What's New in Python

Release 3.8.0

A. M. Kuchling

December 10, 2019

Python Software Foundation
Email: docs@python.org

Contents

Editor Raymond Hettinger

This article explains the new features in Python 3.8, compared to 3.7. For full details, see the changelog.

Python 3.8 was released on October 14th, 2019.

1 Summary – Release highlights

2 New Features

2.1 Assignment expressions

There is new syntax `:=` that assigns values to variables as part of a larger expression. It is affectionately known as "the walrus operator" due to its resemblance to the eyes and tusks of a walrus.

In this example, the assignment expression helps avoid calling `len()` twice:

```
if (n := len(a)) > 10:
    print(f"List is too long ({n} elements, expected <= 10)")
```

A similar benefit arises during regular expression matching where match objects are needed twice, once to test whether a match occurred and another to extract a subgroup:

```
discount = 0.0
if (mo := re.search(r'(\d+)% discount', advertisement)):
    discount = float(mo.group(1)) / 100.0
```

The operator is also useful with while-loops that compute a value to test loop termination and then need that same value again in the body of the loop:

```
# Loop over fixed length blocks
while (block := f.read(256)) != '':
    process(block)
```

Another motivating use case arises in list comprehensions where a value computed in a filtering condition is also needed in the expression body:

```
[clean_name.title() for name in names
 if (clean_name := normalize('NFC', name)) in allowed_names]
```

Try to limit use of the walrus operator to clean cases that reduce complexity and improve readability.

See **PEP 572** for a full description.

(Contributed by Emily Morehouse in bpo-35224.)

2.2 Positional-only parameters

There is a new function parameter syntax `/` to indicate that some function parameters must be specified positionally and cannot be used as keyword arguments. This is the same notation shown by `help()` for C functions annotated with Larry Hastings' Argument Clinic tool.

In the following example, parameters *a* and *b* are positional-only, while *c* or *d* can be positional or keyword, and *e* or *f* are required to be keywords:

```
def f(a, b, /, c, d, *, e, f):
    print(a, b, c, d, e, f)
```

The following is a valid call:

```
f(10, 20, 30, d=40, e=50, f=60)
```

However, these are invalid calls:

```
f(10, b=20, c=30, d=40, e=50, f=60)   # b cannot be a keyword argument
f(10, 20, 30, 40, 50, f=60)           # e must be a keyword argument
```

One use case for this notation is that it allows pure Python functions to fully emulate behaviors of existing C coded functions. For example, the built-in `pow()` function does not accept keyword arguments:

```
def pow(x, y, z=None, /):
    "Emulate the built in pow() function"
    r = x ** y
    return r if z is None else r%z
```

Another use case is to preclude keyword arguments when the parameter name is not helpful. For example, the builtin `len()` function has the signature `len(obj, /)`. This precludes awkward calls such as:

```
len(obj='hello')  # The "obj" keyword argument impairs readability
```

A further benefit of marking a parameter as positional-only is that it allows the parameter name to be changed in the future without risk of breaking client code. For example, in the `statistics` module, the parameter name *dist* may be changed in the future. This was made possible with the following function specification:

```
def quantiles(dist, /, *, n=4, method='exclusive')
    ...
```

Since the parameters to the left of `/` are not exposed as possible keywords, the parameters names remain available for use in `**kwargs`:

```
>>> def f(a, b, /, **kwargs):
...     print(a, b, kwargs)
...
>>> f(10, 20, a=1, b=2, c=3)         # a and b are used in two ways
10 20 {'a': 1, 'b': 2, 'c': 3}
```

This greatly simplifies the implementation of functions and methods that need to accept arbitrary keyword arguments. For example, here is an excerpt from code in the `collections` module:

```
class Counter(dict):

    def __init__(self, iterable=None, /, **kwds):
        # Note "iterable" is a possible keyword argument
```

See **PEP 570** for a full description.

(Contributed by Pablo Galindo in bpo-36540.)

2.3 Parallel filesystem cache for compiled bytecode files

The new `PYTHONPYCACHEPREFIX` setting (also available as `-X pycache_prefix`) configures the implicit bytecode cache to use a separate parallel filesystem tree, rather than the default `__pycache__` subdirectories within each source directory.

The location of the cache is reported in `sys.pycache_prefix` (`None` indicates the default location in `__pycache__` subdirectories).

(Contributed by Carl Meyer in bpo-33499.)

2.4 Debug build uses the same ABI as release build

Python now uses the same ABI whether it's built in release or debug mode. On Unix, when Python is built in debug mode, it is now possible to load C extensions built in release mode and C extensions built using the stable ABI.

Release builds and debug builds are now ABI compatible: defining the `Py_DEBUG` macro no longer implies the `Py_TRACE_REFS` macro, which introduces the only ABI incompatibility. The `Py_TRACE_REFS` macro, which adds the `sys.getobjects()` function and the `PYTHONDUMPREFS` environment variable, can be set using the new `./configure --with-trace-refs` build option. (Contributed by Victor Stinner in bpo-36465.)

On Unix, C extensions are no longer linked to libpython except on Android and Cygwin. It is now possible for a statically linked Python to load a C extension built using a shared library Python. (Contributed by Victor Stinner in bpo-21536.)

On Unix, when Python is built in debug mode, import now also looks for C extensions compiled in release mode and for C extensions compiled with the stable ABI. (Contributed by Victor Stinner in bpo-36722.)

To embed Python into an application, a new `--embed` option must be passed to `python3-config --libs --embed` to get `-lpython3.8` (link the application to libpython). To support both 3.8 and older, try `python3-config --libs --embed` first and fallback to `python3-config --libs` (without `--embed`) if the previous command fails.

Add a pkg-config `python-3.8-embed` module to embed Python into an application: `pkg-config python-3.8-embed --libs` includes `-lpython3.8`. To support both 3.8 and older, try `pkg-config python-X.Y-embed --libs` first and fallback to `pkg-config python-X.Y --libs` (without `--embed`) if the previous command fails (replace `X.Y` with the Python version).

On the other hand, `pkg-config python3.8 --libs` no longer contains `-lpython3.8`. C extensions must not be linked to libpython (except on Android and Cygwin, whose cases are handled by the script); this change is backward incompatible on purpose. (Contributed by Victor Stinner in bpo-36721.)

2.5 f-strings support = for self-documenting expressions and debugging

Added an = specifier to f-strings. An f-string such as `f'{expr=}'` will expand to the text of the expression, an equal sign, then the representation of the evaluated expression. For example:

```
>>> user = 'eric_idle'
>>> member_since = date(1975, 7, 31)
>>> f'{user=} {member_since=}'
"user='eric_idle' member_since=datetime.date(1975, 7, 31)"
```

The usual f-string format specifiers allow more control over how the result of the expression is displayed:

```
>>> delta = date.today() - member_since
>>> f'{user=!s}  {delta.days=:,d}'
'user=eric_idle  delta.days=16,075'
```

The = specifier will display the whole expression so that calculations can be shown:

```
>>> print(f'{theta=}  {cos(radians(theta))=:.3f}')
theta=30  cos(radians(theta))=0.866
```

(Contributed by Eric V. Smith and Larry Hastings in bpo-36817.)

2.6 PEP 578: Python Runtime Audit Hooks

The PEP adds an Audit Hook and Verified Open Hook. Both are available from Python and native code, allowing applications and frameworks written in pure Python code to take advantage of extra notifications, while also allowing embedders or system administrators to deploy builds of Python where auditing is always enabled.

See **PEP 578** for full details.

2.7 PEP 587: Python Initialization Configuration

The **PEP 587** adds a new C API to configure the Python Initialization providing finer control on the whole configuration and better error reporting.

New structures:

- `PyConfig`
- `PyPreConfig`
- `PyStatus`
- `PyWideStringList`

New functions:

- `PyConfig_Clear()`
- `PyConfig_InitIsolatedConfig()`
- `PyConfig_InitPythonConfig()`
- `PyConfig_Read()`
- `PyConfig_SetArgv()`
- `PyConfig_SetBytesArgv()`
- `PyConfig_SetBytesString()`
- `PyConfig_SetString()`
- `PyPreConfig_InitIsolatedConfig()`
- `PyPreConfig_InitPythonConfig()`
- `PyStatus_Error()`
- `PyStatus_Exception()`
- `PyStatus_Exit()`
- `PyStatus_IsError()`
- `PyStatus_IsExit()`
- `PyStatus_NoMemory()`
- `PyStatus_Ok()`
- `PyWideStringList_Append()`
- `PyWideStringList_Insert()`
- `Py_BytesMain()`
- `Py_ExitStatusException()`

- `Py_InitializeFromConfig()`
- `Py_PreInitialize()`
- `Py_PreInitializeFromArgs()`
- `Py_PreInitializeFromBytesArgs()`
- `Py_RunMain()`

This PEP also adds `_PyRuntimeState.preconfig` (`PyPreConfig` type) and `PyInterpreterState.config` (`PyConfig` type) fields to these internal structures. `PyInterpreterState.config` becomes the new reference configuration, replacing global configuration variables and other private variables.

See Python Initialization Configuration for the documentation.

See **PEP 587** for a full description.

(Contributed by Victor Stinner in bpo-36763.)

