What's New in Python

Release 3.4.3

A. M. Kuchling

February 25, 2015

Python Software Foundation
Email: docs@python.org

Contents

Author R. David Murray <rdmurray@bitdance.com> (Editor)

This article explains the new features in Python 3.4, compared to 3.3. Python 3.4 was released on March 16, 2014. For full details, see the changelog.

See also:

PEP 429 – Python 3.4 Release Schedule

1 Summary – Release Highlights

New syntax features:

- No new syntax features were added in Python 3.4.

Other new features:

- *pip should always be available* (**PEP 453**).
- *Newly created file descriptors are non-inheritable* (**PEP 446**).
- command line option for *isolated mode* (issue 16499).
- *improvements in the handling of codecs* that are not text encodings (multiple issues).
- *A ModuleSpec Type* for the Import System (**PEP 451**). (Affects importer authors.)
- The `marshal` format has been made *more compact and efficient* (issue 16475).

New library modules:

- `asyncio`: *New provisional API for asynchronous IO* (**PEP 3156**).
- `ensurepip`: *Bootstrapping the pip installer* (**PEP 453**).
- `enum`: *Support for enumeration types* (**PEP 435**).
- `pathlib`: *Object-oriented filesystem paths* (**PEP 428**).
- `selectors`: *High-level and efficient I/O multiplexing*, built upon the `select` module primitives (part of **PEP 3156**).
- `statistics`: A basic *numerically stable statistics library* (**PEP 450**).
- `tracemalloc`: *Trace Python memory allocations* (**PEP 454**).

Significantly improved library modules:

- *Single-dispatch generic functions* in `functools` (**PEP 443**).
- New `pickle` *protocol 4* (**PEP 3154**).
- `multiprocessing` now has *an option to avoid using os.fork on Unix* (issue 8713).
- `email` has a new submodule, `contentmanager`, and a new `Message` subclass (`EmailMessage`) that *simplify MIME handling* (issue 18891).
- The `inspect` and `pydoc` modules are now capable of correct introspection of a much wider variety of callable objects, which improves the output of the Python `help()` system.
- The `ipaddress` module API has been declared stable

Security improvements:

- *Secure and interchangeable hash algorithm* (**PEP 456**).
- *Make newly created file descriptors non-inheritable* (**PEP 446**) to avoid leaking file descriptors to child processes.
- New command line option for *isolated mode*, (issue 16499).
- `multiprocessing` now has *an option to avoid using os.fork on Unix*. *spawn* and *forkserver* are more secure because they avoid sharing data with child processes.
- `multiprocessing` child processes on Windows no longer inherit all of the parent's inheritable handles, only the necessary ones.
- A new `hashlib.pbkdf2_hmac()` function provides the PKCS#5 password-based key derivation function 2.
- *TLSv1.1 and TLSv1.2 support* for `ssl`.
- *Retrieving certificates from the Windows system cert store support* for `ssl`.
- *Server-side SNI (Server Name Indication) support* for `ssl`.
- The `ssl.SSLContext` class has a *lot of improvements*.
- All modules in the standard library that support SSL now support server certificate verification, including hostname matching (`ssl.match_hostname()`) and CRLs (Certificate Revocation lists, see `ssl.SSLContext.load_verify_locations()`).

CPython implementation improvements:

- *Safe object finalization* (**PEP 442**).
- Leveraging **PEP 442**, in most cases *module globals are no longer set to None during finalization* (issue 18214).
- *Configurable memory allocators* (**PEP 445**).
- *Argument Clinic* (**PEP 436**).

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

2 New Features

2.1 PEP 453: Explicit Bootstrapping of PIP in Python Installations

Bootstrapping pip By Default

The new `ensurepip` module (defined in **PEP 453**) provides a standard cross-platform mechanism to bootstrap the pip installer into Python installations and virtual environments. The version of `pip` included with Python 3.4.0

is `pip` 1.5.4, and future 3.4.x maintenance releases will update the bundled version to the latest version of `pip` that is available at the time of creating the release candidate.

By default, the commands `pipX` and `pipX.Y` will be installed on all platforms (where X.Y stands for the version of the Python installation), along with the `pip` Python package and its dependencies. On Windows and in virtual environments on all platforms, the unversioned `pip` command will also be installed. On other platforms, the system wide unversioned `pip` command typically refers to the separately installed Python 2 version.

The *pyvenv* command line utility and the `venv` module make use of the `ensurepip` module to make `pip` readily available in virtual environments. When using the command line utility, `pip` is installed by default, while when using the `venv` module *venv-api* installation of `pip` must be requested explicitly.

