 0.9.8, 1.0.0 and 1.0.1 are deprecated and no longer supported. In the future the `ssl` module will require at least OpenSSL 1.0.2 or 1.1.0.

SSL-related arguments like `certfile`, `keyfile` and `check_hostname` in `ftplib`, `http.client`, `imaplib`, `poplib`, and `smtplib` have been deprecated in favor of `context`. (Contributed by Christian Heimes in [issue 28022](#).)

A couple of protocols and functions of the `ssl` module are now deprecated. Some features will no longer be available in future versions of OpenSSL. Other features are deprecated in favor of a different API. (Contributed by Christian Heimes in [issue 28022](#) and [issue 26470](#).)

## tkinter

The `tkinter.tix` module is now deprecated. `tkinter` users should use `tkinter.ttk` instead.

## venv

The `pyvenv` script has been deprecated in favour of `python3 -m venv`. This prevents confusion as to what Python interpreter `pyvenv` is connected to and thus what Python interpreter will be used by the virtual environment. (Contributed by Brett Cannon in [issue 25154](#).)

## 9.4 Deprecated functions and types of the C API

Undocumented functions `PyUnicode_AsEncodedObject()`, `PyUnicode_AsDecodedObject()`, `PyUnicode_AsEncodedUnicode()` and `PyUnicode_AsDecodedUnicode()` are deprecated now. Use the generic codec based API instead.

## 9.5 Deprecated Build Options

The `--with-system-ffi` configure flag is now on by default on non-macOS UNIX platforms. It may be disabled by using `--without-system-ffi`, but using the flag is deprecated and will not be accepted in Python 3.7. macOS is unaffected by this change. Note that many OS distributors already use the `--with-system-ffi` flag when building their system Python.

# 10 Removed

## 10.1 API and Feature Removals

- Unknown escapes consisting of `'\'` and an ASCII letter in regular expressions will now cause an error. In replacement templates for `re.sub()` they are still allowed, but deprecated. The `re.LOCALE` flag can now only be used with binary patterns.
- `inspect.getmoduleinfo()` was removed (was deprecated since CPython 3.3). `inspect.getmodulename()` should be used for obtaining the module name for a given path. (Contributed by Yuri Selivanov in [issue 13248](#).)

- `traceback.Ignore` class and `traceback.usage`, `traceback.modname`, `traceback.fullmodname`, `traceback.find_lines_from_code`, `traceback.find_lines`, `traceback.find_strings`, `traceback.find_executable_lines` methods were removed from the `traceback` module. They were undocumented methods deprecated since Python 3.2 and equivalent functionality is available from private methods.
- The `tk_menuBar()` and `tk_bindForTraversal()` dummy methods in `tkinter` widget classes were removed (corresponding Tk commands were obsolete since Tk 4.0).
- The `open()` method of the `zipfile.ZipFile` class no longer supports the 'U' mode (was deprecated since Python 3.4). Use `io.TextIOWrapper` for reading compressed text files in universal newlines mode.
- The undocumented `IN`, `CDROM`, `DLFCN`, `TYPES`, `CDIO`, and `STROPTS` modules have been removed. They had been available in the platform specific `Lib/plat-*/` directories, but were chronically out of date, inconsistently available across platforms, and unmaintained. The script that created these modules is still available in the source distribution at [Tools/scripts/h2py.py](#).
- The deprecated `asyncio.fifo` class has been removed.

## 11 Porting to Python 3.6

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

### 11.1 Changes in 'python' Command Behavior

- The output of a special Python build with defined `COUNT_ALLOCS`, `SHOW_ALLOC_COUNT` or `SHOW_TRACK_COUNT` macros is now off by default. It can be re-enabled using the `-X showalloccount` option. It now outputs to `stderr` instead of `stdout`. (Contributed by Serhiy Storchaka in [issue 23034](#).)

### 11.2 Changes in the Python API

- `open()` will no longer allow combining the 'U' mode flag with '+'. (Contributed by Jeff Balogh and John O'Connor in [issue 2091](#).)
- `sqlite3` no longer implicitly commits an open transaction before DDL statements.
- On Linux, `os.urandom()` now blocks until the system `urandom` entropy pool is initialized to increase the security.
- When `importlib.abc.Loader.exec_module()` is defined, `importlib.abc.Loader.create_module()` must also be defined.
- `PyErr_SetImportError()` now sets `TypeError` when its `msg` argument is not set. Previously only `NULL` was returned.
- The format of the `co_lnotab` attribute of code objects changed to support a negative line number delta. By default, Python does not emit bytecode with a negative line number delta. Functions using `frame.f_lineno`, `PyFrame_GetLineNumber()` or `PyCode_Addr2Line()` are not affected. Functions directly decoding `co_lnotab` should be updated to use a signed 8-bit integer type for the line number delta, but this is only required to support applications using a negative line number delta. See [Objects/lnotab\\_notes.txt](#) for the `co_lnotab` format and how to decode it, and see the [PEP 511](#) for the rationale.
- The functions in the `compileall` module now return booleans instead of 1 or 0 to represent success or failure, respectively. Thanks to booleans being a subclass of integers, this should only be an issue if you were doing identity checks for 1 or 0. See [issue 25768](#).