2.8 Vectorcall: a fast calling protocol for CPython

The "vectorcall" protocol is added to the Python/C API. It is meant to formalize existing optimizations which were already done for various classes. Any extension type implementing a callable can use this protocol.

This is currently provisional. The aim is to make it fully public in Python 3.9.

See **PEP 590** for a full description.

(Contributed by Jeroen Demeyer and Mark Shannon in bpo-36974.)

2.9 Pickle protocol 5 with out-of-band data buffers

When `pickle` is used to transfer large data between Python processes in order to take advantage of multi-core or multi-machine processing, it is important to optimize the transfer by reducing memory copies, and possibly by applying custom techniques such as data-dependent compression.

The `pickle` protocol 5 introduces support for out-of-band buffers where **PEP 3118**-compatible data can be transmitted separately from the main pickle stream, at the discretion of the communication layer.

See **PEP 574** for a full description.

(Contributed by Antoine Pitrou in bpo-36785.)

3 Other Language Changes

- A `continue` statement was illegal in the `finally` clause due to a problem with the implementation. In Python 3.8 this restriction was lifted. (Contributed by Serhiy Storchaka in bpo-32489.)
- The `bool`, `int`, and `fractions.Fraction` types now have an `as_integer_ratio()` method like that found in `float` and `decimal.Decimal`. This minor API extension makes it possible to write `numerator, denominator = x.as_integer_ratio()` and have it work across multiple numeric types. (Contributed by Lisa Roach in bpo-33073 and Raymond Hettinger in bpo-37819.)
- Constructors of `int`, `float` and `complex` will now use the `__index__()` special method, if available and the corresponding method `__int__()`, `__float__()` or `__complex__()` is not available. (Contributed by Serhiy Storchaka in bpo-20092.)
- Added support of `\N{name}` escapes in `regular expressions`:

```
>>> notice = 'Copyright © 2019'
>>> copyright_year_pattern = re.compile(r'\N{copyright sign}\s*(\d{4})')
>>> int(copyright_year_pattern.search(notice).group(1))
2019
```

(Contributed by Jonathan Eunice and Serhiy Storchaka in bpo-30688.)

- Dict and dictviews are now iterable in reversed insertion order using `reversed()`. (Contributed by Rémi Lapeyre in bpo-33462.)
- The syntax allowed for keyword names in function calls was further restricted. In particular, `f((keyword)=arg)` is no longer allowed. It was never intended to permit more than a bare name on the left-hand side of a keyword argument assignment term. (Contributed by Benjamin Peterson in bpo-34641.)
- Generalized iterable unpacking in `yield` and `return` statements no longer requires enclosing parentheses. This brings the *yield* and *return* syntax into better agreement with normal assignment syntax:

```
>>> def parse(family):
        lastname, *members = family.split()
        return lastname.upper(), *members

>>> parse('simpsons homer marge bart lisa sally')
('SIMPSONS', 'homer', 'marge', 'bart', 'lisa', 'sally')
```

(Contributed by David Cuthbert and Jordan Chapman in bpo-32117.)

- When a comma is missed in code such as `[(10, 20) (30, 40)]`, the compiler displays a `SyntaxWarning` with a helpful suggestion. This improves on just having a `TypeError` indicating that the first tuple was not callable. (Contributed by Serhiy Storchaka in bpo-15248.)
- Arithmetic operations between subclasses of `datetime.date` or `datetime.datetime` and `datetime.timedelta` objects now return an instance of the subclass, rather than the base class. This also affects the return type of operations whose implementation (directly or indirectly) uses `datetime.timedelta` arithmetic, such as `astimezone()`. (Contributed by Paul Ganssle in bpo-32417.)
- When the Python interpreter is interrupted by Ctrl-C (SIGINT) and the resulting `KeyboardInterrupt` exception is not caught, the Python process now exits via a SIGINT signal or with the correct exit code such that the calling process can detect that it died due to a Ctrl-C. Shells on POSIX and Windows use this to properly terminate scripts in interactive sessions. (Contributed by Google via Gregory P. Smith in bpo-1054041.)
- Some advanced styles of programming require updating the `types.CodeType` object for an existing function. Since code objects are immutable, a new code object needs to be created, one that is modeled on the existing code object. With 19 parameters, this was somewhat tedious. Now, the new `replace()` method makes it possible to create a clone with a few altered parameters.

  Here's an example that alters the `statistics.mean()` function to prevent the *data* parameter from being used as a keyword argument:

```
>>> from statistics import mean
>>> mean(data=[10, 20, 90])
40
>>> mean.__code__ = mean.__code__.replace(co_posonlyargcount=1)
>>> mean(data=[10, 20, 90])
Traceback (most recent call last):
  ...
TypeError: mean() got some positional-only arguments passed as keyword arguments:
↪'data'
```

(Contributed by Victor Stinner in bpo-37032.)

- For integers, the three-argument form of the `pow()` function now permits the exponent to be negative in the case where the base is relatively prime to the modulus. It then computes a modular inverse to the base when the exponent is `-1`, and a suitable power of that inverse for other negative exponents. For example, to compute the modular multiplicative inverse of 38 modulo 137, write:

```
>>> pow(38, -1, 137)
119
>>> 119 * 38 % 137
1
```

  Modular inverses arise in the solution of linear Diophantine equations. For example, to find integer solutions for `4258x + 147y = 369`, first rewrite as `4258x ≡ 369 (mod 147)` then solve:

```
>>> x = 369 * pow(4258, -1, 147) % 147
>>> y = (4258 * x - 369) // -147
>>> 4258 * x + 147 * y
369
```

  (Contributed by Mark Dickinson in bpo-36027.)

- Dict comprehensions have been synced-up with dict literals so that the key is computed first and the value second:

```
>>> # Dict comprehension
>>> cast = {input('role? '): input('actor? ') for i in range(2)}
role? King Arthur
actor? Chapman
role? Black Knight
actor? Cleese

>>> # Dict literal
>>> cast = {input('role? '): input('actor? ')}
role? Sir Robin
actor? Eric Idle
```

  The guaranteed execution order is helpful with assignment expressions because variables assigned in the key expression will be available in the value expression:

```
>>> names = ['Martin von Löwis', 'Łukasz Langa', 'Walter Dörwald']
>>> {(n := normalize('NFC', name)).casefold() : n for name in names}
{'martin von löwis': 'Martin von Löwis',
 'łukasz langa': 'Łukasz Langa',
 'walter dörwald': 'Walter Dörwald'}
```

  (Contributed by Jörn Heissler in bpo-35224.)

- The `object.__reduce__()` method can now return a tuple from two to six elements long. Formerly, five was the limit. The new, optional sixth element is a callable with a `(obj, state)` signature. This allows the direct control over the state-updating behavior of a specific object. If not *None*, this callable will have priority over the object's `__setstate__()` method. (Contributed by Pierre Glaser and Olivier Grisel in bpo-35900.)

4 New Modules

- The new `importlib.metadata` module provides (provisional) support for reading metadata from third-party packages. For example, it can extract an installed package's version number, list of entry points, and more:

```
>>> # Note following example requires that the popular "requests"
>>> # package has been installed.
>>>
>>> from importlib.metadata import version, requires, files
>>> version('requests')
'2.22.0'
>>> list(requires('requests'))
['chardet (<3.1.0,>=3.0.2)']
>>> list(files('requests'))[:5]
[PackagePath('requests-2.22.0.dist-info/INSTALLER'),
 PackagePath('requests-2.22.0.dist-info/LICENSE'),
 PackagePath('requests-2.22.0.dist-info/METADATA'),
 PackagePath('requests-2.22.0.dist-info/RECORD'),
 PackagePath('requests-2.22.0.dist-info/WHEEL')]
```

(Contributed by Barry Warsaw and Jason R. Coombs in bpo-34632.)

5 Improved Modules

5.1 ast

AST nodes now have `end_lineno` and `end_col_offset` attributes, which give the precise location of the end of the node. (This only applies to nodes that have `lineno` and `col_offset` attributes.)

New function `ast.get_source_segment()` returns the source code for a specific AST node.

(Contributed by Ivan Levkivskyi in bpo-33416.)

The `ast.parse()` function has some new flags:

- `type_comments=True` causes it to return the text of **PEP 484** and **PEP 526** type comments associated with certain AST nodes;
- `mode='func_type'` can be used to parse **PEP 484** "signature type comments" (returned for function definition AST nodes);
- `feature_version=(3, N)` allows specifying an earlier Python 3 version. For example, `feature_version=(3, 4)` will treat `async` and `await` as non-reserved words.

(Contributed by Guido van Rossum in bpo-35766.)

5.2 asyncio

`asyncio.run()` has graduated from the provisional to stable API. This function can be used to execute a coroutine and return the result while automatically managing the event loop. For example:

```
import asyncio

async def main():
    await asyncio.sleep(0)
    return 42

asyncio.run(main())
```

This is *roughly* equivalent to:

```
import asyncio

async def main():
    await asyncio.sleep(0)
    return 42

loop = asyncio.new_event_loop()
asyncio.set_event_loop(loop)
try:
    loop.run_until_complete(main())
finally:
    asyncio.set_event_loop(None)
    loop.close()
```

The actual implementation is significantly more complex. Thus, `asyncio.run()` should be the preferred way of running asyncio programs.

(Contributed by Yury Selivanov in bpo-32314.)

Running `python -m asyncio` launches a natively async REPL. This allows rapid experimentation with code that has a top-level `await`. There is no longer a need to directly call `asyncio.run()` which would spawn a new event loop on every invocation:

```
$ python -m asyncio
asyncio REPL 3.8.0
Use "await" directly instead of "asyncio.run()".
Type "help", "copyright", "credits" or "license" for more information.
>>> import asyncio
>>> await asyncio.sleep(10, result='hello')
hello
```

(Contributed by Yury Selivanov in bpo-37028.)

The exception `asyncio.CancelledError` now inherits from `BaseException` rather than `Exception`. (Contributed by Yury Selivanov in bpo-32528.)

On Windows, the default event loop is now `ProactorEventLoop`. (Contributed by Victor Stinner in bpo-34687.)

`ProactorEventLoop` now also supports UDP. (Contributed by Adam Meily and Andrew Svetlov in bpo-29883.)

`ProactorEventLoop` can now be interrupted by `KeyboardInterrupt` ("CTRL+C"). (Contributed by Vladimir Matveev in bpo-23057.)

Added `asyncio.Task.get_coro()` for getting the wrapped coroutine within an `asyncio.Task`. (Contributed by Alex Grönholm in bpo-36999.)

Asyncio tasks can now be named, either by passing the `name` keyword argument to `asyncio.create_task()` or the `create_task()` event loop method, or by calling the `set_name()` method on the task object. The task name is visible in the `repr()` output of `asyncio.Task` and can also be retrieved using the `get_name()` method. (Contributed by Alex Grönholm in bpo-34270.)

Added support for Happy Eyeballs to `asyncio.loop.create_connection()`. To specify the behavior, two new parameters have been added: *happy_eyeballs_delay* and *interleave*. The Happy Eyeballs algorithm improves responsiveness in applications that support IPv4 and IPv6 by attempting to simultaneously connect using both. (Contributed by twisteroid ambassador in bpo-33530.)

5.3 builtins

The `compile()` built-in has been improved to accept the `ast.PyCF_ALLOW_TOP_LEVEL_AWAIT` flag. With this new flag passed, `compile()` will allow top-level `await`, `async for` and `async with` constructs that are usually considered invalid syntax. Asynchronous code object marked with the `CO_COROUTINE` flag may then be returned. (Contributed by Matthias Bussonnier in bpo-34616)

5.4 collections

The `_asdict()` method for `collections.namedtuple()` now returns a `dict` instead of a `collections.OrderedDict`. This works because regular dicts have guaranteed ordering since Python 3.7. If the extra features of `OrderedDict` are required, the suggested remediation is to cast the result to the desired type: `OrderedDict(nt._asdict())`. (Contributed by Raymond Hettinger in bpo-35864.)

5.5 cProfile

The `cProfile.Profile` class can now be used as a context manager. Profile a block of code by running:

```
import cProfile

with cProfile.Profile() as profiler:
      # code to be profiled
      ...
```

(Contributed by Scott Sanderson in bpo-29235.)

5.6 csv

The `csv.DictReader` now returns instances of `dict` instead of a `collections.OrderedDict`. The tool is now faster and uses less memory while still preserving the field order. (Contributed by Michael Seek in bpo-34003.)

5.7 curses

Added a new variable holding structured version information for the underlying ncurses library: `ncurses_version`. (Contributed by Serhiy Storchaka in bpo-31680.)

5.8 ctypes

On Windows, `CDLL` and subclasses now accept a *winmode* parameter to specify flags for the underlying `LoadLibraryEx` call. The default flags are set to only load DLL dependencies from trusted locations, including the path where the DLL is stored (if a full or partial path is used to load the initial DLL) and paths added by `add_dll_directory()`. (Contributed by Steve Dower in bpo-36085.)

5.9 datetime

Added new alternate constructors `datetime.date.fromisocalendar()` and `datetime.datetime.fromisocalendar()`, which construct `date` and `datetime` objects respectively from ISO year, week number, and weekday; these are the inverse of each class's `isocalendar` method. (Contributed by Paul Ganssle in bpo-36004.)

5.10 functools

`functools.lru_cache()` can now be used as a straight decorator rather than as a function returning a decorator. So both of these are now supported:

```
@lru_cache
def f(x):
    ...

@lru_cache(maxsize=256)
def f(x):
    ...
```

(Contributed by Raymond Hettinger in bpo-36772.)

Added a new `functools.cached_property()` decorator, for computed properties cached for the life of the instance.

```
import functools
import statistics

class Dataset:
   def __init__(self, sequence_of_numbers):
      self.data = sequence_of_numbers

   @functools.cached_property
   def variance(self):
      return statistics.variance(self.data)
```

(Contributed by Carl Meyer in bpo-21145)

Added a new `functools.singledispatchmethod()` decorator that converts methods into generic functions using single dispatch:

```
from functools import singledispatchmethod
from contextlib import suppress

class TaskManager:

    def __init__(self, tasks):
        self.tasks = list(tasks)

    @singledispatchmethod
    def discard(self, value):
        with suppress(ValueError):
            self.tasks.remove(value)

    @discard.register(list)
    def _(self, tasks):
        targets = set(tasks)
        self.tasks = [x for x in self.tasks if x not in targets]
```

(Contributed by Ethan Smith in bpo-32380)

5.11 gc

`get_objects()` can now receive an optional *generation* parameter indicating a generation to get objects from. (Contributed by Pablo Galindo in bpo-36016.)

5.12 gettext

Added `pgettext()` and its variants. (Contributed by Franz Glasner, Éric Araujo, and Cheryl Sabella in bpo-2504.)

5.13 gzip

Added the *mtime* parameter to `gzip.compress()` for reproducible output. (Contributed by Guo Ci Teo in bpo-34898.)

A `BadGzipFile` exception is now raised instead of `OSError` for certain types of invalid or corrupt gzip files. (Contributed by Filip Gruszczyński, Michele Orrù, and Zackery Spytz in bpo-6584.)

5.14 IDLE and idlelib

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

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

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

OS native encoding is now used for converting between Python strings and Tcl objects. This allows IDLE to work with emoji and other non-BMP characters. These characters can be displayed or copied and pasted to or from the clipboard. Converting strings from Tcl to Python and back now never fails. (Many people worked on this for eight years but the problem was finally solved by Serhiy Storchaka in bpo-13153.)

The changes above have been backported to 3.7 maintenance releases.

5.15 inspect

The `inspect.getdoc()` function can now find docstrings for `__slots__` if that attribute is a `dict` where the values are docstrings. This provides documentation options similar to what we already have for `property()`, `classmethod()`, and `staticmethod()`:

```
class AudioClip:
    __slots__ = {'bit_rate': 'expressed in kilohertz to one decimal place',
                 'duration': 'in seconds, rounded up to an integer'}
    def __init__(self, bit_rate, duration):
        self.bit_rate = round(bit_rate / 1000.0, 1)
        self.duration = ceil(duration)
```

(Contributed by Raymond Hettinger in bpo-36326.)

5.16 io

In development mode (`-X env`) and in debug build, the `io.IOBase` finalizer now logs the exception if the `close()` method fails. The exception is ignored silently by default in release build. (Contributed by Victor Stinner in bpo-18748.)

5.17 itertools

The `itertools.accumulate()` function added an option *initial* keyword argument to specify an initial value:

```
>>> from itertools import accumulate
>>> list(accumulate([10, 5, 30, 15], initial=1000))
[1000, 1010, 1015, 1045, 1060]
```

(Contributed by Lisa Roach in bpo-34659.)

5.18 json.tool

Add option `--json-lines` to parse every input line as a separate JSON object. (Contributed by Weipeng Hong in bpo-31553.)

5.19 logging

Added a *force* keyword argument to `logging.basicConfig()` When set to true, any existing handlers attached to the root logger are removed and closed before carrying out the configuration specified by the other arguments.

This solves a long-standing problem. Once a logger or *basicConfig()* had been called, subsequent calls to *basicConfig()* were silently ignored. This made it difficult to update, experiment with, or teach the various logging configuration options using the interactive prompt or a Jupyter notebook.

(Suggested by Raymond Hettinger, implemented by Dong-hee Na, and reviewed by Vinay Sajip in bpo-33897.)

5.20 math

Added new function `math.dist()` for computing Euclidean distance between two points. (Contributed by Raymond Hettinger in bpo-33089.)

Expanded the `math.hypot()` function to handle multiple dimensions. Formerly, it only supported the 2-D case. (Contributed by Raymond Hettinger in bpo-33089.)

Added new function, `math.prod()`, as analogous function to `sum()` that returns the product of a 'start' value (default: 1) times an iterable of numbers:

```
>>> prior = 0.8
>>> likelihoods = [0.625, 0.84, 0.30]
>>> math.prod(likelihoods, start=prior)
0.126
```

(Contributed by Pablo Galindo in bpo-35606.)