- Reading the `port` attribute of `urllib.parse.urlsplit()` and `urlparse()` results now raises `ValueError` for out-of-range values, rather than returning `None`. See [issue 20059](#).
- The `imp` module now raises a `DeprecationWarning` instead of `PendingDeprecationWarning`.
- The following modules have had missing APIs added to their `__all__` attributes to match the documented APIs: `calendar`, `cgi`, `csv`, `ElementTree`, `enum`, `fileinput`, `ftplib`, `logging`, `mailbox`, `mimetypes`, `optparse`, `plistlib`, `smtpd`, `subprocess`, `tarfile`, `threading` and `wave`. This means they will export new symbols when `import *` is used. (Contributed by Joel Taddei and Jacek Kołodziej in [issue 23883](#).)
- When performing a relative import, if `__package__` does not compare equal to `__spec__.parent` then `ImportWarning` is raised. (Contributed by Brett Cannon in [issue 25791](#).)
- When a relative import is performed and no parent package is known, then `ImportError` will be raised. Previously, `SystemError` could be raised. (Contributed by Brett Cannon in [issue 18018](#).)
- Servers based on the `socketserver` module, including those defined in `http.server`, `xmlrpc.server` and `wsgiref.simple_server`, now only catch exceptions derived from `Exception`. Therefore if a request handler raises an exception like `SystemExit` or `KeyboardInterrupt`, `handle_error()` is no longer called, and the exception will stop a single-threaded server. (Contributed by Martin Panter in [issue 23430](#).)
- `spwd.getspnam()` now raises a `PermissionError` instead of `KeyError` if the user doesn't have privileges.
- The `socket.socket.close()` method now raises an exception if an error (e.g. `EBADF`) was reported by the underlying system call. (Contributed by Martin Panter in [issue 26685](#).)
- The `decode_data` argument for the `smtpd.SMTPChannel` and `smtpd.SMTPServer` constructors is now `False` by default. This means that the argument passed to `process_message()` is now a bytes object by default, and `process_message()` will be passed keyword arguments. Code that has already been updated in accordance with the deprecation warning generated by 3.5 will not be affected.
- All optional arguments of the `dump()`, `dumps()`, `load()` and `loads()` functions and `JSONEncoder` and `JSONDecoder` class constructors in the `json` module are now keyword-only. (Contributed by Serhiy Storchaka in [issue 18726](#).)
- Subclasses of `type` which don't override `type.__new__` may no longer use the one-argument form to get the type of an object.
- As part of [PEP 487](#), the handling of keyword arguments passed to `type` (other than the metaclass hint, `metaclass`) is now consistently delegated to `object.__init_subclass__()`. This means that `type.__new__()` and `type.__init__()` both now accept arbitrary keyword arguments, but `object.__init_subclass__()` (which is called from `type.__new__()`) will reject them by default. Custom metaclasses accepting additional keyword arguments will need to adjust their calls to `type.__new__()` (whether direct or via `super`) accordingly.
- In `distutils.command.sdist.sdist`, the `default_format` attribute has been removed and is no longer honored. Instead, the gzipped tarfile format is the default on all platforms and no platform-specific selection is made. In environments where distributions are built on Windows and zip distributions are required, configure the project with a `setup.cfg` file containing the following:

```
[sdist]
formats=zip
```

This behavior has also been backported to earlier Python versions by Setuptools 26.0.0.

- In the `urllib.request` module and the `http.client.HTTPConnection.request()` method, if no `Content-Length` header field has been specified and the request body is a file object, it is now sent with HTTP 1.1 chunked encoding. If a file object has to be sent to a HTTP 1.0 server, the `Content-Length` value now has to be specified by the caller. (Contributed by Demian Brecht and Rolf Krahle with tweaks from Martin Panter in [issue 12319](#).)