Added two new combinatoric functions `math.perm()` and `math.comb()`:

```
>>> math.perm(10, 3)     # Permutations of 10 things taken 3 at a time
720
>>> math.comb(10, 3)     # Combinations of 10 things taken 3 at a time
120
```

(Contributed by Yash Aggarwal, Keller Fuchs, Serhiy Storchaka, and Raymond Hettinger in bpo-37128, bpo-37178, and bpo-35431.)

Added a new function `math.isqrt()` for computing accurate integer square roots without conversion to floating point. The new function supports arbitrarily large integers. It is faster than `floor(sqrt(n))` but slower than `math.sqrt()`:

```
>>> r = 650320427
>>> s = r ** 2
>>> isqrt(s - 1)         # correct
650320426
>>> floor(sqrt(s - 1))   # incorrect
650320427
```

(Contributed by Mark Dickinson in bpo-36887.)

The function `math.factorial()` no longer accepts arguments that are not int-like. (Contributed by Pablo Galindo in bpo-33083.)

5.21 mmap

The `mmap.mmap` class now has an `madvise()` method to access the `madvise()` system call. (Contributed by Zackery Spytz in bpo-32941.)

5.22 multiprocessing

Added new `multiprocessing.shared_memory` module. (Contributed by Davin Potts in bpo-35813.)

On macOS, the *spawn* start method is now used by default. (Contributed by Victor Stinner in bpo-33725.)

5.23 os

Added new function `add_dll_directory()` on Windows for providing additional search paths for native dependencies when importing extension modules or loading DLLs using `ctypes`. (Contributed by Steve Dower in bpo-36085.)

A new `os.memfd_create()` function was added to wrap the `memfd_create()` syscall. (Contributed by Zackery Spytz and Christian Heimes in bpo-26836.)

On Windows, much of the manual logic for handling reparse points (including symlinks and directory junctions) has been delegated to the operating system. Specifically, `os.stat()` will now traverse anything supported by the operating system, while `os.lstat()` will only open reparse points that identify as "name surrogates" while others are opened as for `os.stat()`. In all cases, `stat_result.st_mode` will only have `S_IFLNK` set for symbolic links and not other kinds of reparse points. To identify other kinds of reparse point, check the new `stat_result.st_reparse_tag` attribute.

On Windows, `os.readlink()` is now able to read directory junctions. Note that `islink()` will return `False` for directory junctions, and so code that checks `islink` first will continue to treat junctions as directories, while code that handles errors from `os.readlink()` may now treat junctions as links.

(Contributed by Steve Dower in bpo-37834.)

5.24 os.path

`os.path` functions that return a boolean result like `exists()`, `lexists()`, `isdir()`, `isfile()`, `islink()`, and `ismount()` now return `False` instead of raising `ValueError` or its subclasses `UnicodeEncodeError` and `UnicodeDecodeError` for paths that contain characters or bytes unrepresentable at the OS level. (Contributed by Serhiy Storchaka in bpo-33721.)

`expanduser()` on Windows now prefers the `USERPROFILE` environment variable and does not use `HOME`, which is not normally set for regular user accounts. (Contributed by Anthony Sottile in bpo-36264.)

`isdir()` on Windows no longer returns `True` for a link to a non-existent directory.

`realpath()` on Windows now resolves reparse points, including symlinks and directory junctions.

(Contributed by Steve Dower in bpo-37834.)

5.25 pathlib

`pathlib.Path` methods that return a boolean result like `exists()`, `is_dir()`, `is_file()`, `is_mount()`, `is_symlink()`, `is_block_device()`, `is_char_device()`, `is_fifo()`, `is_socket()` now return `False` instead of raising `ValueError` or its subclass `UnicodeEncodeError` for paths that contain characters unrepresentable at the OS level. (Contributed by Serhiy Storchaka in bpo-33721.)

Added `pathlib.Path.link_to()` which creates a hard link pointing to a path. (Contributed by Joannah Nanjekye in bpo-26978)

5.26 pickle

`pickle` extensions subclassing the C-optimized `Pickler` can now override the pickling logic of functions and classes by defining the special `reducer_override()` method. (Contributed by Pierre Glaser and Olivier Grisel in bpo-35900.)

5.27 plistlib

Added new `plistlib.UID` and enabled support for reading and writing NSKeyedArchiver-encoded binary plists. (Contributed by Jon Janzen in bpo-26707.)

5.28 pprint

The `pprint` module added a *sort_dicts* parameter to several functions. By default, those functions continue to sort dictionaries before rendering or printing. However, if *sort_dicts* is set to false, the dictionaries retain the order that keys were inserted. This can be useful for comparison to JSON inputs during debugging.

In addition, there is a convenience new function, `pprint.pp()` that is like `pprint.pprint()` but with *sort_dicts* defaulting to `False`:

```
>>> from pprint import pprint, pp
>>> d = dict(source='input.txt', operation='filter', destination='output.txt')
>>> pp(d, width=40)                  # Original order
{'source': 'input.txt',
 'operation': 'filter',
 'destination': 'output.txt'}
>>> pprint(d, width=40)              # Keys sorted alphabetically
{'destination': 'output.txt',
 'operation': 'filter',
 'source': 'input.txt'}
```

(Contributed by Rémi Lapeyre in bpo-30670.)

5.29 py_compile

`py_compile.compile()` now supports silent mode. (Contributed by Joannah Nanjekye in bpo-22640.)

5.30 shlex

The new `shlex.join()` function acts as the inverse of `shlex.split()`. (Contributed by Bo Bayles in bpo-32102.)

5.31 shutil

`shutil.copytree()` now accepts a new `dirs_exist_ok` keyword argument. (Contributed by Josh Bronson in bpo-20849.)

`shutil.make_archive()` now defaults to the modern pax (POSIX.1-2001) format for new archives to improve portability and standards conformance, inherited from the corresponding change to the `tarfile` module. (Contributed by C.A.M. Gerlach in bpo-30661.)

`shutil.rmtree()` on Windows now removes directory junctions without recursively removing their contents first. (Contributed by Steve Dower in bpo-37834.)

5.32 socket

Added `create_server()` and `has_dualstack_ipv6()` convenience functions to automate the necessary tasks usually involved when creating a server socket, including accepting both IPv4 and IPv6 connections on the same socket. (Contributed by Giampaolo Rodolà in bpo-17561.)

The `socket.if_nameindex()`, `socket.if_nametoindex()`, and `socket.if_indextoname()` functions have been implemented on Windows. (Contributed by Zackery Spytz in bpo-37007.)

5.33 ssl

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

5.34 statistics

Added `statistics.fmean()` as a faster, floating point variant of `statistics.mean()`. (Contributed by Raymond Hettinger and Steven D'Aprano in bpo-35904.)

Added `statistics.geometric_mean()` (Contributed by Raymond Hettinger in bpo-27181.)

Added `statistics.multimode()` that returns a list of the most common values. (Contributed by Raymond Hettinger in bpo-35892.)

Added `statistics.quantiles()` that divides data or a distribution in to equiprobable intervals (e.g. quartiles, deciles, or percentiles). (Contributed by Raymond Hettinger in bpo-36546.)

Added `statistics.NormalDist`, a tool for creating and manipulating normal distributions of a random variable. (Contributed by Raymond Hettinger in bpo-36018.)

```
>>> temperature_feb = NormalDist.from_samples([4, 12, -3, 2, 7, 14])
>>> temperature_feb.mean
6.0
>>> temperature_feb.stdev
6.356099432828281

>>> temperature_feb.cdf(3)           # Chance of being under 3 degrees
0.3184678262814532
>>> # Relative chance of being 7 degrees versus 10 degrees
>>> temperature_feb.pdf(7) / temperature_feb.pdf(10)
1.2039930378537762

>>> el_niño = NormalDist(4, 2.5)
>>> temperature_feb += el_niño       # Add in a climate effect
>>> temperature_feb
NormalDist(mu=10.0, sigma=6.830080526611674)

>>> temperature_feb * (9/5) + 32     # Convert to Fahrenheit
NormalDist(mu=50.0, sigma=12.294144947901014)
>>> temperature_feb.samples(3)       # Generate random samples
[7.672102882379219, 12.000027119750287, 4.647488369766392]
```

5.35 sys

Add new `sys.unraisablehook()` function which can be overridden to control how "unraisable exceptions" are handled. It is called when an exception has occurred but there is no way for Python to handle it. For example, when a destructor raises an exception or during garbage collection (`gc.collect()`). (Contributed by Victor Stinner in bpo-36829.)

5.36 tarfile

The `tarfile` module now defaults to the modern pax (POSIX.1-2001) format for new archives, instead of the previous GNU-specific one. This improves cross-platform portability with a consistent encoding (UTF-8) in a standardized and extensible format, and offers several other benefits. (Contributed by C.A.M. Gerlach in bpo-36268.)

5.37 threading

Add a new `threading.excepthook()` function which handles uncaught `threading.Thread.run()` exception. It can be overridden to control how uncaught `threading.Thread.run()` exceptions are handled. (Contributed by Victor Stinner in bpo-1230540.)

Add a new `threading.get_native_id()` function and a `native_id` attribute to the `threading.Thread` class. These return the native integral Thread ID of the current thread assigned by the kernel. This feature is only available on certain platforms, see `get_native_id` for more information. (Contributed by Jake Tesler in bpo-36084.)

5.38 tokenize

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

5.39 tkinter

Added methods `selection_from()`, `selection_present()`, `selection_range()` and `selection_to()` in the `tkinter.Spinbox` class. (Contributed by Juliette Monsel in bpo-34829.)

Added method `moveto()` in the `tkinter.Canvas` class. (Contributed by Juliette Monsel in bpo-23831.)

The `tkinter.PhotoImage` class now has `transparency_get()` and `transparency_set()` methods. (Contributed by Zackery Spytz in bpo-25451.)

5.40 time

Added new clock `CLOCK_UPTIME_RAW` for macOS 10.12. (Contributed by Joannah Nanjekye in bpo-35702.)

5.41 typing

The `typing` module incorporates several new features:

- A dictionary type with per-key types. See **PEP 589** and `typing.TypedDict`. TypedDict uses only string keys. By default, every key is required to be present. Specify "total=False" to allow keys to be optional:

```
class Location(TypedDict, total=False):
    lat_long: tuple
    grid_square: str
    xy_coordinate: tuple
```

- Literal types. See **PEP 586** and `typing.Literal`. Literal types indicate that a parameter or return value is constrained to one or more specific literal values:

```
def get_status(port: int) -> Literal['connected', 'disconnected']:
    ...
```

- "Final" variables, functions, methods and classes. See **PEP 591**, `typing.Final` and `typing.final()`. The final qualifier instructs a static type checker to restrict subclassing, overriding, or reassignment:

```
pi: Final[float] = 3.1415926536
```

- Protocol definitions. See **PEP 544**, `typing.Protocol` and `typing.runtime_checkable()`. Simple ABCs like `typing.SupportsInt` are now `Protocol` subclasses.
- New protocol class `typing.SupportsIndex`.
- New functions `typing.get_origin()` and `typing.get_args()`.

5.42 unicodedata

The `unicodedata` module has been upgraded to use the Unicode 12.1.0 release.

New function `is_normalized()` can be used to verify a string is in a specific normal form, often much faster than by actually normalizing the string. (Contributed by Max Belanger, David Euresti, and Greg Price in bpo-32285 and bpo-37966).

5.43 unittest

Added `AsyncMock` to support an asynchronous version of `Mock`. Appropriate new assert functions for testing have been added as well. (Contributed by Lisa Roach in bpo-26467).

Added `addModuleCleanup()` and `addClassCleanup()` to unittest to support cleanups for `setUpModule()` and `setUpClass()`. (Contributed by Lisa Roach in bpo-24412.)

Several mock assert functions now also print a list of actual calls upon failure. (Contributed by Petter Strandmark in bpo-35047.)

`unittest` module gained support for coroutines to be used as test cases with `unittest.IsolatedAsyncioTestCase`. (Contributed by Andrew Svetlov in bpo-32972.)

Example:

```
import unittest


class TestRequest(unittest.IsolatedAsyncioTestCase):

    async def asyncSetUp(self):
        self.connection = await AsyncConnection()

    async def test_get(self):
        response = await self.connection.get("https://example.com")
        self.assertEqual(response.status_code, 200)

    async def asyncTearDown(self):
        await self.connection.close()


if __name__ == "__main__":
    unittest.main()
```

5.44 venv

`venv` now includes an `Activate.ps1` script on all platforms for activating virtual environments under PowerShell Core 6.1. (Contributed by Brett Cannon in bpo-32718.)

5.45 weakref

The proxy objects returned by `weakref.proxy()` now support the matrix multiplication operators `@` and `@=` in addition to the other numeric operators. (Contributed by Mark Dickinson in bpo-36669.)

5.46 xml

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

The `.find*()` methods in the `xml.etree.ElementTree` module support wildcard searches like `{*}tag` which ignores the namespace and `{namespace}*` which returns all tags in the given namespace. (Contributed by Stefan Behnel in bpo-28238.)

The `xml.etree.ElementTree` module provides a new function `–xml.etree.ElementTree.canonicalize()` that implements C14N 2.0. (Contributed by Stefan Behnel in bpo-13611.)

The target object of `xml.etree.ElementTree.XMLParser` can receive namespace declaration events through the new callback methods `start_ns()` and `end_ns()`. Additionally, the `xml.etree.ElementTree.TreeBuilder` target can be configured to process events about comments and processing instructions to include them in the generated tree. (Contributed by Stefan Behnel in bpo-36676 and bpo-36673.)

5.47 xmlrpc

`xmlrpc.client.ServerProxy` now supports an optional *headers* keyword argument for a sequence of HTTP headers to be sent with each request. Among other things, this makes it possible to upgrade from default basic authentication to faster session authentication. (Contributed by Cédric Krier in bpo-35153.)

6 Optimizations

- The `subprocess` module can now use the `os.posix_spawn()` function in some cases for better performance. Currently, it is only used on macOS and Linux (using glibc 2.24 or newer) if all these conditions are met:
  - *close_fds* is false;
  - *preexec_fn*, *pass_fds*, *cwd* and *start_new_session* parameters are not set;
  - the *executable* path contains a directory.

  (Contributed by Joannah Nanjekye and Victor Stinner in bpo-35537.)
- `shutil.copyfile()`, `shutil.copy()`, `shutil.copy2()`, `shutil.copytree()` and `shutil.move()` use platform-specific "fast-copy" syscalls on Linux and macOS in order to copy the file more efficiently. "fast-copy" means that the copying operation occurs within the kernel, avoiding the use of userspace buffers in Python as in "`outfd.write(infd.read())`". On Windows `shutil.copyfile()` uses a bigger default buffer size (1 MiB instead of 16 KiB) and a `memoryview()`-based variant of `shutil.copyfileobj()` is used. The speedup for copying a 512 MiB file within the same partition is about +26% on Linux, +50% on macOS and +40% on Windows. Also, much less CPU cycles are consumed. See shutil-platform-dependent-efficient-copy-operations section. (Contributed by Giampaolo Rodolà in bpo-33671.)
- `shutil.copytree()` uses `os.scandir()` function and all copy functions depending from it use cached `os.stat()` values. The speedup for copying a directory with 8000 files is around +9% on Linux, +20% on Windows and +30% on a Windows SMB share. Also the number of `os.stat()` syscalls is reduced by 38% making `shutil.copytree()` especially faster on network filesystems. (Contributed by Giampaolo Rodolà in bpo-33695.)
- The default protocol in the `pickle` module is now Protocol 4, first introduced in Python 3.4. It offers better performance and smaller size compared to Protocol 3 available since Python 3.0.
- Removed one `Py_ssize_t` member from `PyGC_Head`. All GC tracked objects (e.g. tuple, list, dict) size is reduced 4 or 8 bytes. (Contributed by Inada Naoki in bpo-33597.)
- `uuid.UUID` now uses `__slots__` to reduce its memory footprint. (Contributed by Wouter Bolsterlee and Tal Einat in bpo-30977)
- Improved performance of `operator.itemgetter()` by 33%. Optimized argument handling and added a fast path for the common case of a single non-negative integer index into a tuple (which is the typical use case in the standard library). (Contributed by Raymond Hettinger in bpo-35664.)

- Sped-up field lookups in `collections.namedtuple()`. They are now more than two times faster, making them the fastest form of instance variable lookup in Python. (Contributed by Raymond Hettinger, Pablo Galindo, and Joe Jevnik, Serhiy Storchaka in bpo-32492.)
- The `list` constructor does not overallocate the internal item buffer if the input iterable has a known length (the input implements `__len__`). This makes the created list 12% smaller on average. (Contributed by Raymond Hettinger and Pablo Galindo in bpo-33234.)
- Doubled the speed of class variable writes. When a non-dunder attribute was updated, there was an unnecessary call to update slots. (Contributed by Stefan Behnel, Pablo Galindo Salgado, Raymond Hettinger, Neil Schemenauer, and Serhiy Storchaka in bpo-36012.)
- Reduced an overhead of converting arguments passed to many builtin functions and methods. This sped up calling some simple builtin functions and methods up to 20–50%. (Contributed by Serhiy Storchaka in bpo-23867, bpo-35582 and bpo-36127.)
- `LOAD_GLOBAL` instruction now uses new "per opcode cache" mechanism. It is about 40% faster now. (Contributed by Yury Selivanov and Inada Naoki in bpo-26219.)

7 Build and C API Changes

- Default `sys.abiflags` became an empty string: the `m` flag for pymalloc became useless (builds with and without pymalloc are ABI compatible) and so has been removed. (Contributed by Victor Stinner in bpo-36707.)

  Example of changes:

  - Only `python3.8` program is installed, `python3.8m` program is gone.
  - Only `python3.8-config` script is installed, `python3.8m-config` script is gone.
  - The `m` flag has been removed from the suffix of dynamic library filenames: extension modules in the standard library as well as those produced and installed by third-party packages, like those downloaded from PyPI. On Linux, for example, the Python 3.7 suffix `.cpython-37m-x86_64-linux-gnu.so` became `.cpython-38-x86_64-linux-gnu.so` in Python 3.8.
- The header files have been reorganized to better separate the different kinds of APIs:
  - `Include/*.h` should be the portable public stable C API.
  - `Include/cpython/*.h` should be the unstable C API specific to CPython; public API, with some private API prefixed by `_Py` or `_PY`.
  - `Include/internal/*.h` is the private internal C API very specific to CPython. This API comes with no backward compatibility warranty and should not be used outside CPython. It is only exposed for very specific needs like debuggers and profiles which has to access to CPython internals without calling functions. This API is now installed by `make install`.