- The `DictReader` now returns rows of type `OrderedDict` . (Contributed by Steve Holden in [issue 27842](#).)
- The `crypt.METHOD_CRYPT` will no longer be added to `crypt.methods` if unsupported by the platform. (Contributed by Victor Stinner in [issue 25287](#).)
- The `verbose` and `rename` arguments for `namedtuple()` are now keyword-only. (Contributed by Raymond Hettinger in [issue 25628](#).)
- On Linux, `ctypes.util.find_library()` now looks in `LD_LIBRARY_PATH` for shared libraries. (Contributed by Vinay Sajip in [issue 9998](#).)
- The `imaplib.IMAP4` class now handles flags containing the `']'` character in messages sent from the server to improve real-world compatibility. (Contributed by Lita Cho in [issue 21815](#).)
- The `mmap.write()` function now returns the number of bytes written like other write methods. (Contributed by Jakub Stasiak in [issue 26335](#).)
- The `pkgutil.iter_modules()` and `pkgutil.walk_packages()` functions now return `ModuleInfo` named tuples. (Contributed by Ramchandra Apte in [issue 17211](#).)
- `re.sub()` now raises an error for invalid numerical group references in replacement templates even if the pattern is not found in the string. The error message for invalid group references now includes the group index and the position of the reference. (Contributed by SilentGhost, Serhiy Storchaka in [issue 25953](#).)
- `zipfile.ZipFile` will now raise `NotImplementedError` for unrecognized compression values. Previously a plain `RuntimeError` was raised. Additionally, calling `ZipFile` methods on a closed `ZipFile` or calling the `write()` method on a `ZipFile` created with mode `'r'` will raise a `ValueError` . Previously, a `RuntimeError` was raised in those scenarios.
- when custom metaclasses are combined with zero-argument `super()` or direct references from methods to the implicit `__class__` closure variable, the implicit `__classcell__` namespace entry must now be passed up to `type.__new__` for initialisation. Failing to do so will result in a `DeprecationWarning` in 3.6 and a `RuntimeWarning` in the future.

### 11.3 Changes in the C API

- The `PyMem_Malloc()` allocator family now uses the `pymalloc` allocator rather than the system `malloc()` . Applications calling `PyMem_Malloc()` without holding the GIL can now crash. Set the `PYTHONMALLOC` environment variable to `debug` to validate the usage of memory allocators in your application. See [issue 26249](#).
- `Py_Exit()` (and the main interpreter) now override the exit status with 120 if flushing buffered data failed. See [issue 5319](#).

### 11.4 CPython bytecode changes

There have been several major changes to the bytecode in Python 3.6.

- The Python interpreter now uses a 16-bit wordcode instead of bytecode. (Contributed by Demur Rumed with input and reviews from Serhiy Storchaka and Victor Stinner in [issue 26647](#) and [issue 28050](#).)
- The new `FORMAT_VALUE` and `BUILD_STRING` opcodes as part of the *formatted string literal* implementation. (Contributed by Eric Smith in [issue 25483](#) and Serhiy Storchaka in [issue 27078](#).)
- The new `BUILD_CONST_KEY_MAP` opcode to optimize the creation of dictionaries with constant keys. (Contributed by Serhiy Storchaka in [issue 27140](#).)
- The function call opcodes have been heavily reworked for better performance and simpler implementation. The `MAKE_FUNCTION` , `CALL_FUNCTION` , `CALL_FUNCTION_KW` and `BUILD_MAP_UNPACK_WITH_CALL` opcodes have been modified, the new `CALL_FUNCTION_EX` and `BUILD_TUPLE_UNPACK_WITH_CALL` have been added, and `CALL_FUNCTION_VAR` ,

`CALL_FUNCTION_VAR_KW` and `MAKE_CLOSURE` opcodes have been removed. (Contributed by Demur Rumed in [issue 27095](#), and Serhiy Storchaka in [issue 27213](#), [issue 28257](#).)

- The new `SETUP_ANNOTATIONS` and `STORE_ANNOTATION` opcodes have been added to support the new variable annotation syntax. (Contributed by Ivan Levkivskyi in [issue 27985](#).)

## Index

### E

environment variable

PYTHONHOME, 4

PYTHONLEGACYWINDOWSFSENCODING,  
9

PYTHONLEGACYWINDOWSIOENCODING,  
9

PYTHONMALLOC, 10, 29

### P

Python Enhancement Proposals

PEP 468, 9

PEP 484, 5

PEP 487, 7, 28

PEP 492, 6, 24

PEP 494, 3

PEP 495, 8

PEP 498, 5

PEP 506, 12

PEP 511, 27

PEP 515, 5, 6

PEP 519, 8

PEP 520, 9

PEP 523, 10

PEP 524, 4, 17

PEP 525, 6

PEP 526, 5, 21

PEP 528, 9

PEP 529, 9

PEP 530, 6

PEP 628, 13, 17

PEP 7, 24

PYTHONHOME, 4

PYTHONLEGACYWINDOWSFSENCODING, 9

PYTHONLEGACYWINDOWSIOENCODING, 9

PYTHONMALLOC, 10, 29