  (Contributed by Victor Stinner in bpo-35134 and bpo-35081, work initiated by Eric Snow in Python 3.7.)
- Some macros have been converted to static inline functions: parameter types and return type are well defined, they don't have issues specific to macros, variables have a local scopes. Examples:
  - `Py_INCREF()`, `Py_DECREF()`
  - `Py_XINCREF()`, `Py_XDECREF()`
  - `PyObject_INIT()`, `PyObject_INIT_VAR()`
  - Private functions: `_PyObject_GC_TRACK()`, `_PyObject_GC_UNTRACK()`, `_Py_Dealloc()`

  (Contributed by Victor Stinner in bpo-35059.)

- The `PyByteArray_Init()` and `PyByteArray_Fini()` functions have been removed. They did nothing since Python 2.7.4 and Python 3.2.0, were excluded from the limited API (stable ABI), and were not documented. (Contributed by Victor Stinner in bpo-35713.)
- The result of `PyExceptionClass_Name()` is now of type `const char *` rather of `char *`. (Contributed by Serhiy Storchaka in bpo-33818.)
- The duality of `Modules/Setup.dist` and `Modules/Setup` has been removed. Previously, when updating the CPython source tree, one had to manually copy `Modules/Setup.dist` (inside the source tree) to `Modules/Setup` (inside the build tree) in order to reflect any changes upstream. This was of a small benefit to packagers at the expense of a frequent annoyance to developers following CPython development, as forgetting to copy the file could produce build failures.

  Now the build system always reads from `Modules/Setup` inside the source tree. People who want to customize that file are encouraged to maintain their changes in a git fork of CPython or as patch files, as they would do for any other change to the source tree.

  (Contributed by Antoine Pitrou in bpo-32430.)
- Functions that convert Python number to C integer like `PyLong_AsLong()` and argument parsing functions like `PyArg_ParseTuple()` with integer converting format units like `'i'` will now use the `__index__()` special method instead of `__int__()`, if available. The deprecation warning will be emitted for objects with the `__int__()` method but without the `__index__()` method (like `Decimal` and `Fraction`). `PyNumber_Check()` will now return `1` for objects implementing `__index__()`. `PyNumber_Long()`, `PyNumber_Float()` and `PyFloat_AsDouble()` also now use the `__index__()` method if available. (Contributed by Serhiy Storchaka in bpo-36048 and bpo-20092.)
- Heap-allocated type objects will now increase their reference count in `PyObject_Init()` (and its parallel macro `PyObject_INIT`) instead of in `PyType_GenericAlloc()`. Types that modify instance allocation or deallocation may need to be adjusted. (Contributed by Eddie Elizondo in bpo-35810.)
- The new function `PyCode_NewWithPosOnlyArgs()` allows to create code objects like `PyCode_New()`, but with an extra *posonlyargcount* parameter for indicating the number of positional-only arguments. (Contributed by Pablo Galindo in bpo-37221.)
- `Py_SetPath()` now sets `sys.executable` to the program full path (`Py_GetProgramFullPath()`) rather than to the program name (`Py_GetProgramName()`). (Contributed by Victor Stinner in bpo-38234.)

8 Deprecated

- The distutils `bdist_wininst` command is now deprecated, use `bdist_wheel` (wheel packages) instead. (Contributed by Victor Stinner in bpo-37481.)
- Deprecated methods `getchildren()` and `getiterator()` in the `ElementTree` module now emit a `DeprecationWarning` instead of `PendingDeprecationWarning`. They will be removed in Python 3.9. (Contributed by Serhiy Storchaka in bpo-29209.)
- Passing an object that is not an instance of `concurrent.futures.ThreadPoolExecutor` to `loop.set_default_executor()` is deprecated and will be prohibited in Python 3.9. (Contributed by Elvis Pranskevichus in bpo-34075.)
- The `__getitem__()` methods of `xml.dom.pulldom.DOMEventStream`, `wsgiref.util.FileWrapper` and `fileinput.FileInput` have been deprecated.

  Implementations of these methods have been ignoring their *index* parameter, and returning the next item instead. (Contributed by Berker Peksag in bpo-9372.)

- The `typing.NamedTuple` class has deprecated the `_field_types` attribute in favor of the `__annotations__` attribute which has the same information. (Contributed by Raymond Hettinger in bpo-36320.)

- `ast` classes `Num`, `Str`, `Bytes`, `NameConstant` and `Ellipsis` are considered deprecated and will be removed in future Python versions. `Constant` should be used instead. (Contributed by Serhiy Storchaka in bpo-32892.)

- `ast.NodeVisitor` methods `visit_Num()`, `visit_Str()`, `visit_Bytes()`, `visit_NameConstant()` and `visit_Ellipsis()` are deprecated now and will not be called in future Python versions. Add the `visit_Constant()` method to handle all constant nodes. (Contributed by Serhiy Storchaka in bpo-36917.)

- The `asyncio.coroutine()` decorator is deprecated and will be removed in version 3.10. Instead of `@asyncio.coroutine`, use `async def` instead. (Contributed by Andrew Svetlov in bpo-36921.)

- In `asyncio`, the explicit passing of a *loop* argument has been deprecated and will be removed in version 3.10 for the following: `asyncio.sleep()`, `asyncio.gather()`, `asyncio.shield()`, `asyncio.wait_for()`, `asyncio.wait()`, `asyncio.as_completed()`, `asyncio.Task`, `asyncio.Lock`, `asyncio.Event`, `asyncio.Condition`, `asyncio.Semaphore`, `asyncio.BoundedSemaphore`, `asyncio.Queue`, `asyncio.create_subprocess_exec()`, and `asyncio.create_subprocess_shell()`.

- The explicit passing of coroutine objects to `asyncio.wait()` has been deprecated and will be removed in version 3.11. (Contributed by Yury Selivanov in bpo-34790.)

- The following functions and methods are deprecated in the `gettext` module: `lgettext()`, `ldgettext()`, `lngettext()` and `ldngettext()`. They return encoded bytes, and it's possible that you will get unexpected Unicode-related exceptions if there are encoding problems with the translated strings. It's much better to use alternatives which return Unicode strings in Python 3. These functions have been broken for a long time.

  Function `bind_textdomain_codeset()`, methods `output_charset()` and `set_output_charset()`, and the *codeset* parameter of functions `translation()` and `install()` are also deprecated, since they are only used for for the `l*gettext()` functions. (Contributed by Serhiy Storchaka in bpo-33710.)

- The `isAlive()` method of `threading.Thread` has been deprecated. (Contributed by Dong-hee Na in bpo-35283.)

- Many builtin and extension functions that take integer arguments will now emit a deprecation warning for `Decimal`s, `Fraction`s and any other objects that can be converted to integers only with a loss (e.g. that have the `__int__()` method but do not have the `__index__()` method). In future version they will be errors. (Contributed by Serhiy Storchaka in bpo-36048.)

- Deprecated passing the following arguments as keyword arguments:

  - *func* in `functools.partialmethod()`, `weakref.finalize()`, `profile.Profile.runcall()`, `cProfile.Profile.runcall()`, `bdb.Bdb.runcall()`, `trace.Trace.runfunc()` and `curses.wrapper()`.

  - *function* in `unittest.TestCase.addCleanup()`.

  - *fn* in the `submit()` method of `concurrent.futures.ThreadPoolExecutor` and `concurrent.futures.ProcessPoolExecutor`.

  - *callback* in `contextlib.ExitStack.callback()`, `contextlib.AsyncExitStack.callback()` and `contextlib.AsyncExitStack.push_async_callback()`.

  - *c* and *typeid* in the `create()` method of `multiprocessing.managers.Server` and `multiprocessing.managers.SharedMemoryServer`.

  - *obj* in `weakref.finalize()`.

In future releases of Python, they will be positional-only. (Contributed by Serhiy Storchaka in bpo-36492.)

9 API and Feature Removals

The following features and APIs have been removed from Python 3.8:

- Starting with Python 3.3, importing ABCs from `collections` was deprecated, and importing should be done from `collections.abc`. Being able to import from collections was marked for removal in 3.8, but has been delayed to 3.9. (See bpo-36952.)
- The `macpath` module, deprecated in Python 3.7, has been removed. (Contributed by Victor Stinner in bpo-35471.)
- The function `platform.popen()` has been removed, after having been deprecated since Python 3.3: use `os.popen()` instead. (Contributed by Victor Stinner in bpo-35345.)
- The function `time.clock()` has been removed, after having been deprecated since Python 3.3: use `time.perf_counter()` or `time.process_time()` instead, depending on your requirements, to have well-defined behavior. (Contributed by Matthias Bussonnier in bpo-36895.)
- The `pyvenv` script has been removed in favor of `python3.8 -m venv` to help eliminate confusion as to what Python interpreter the `pyvenv` script is tied to. (Contributed by Brett Cannon in bpo-25427.)
- `parse_qs`, `parse_qsl`, and `escape` are removed from the `cgi` module. They are deprecated in Python 3.2 or older. They should be imported from the `urllib.parse` and `html` modules instead.
- `filemode` function is removed from the `tarfile` module. It is not documented and deprecated since Python 3.3.
- The `XMLParser` constructor no longer accepts the *html* argument. It never had an effect and was deprecated in Python 3.4. All other parameters are now keyword-only. (Contributed by Serhiy Storchaka in bpo-29209.)
- Removed the `doctype()` method of `XMLParser`. (Contributed by Serhiy Storchaka in bpo-29209.)
- "unicode_internal" codec is removed. (Contributed by Inada Naoki in bpo-36297.)
- The `Cache` and `Statement` objects of the `sqlite3` module are not exposed to the user. (Contributed by Aviv Palivoda in bpo-30262.)
- The `bufsize` keyword argument of `fileinput.input()` and `fileinput.FileInput()` which was ignored and deprecated since Python 3.6 has been removed. bpo-36952 (Contributed by Matthias Bussonnier.)
- The functions `sys.set_coroutine_wrapper()` and `sys.get_coroutine_wrapper()` deprecated in Python 3.7 have been removed; bpo-36933 (Contributed by Matthias Bussonnier.)

10 Porting to Python 3.8

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

10.1 Changes in Python behavior

- Yield expressions (both `yield` and `yield from` clauses) are now disallowed in comprehensions and generator expressions (aside from the iterable expression in the leftmost `for` clause). (Contributed by Serhiy Storchaka in bpo-10544.)

- The compiler now produces a `SyntaxWarning` when identity checks (`is` and `is not`) are used with certain types of literals (e.g. strings, numbers). These can often work by accident in CPython, but are not guaranteed by the language spec. The warning advises users to use equality tests (== and `!=`) instead. (Contributed by Serhiy Storchaka in bpo-34850.)
- The CPython interpreter can swallow exceptions in some circumstances. In Python 3.8 this happens in fewer cases. In particular, exceptions raised when getting the attribute from the type dictionary are no longer ignored. (Contributed by Serhiy Storchaka in bpo-35459.)
- Removed `__str__` implementations from builtin types `bool`, `int`, `float`, `complex` and few classes from the standard library. They now inherit `__str__()` from `object`. As result, defining the `__repr__()` method in the subclass of these classes will affect their string representation. (Contributed by Serhiy Storchaka in bpo-36793.)
- On AIX, `sys.platform` doesn't contain the major version anymore. It is always `'aix'`, instead of `'aix3'` .. `'aix7'`. Since older Python versions include the version number, so it is recommended to always use `sys.platform.startswith('aix')`. (Contributed by M. Felt in bpo-36588.)
- `PyEval_AcquireLock()` and `PyEval_AcquireThread()` now terminate the current thread if called while the interpreter is finalizing, making them consistent with `PyEval_RestoreThread()`, `Py_END_ALLOW_THREADS()`, and `PyGILState_Ensure()`. If this behavior is not desired, guard the call by checking `_Py_IsFinalizing()` or `sys.is_finalizing()`. (Contributed by Joannah Nanjekye in bpo-36475.)

10.2 Changes in the Python API

- The `os.getcwdb()` function now uses the UTF-8 encoding on Windows, rather than the ANSI code page: see **PEP 529** for the rationale. The function is no longer deprecated on Windows. (Contributed by Victor Stinner in bpo-37412.)
- `subprocess.Popen` can now use `os.posix_spawn()` in some cases for better performance. On Windows Subsystem for Linux and QEMU User Emulation, the `Popen` constructor using `os.posix_spawn()` no longer raises an exception on errors like "missing program". Instead the child process fails with a non-zero `returncode`. (Contributed by Joannah Nanjekye and Victor Stinner in bpo-35537.)
- The *preexec_fn* argument of * `subprocess.Popen` is no longer compatible with subinterpreters. The use of the parameter in a subinterpreter now raises `RuntimeError`. (Contributed by Eric Snow in bpo-34651, modified by Christian Heimes in bpo-37951.)
- The `imap.IMAP4.logout()` method no longer silently ignores arbitrary exceptions. (Contributed by Victor Stinner in bpo-36348.)
- The function `platform.popen()` has been removed, after having been deprecated since Python 3.3: use `os.popen()` instead. (Contributed by Victor Stinner in bpo-35345.)
- The `statistics.mode()` function no longer raises an exception when given multimodal data. Instead, it returns the first mode encountered in the input data. (Contributed by Raymond Hettinger in bpo-35892.)
- The `selection()` method of the `tkinter.ttk.Treeview` class no longer takes arguments. Using it with arguments for changing the selection was deprecated in Python 3.6. Use specialized methods like `selection_set()` for changing the selection. (Contributed by Serhiy Storchaka in bpo-31508.)
- The `writexml()`, `toxml()` and `toprettyxml()` methods of `xml.dom.minidom`, and the `write()` method of `xml.etree`, now preserve the attribute order specified by the user. (Contributed by Diego Rojas and Raymond Hettinger in bpo-34160.)
- A `dbm.dumb` database opened with flags `'r'` is now read-only. `dbm.dumb.open()` with flags `'r'` and `'w'` no longer creates a database if it does not exist. (Contributed by Serhiy Storchaka in bpo-32749.)

- The `doctype()` method defined in a subclass of `XMLParser` will no longer be called and will emit a `RuntimeWarning` instead of a `DeprecationWarning`. Define the `doctype()` method on a target for handling an XML doctype declaration. (Contributed by Serhiy Storchaka in bpo-29209.)
- A `RuntimeError` is now raised when the custom metaclass doesn't provide the `__classcell__` entry in the namespace passed to `type.__new__`. A `DeprecationWarning` was emitted in Python 3.6–3.7. (Contributed by Serhiy Storchaka in bpo-23722.)
- The `cProfile.Profile` class can now be used as a context manager. (Contributed by Scott Sanderson in bpo-29235.)
- `shutil.copyfile()`, `shutil.copy()`, `shutil.copy2()`, `shutil.copytree()` and `shutil.move()` use platform-specific "fast-copy" syscalls (see shutil-platform-dependent-efficient-copy-operations section).
- `shutil.copyfile()` default buffer size on Windows was changed from 16 KiB to 1 MiB.
- The `PyGC_Head` struct has changed completely. All code that touched the struct member should be rewritten. (See bpo-33597.)
- The `PyInterpreterState` struct has been moved into the "internal" header files (specifically Include/internal/pycore_pystate.h). An opaque `PyInterpreterState` is still available as part of the public API (and stable ABI). The docs indicate that none of the struct's fields are public, so we hope no one has been using them. However, if you do rely on one or more of those private fields and have no alternative then please open a BPO issue. We'll work on helping you adjust (possibly including adding accessor functions to the public API). (See bpo-35886.)
- The `mmap.flush()` method now returns `None` on success and raises an exception on error under all platforms. Previously, its behavior was platform-dependent: a nonzero value was returned on success; zero was returned on error under Windows. A zero value was returned on success; an exception was raised on error under Unix. (Contributed by Berker Peksag in bpo-2122.)
- `xml.dom.minidom` and `xml.sax` modules no longer process external entities by default. (Contributed by Christian Heimes in bpo-17239.)
- Deleting a key from a read-only `dbm` database (`dbm.dumb`, `dbm.gnu` or `dbm.ndbm`) raises `error` (`dbm.dumb.error`, `dbm.gnu.error` or `dbm.ndbm.error`) instead of `KeyError`. (Contributed by Xiang Zhang in bpo-33106.)
- `expanduser()` on Windows now prefers the `USERPROFILE` environment variable and does not use `HOME`, which is not normally set for regular user accounts. (Contributed by Anthony Sottile in bpo-36264.)
- The exception `asyncio.CancelledError` now inherits from `BaseException` rather than `Exception`. (Contributed by Yury Selivanov in bpo-32528.)
- The function `asyncio.wait_for()` now correctly waits for cancellation when using an instance of `asyncio.Task`. Previously, upon reaching *timeout*, it was cancelled and immediately returned. (Contributed by Elvis Pranskevichus in bpo-32751.)
- The function `asyncio.BaseTransport.get_extra_info()` now returns a safe to use socket object when 'socket' is passed to the *name* parameter. (Contributed by Yury Selivanov in bpo-37027.)
- `asyncio.BufferedProtocol` has graduated to the stable API.
- DLL dependencies for extension modules and DLLs loaded with `ctypes` on Windows are now resolved more securely. Only the system paths, the directory containing the DLL or PYD file, and directories added with `add_dll_directory()` are searched for load-time dependencies. Specifically, `PATH` and the current working directory are no longer used, and modifications to these will no longer have any effect on normal DLL resolution. If your application relies on these mechanisms, you should check for `add_dll_directory()` and if it exists, use it to add your DLLs directory while loading your library. Note that Windows 7 users will need to ensure that

Windows Update KB2533623 has been installed (this is also verified by the installer). (Contributed by Steve Dower in bpo-36085.)

- The header files and functions related to pgen have been removed after its replacement by a pure Python implementation. (Contributed by Pablo Galindo in bpo-36623.)
- `types.CodeType` has a new parameter in the second position of the constructor (*posonlyargcount*) to support positional-only arguments defined in **PEP 570**. The first argument (*argcount*) now represents the total number of positional arguments (including positional-only arguments). The new `replace()` method of `types.CodeType` can be used to make the code future-proof.

10.3 Changes in the C API

- The `PyCompilerFlags` structure got a new *cf_feature_version* field. It should be initialized to `PY_MINOR_VERSION`. The field is ignored by default, and is used if and only if `PyCF_ONLY_AST` flag is set in *cf_flags*. (Contributed by Guido van Rossum in bpo-35766.)
- The `PyEval_ReInitThreads()` function has been removed from the C API. It should not be called explicitly: use `PyOS_AfterFork_Child()` instead. (Contributed by Victor Stinner in bpo-36728.)
- On Unix, C extensions are no longer linked to libpython except on Android and Cygwin. When Python is embedded, `libpython` must not be loaded with `RTLD_LOCAL`, but `RTLD_GLOBAL` instead. Previously, using `RTLD_LOCAL`, it was already not possible to load C extensions which were not linked to `libpython`, like C extensions of the standard library built by the `*shared*` section of `Modules/Setup`. (Contributed by Victor Stinner in bpo-21536.)
- Use of `#` variants of formats in parsing or building value (e.g. `PyArg_ParseTuple()`, `Py_BuildValue()`, `PyObject_CallFunction()`, etc.) without `PY_SSIZE_T_CLEAN` defined raises `DeprecationWarning` now. It will be removed in 3.10 or 4.0. Read arg-parsing for detail. (Contributed by Inada Naoki in bpo-36381.)
- Instances of heap-allocated types (such as those created with `PyType_FromSpec()`) hold a reference to their type object. Increasing the reference count of these type objects has been moved from `PyType_GenericAlloc()` to the more low-level functions, `PyObject_Init()` and `PyObject_INIT()`. This makes types created through `PyType_FromSpec()` behave like other classes in managed code.

  Statically allocated types are not affected.

  For the vast majority of cases, there should be no side effect. However, types that manually increase the reference count after allocating an instance (perhaps to work around the bug) may now become immortal. To avoid this, these classes need to call Py_DECREF on the type object during instance deallocation.

  To correctly port these types into 3.8, please apply the following changes:

  – Remove `Py_INCREF` on the type object after allocating an instance - if any. This may happen after calling `PyObject_New()`, `PyObject_NewVar()`, `PyObject_GC_New()`, `PyObject_GC_NewVar()`, or any other custom allocator that uses `PyObject_Init()` or `PyObject_INIT()`.

  Example:

```
static foo_struct *
foo_new(PyObject *type) {
    foo_struct *foo = PyObject_GC_New(foo_struct, (PyTypeObject *) type);
    if (foo == NULL)
        return NULL;
#if PY_VERSION_HEX < 0x03080000
```

(continues on next page)

(continued from previous page)

```
    // Workaround for Python issue 35810; no longer necessary in Python 3.8
    PY_INCREF(type)
#endif
    return foo;
}
```

– Ensure that all custom `tp_dealloc` functions of heap-allocated types decrease the type's reference count.

Example:

```
static void
foo_dealloc(foo_struct *instance) {
    PyObject *type = Py_TYPE(instance);
    PyObject_GC_Del(instance);
#if PY_VERSION_HEX >= 0x03080000
    // This was not needed before Python 3.8 (Python issue 35810)
    Py_DECREF(type);
#endif
}
```

(Contributed by Eddie Elizondo in bpo-35810.)

- The `Py_DEPRECATED()` macro has been implemented for MSVC. The macro now must be placed before the symbol name.

  Example:

```
Py_DEPRECATED(3.8) PyAPI_FUNC(int) Py_OldFunction(void);
```

  (Contributed by Zackery Spytz in bpo-33407.)

- The interpreter does not pretend to support binary compatibility of extension types across feature releases, anymore. A `PyTypeObject` exported by a third-party extension module is supposed to have all the slots expected in the current Python version, including `tp_finalize` (`Py_TPFLAGS_HAVE_FINALIZE` is not checked anymore before reading `tp_finalize`).

  (Contributed by Antoine Pitrou in bpo-32388.)

- The `PyCode_New()` has a new parameter in the second position (*posonlyargcount*) to support **PEP 570**, indicating the number of positional-only arguments.

- The functions `PyNode_AddChild()` and `PyParser_AddToken()` now accept two additional `int` arguments *end_lineno* and *end_col_offset*.

- The `libpython38.a` file to allow MinGW tools to link directly against `python38.dll` is no longer included in the regular Windows distribution. If you require this file, it may be generated with the `gendef` and `dlltool` tools, which are part of the MinGW binutils package:

```
gendef python38.dll > tmp.def
dlltool --dllname python38.dll --def tmp.def --output-lib libpython38.a
```

  The location of an installed `pythonXY.dll` will depend on the installation options and the version and language of Windows. See using-on-windows for more information. The resulting library should be placed in the same directory as `pythonXY.lib`, which is generally the `libs` directory under your Python installation.

  (Contributed by Steve Dower in bpo-37351.)

10.4 CPython bytecode changes

- The interpreter loop has been simplified by moving the logic of unrolling the stack of blocks into the compiler. The compiler emits now explicit instructions for adjusting the stack of values and calling the cleaning-up code for `break`, `continue` and `return`.

  Removed opcodes `BREAK_LOOP`, `CONTINUE_LOOP`, `SETUP_LOOP` and `SETUP_EXCEPT`. Added new opcodes `ROT_FOUR`, `BEGIN_FINALLY`, `CALL_FINALLY` and `POP_FINALLY`. Changed the behavior of `END_FINALLY` and `WITH_CLEANUP_START`.

  (Contributed by Mark Shannon, Antoine Pitrou and Serhiy Storchaka in bpo-17611.)

- Added new opcode `END_ASYNC_FOR` for handling exceptions raised when awaiting a next item in an `async for` loop. (Contributed by Serhiy Storchaka in bpo-33041.)

- The `MAP_ADD` now expects the value as the first element in the stack and the key as the second element. This change was made so the key is always evaluated before the value in dictionary comprehensions, as proposed by **PEP 572**. (Contributed by Jörn Heissler in bpo-35224.)

10.5 Demos and Tools

Added a benchmark script for timing various ways to access variables: `Tools/scripts/var_access_benchmark.py`. (Contributed by Raymond Hettinger in bpo-35884.)

Here's a summary of performance improvements since Python 3.3:

```
Python version                       3.3     3.4     3.5     3.6     3.7     3.8
--------------                       ---     ---     ---     ---     ---     ---

Variable and attribute read access:
    read_local                       4.0     7.1     7.1     5.4     5.1     3.9
    read_nonlocal                    5.3     7.1     8.1     5.8     5.4     4.4
    read_global                     13.3    15.5    19.0    14.3    13.6     7.6
    read_builtin                    20.0    21.1    21.6    18.5    19.0     7.5
    read_classvar_from_class        20.5    25.6    26.5    20.7    19.5    18.4
    read_classvar_from_instance     18.5    22.8    23.5    18.8    17.1    16.4
    read_instancevar                26.8    32.4    33.1    28.0    26.3    25.4
    read_instancevar_slots          23.7    27.8    31.3    20.8    20.8    20.2
    read_namedtuple                 68.5    73.8    57.5    45.0    46.8    18.4
    read_boundmethod                29.8    37.6    37.9    29.6    26.9    27.7

Variable and attribute write access:
    write_local                      4.6     8.7     9.3     5.5     5.3     4.3
    write_nonlocal                   7.3    10.5    11.1     5.6     5.5     4.7
    write_global                    15.9    19.7    21.2    18.0    18.0    15.8
    write_classvar                  81.9    92.9    96.0   104.6   102.1    39.2
    write_instancevar               36.4    44.6    45.8    40.0    38.9    35.5
    write_instancevar_slots         28.7    35.6    36.1    27.3    26.6    25.7

Data structure read access:
    read_list                       19.2    24.2    24.5    20.8    20.8    19.0
    read_deque                      19.9    24.7    25.5    20.2    20.6    19.8
    read_dict                       19.7    24.3    25.7    22.3    23.0    21.0
    read_strdict                    17.9    22.6    24.3    19.5    21.2    18.9

Data structure write access:
    write_list                      21.2    27.1    28.5    22.5    21.6    20.0
```

(continues on next page)

(continued from previous page)

```
    write_deque                     23.8     28.7     30.1     22.7     21.8     23.5
    write_dict                      25.9     31.4     33.3     29.3     29.2     24.7
    write_strdict                   22.9     28.4     29.9     27.5     25.2     23.1

Stack (or queue) operations:
    list_append_pop                144.2     93.4    112.7     75.4     74.2     50.8
    deque_append_pop                30.4     43.5     57.0     49.4     49.2     42.5
    deque_append_popleft            30.8     43.7     57.3     49.7     49.7     42.8

Timing loop:
    loop_overhead                    0.3      0.5      0.6      0.4      0.3      0.3

(Measured from the macOS 64-bit builds found at python.org)
```

Index