For CPython *source builds on POSIX systems*, the `make install` and `make altinstall` commands bootstrap `pip` by default. This behaviour can be controlled through configure options, and overridden through Makefile options.

On Windows and Mac OS X, the CPython installers now default to installing `pip` along with CPython itself (users may opt out of installing it during the installation process). Window users will need to opt in to the automatic `PATH` modifications to have `pip` available from the command line by default, otherwise it can still be accessed through the Python launcher for Windows as `py -m pip`.

As discussed in the PEP, platform packagers may choose not to install these commands by default, as long as, when invoked, they provide clear and simple directions on how to install them on that platform (usually using the system package manager).

Note: To avoid conflicts between parallel Python 2 and Python 3 installations, only the versioned `pip3` and `pip3.4` commands are bootstrapped by default when `ensurepip` is invoked directly - the `--default-pip` option is needed to also request the unversioned `pip` command. `pyvenv` and the Windows installer ensure that the unqualified `pip` command is made available in those environments, and `pip` can always be invoked via the `-m` switch rather than directly to avoid ambiguity on systems with multiple Python installations.

Documentation Changes

As part of this change, the *installing-index* and *distributing-index* sections of the documentation have been completely redesigned as short getting started and FAQ documents. Most packaging documentation has now been moved out to the Python Packaging Authority maintained Python Packaging User Guide and the documentation of the individual projects.

However, as this migration is currently still incomplete, the legacy versions of those guides remaining available as *install-index* and *distutils-index*.

See also:

PEP 453 – Explicit bootstrapping of pip in Python installations PEP written by Donald Stufft and Nick Coghlan, implemented by Donald Stufft, Nick Coghlan, Martin von Löwis and Ned Deily.

2.2 PEP 446: Newly Created File Descriptors Are Non-Inheritable

PEP 446 makes newly created file descriptors *non-inheritable*. In general, this is the behavior an application will want: when launching a new process, having currently open files also open in the new process can lead to all sorts of hard to find bugs, and potentially to security issues.

However, there are occasions when inheritance is desired. To support these cases, the following new functions and methods are available:

- `os.get_inheritable()`, `os.set_inheritable()`
- `os.get_handle_inheritable()`, `os.set_handle_inheritable()`
- `socket.socket.get_inheritable()`, `socket.socket.set_inheritable()`

See also:

PEP 446 – Make newly created file descriptors non-inheritable PEP written and implemented by Victor Stinner.

2.3 Improvements to Codec Handling

Since it was first introduced, the `codecs` module has always been intended to operate as a type-neutral dynamic encoding and decoding system. However, its close coupling with the Python text model, especially the type restricted convenience methods on the builtin `str`, `bytes` and `bytearray` types, has historically obscured that fact.

As a key step in clarifying the situation, the `codecs.encode()` and `codecs.decode()` convenience functions are now properly documented in Python 2.7, 3.3 and 3.4. These functions have existed in the `codecs` module (and have been covered by the regression test suite) since Python 2.4, but were previously only discoverable through runtime introspection.

Unlike the convenience methods on `str`, `bytes` and `bytearray`, the `codecs` convenience functions support arbitrary codecs in both Python 2 and Python 3, rather than being limited to Unicode text encodings (in Python 3) or `basestring` <-> `basestring` conversions (in Python 2).

In Python 3.4, the interpreter is able to identify the known non-text encodings provided in the standard library and direct users towards these general purpose convenience functions when appropriate:

```
>>> b"abcdef".decode("hex")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
LookupError: 'hex' is not a text encoding; use codecs.decode() to handle arbitrary codec

>>> "hello".encode("rot13")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
LookupError: 'rot13' is not a text encoding; use codecs.encode() to handle arbitrary cod

>>> open("foo.txt", encoding="hex")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
LookupError: 'hex' is not a text encoding; use codecs.open() to handle arbitrary codecs
```

In a related change, whenever it is feasible without breaking backwards compatibility, exceptions raised during encoding and decoding operations are wrapped in a chained exception of the same type that mentions the name of the codec responsible for producing the error:

```
>>> import codecs

>>> codecs.decode(b"abcdefgh", "hex")
Traceback (most recent call last):
  File "/usr/lib/python3.4/encodings/hex_codec.py", line 20, in hex_decode
    return (binascii.a2b_hex(input), len(input))
binascii.Error: Non-hexadecimal digit found

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
binascii.Error: decoding with 'hex' codec failed (Error: Non-hexadecimal digit found)

>>> codecs.encode("hello", "bz2")
Traceback (most recent call last):
  File "/usr/lib/python3.4/encodings/bz2_codec.py", line 17, in bz2_encode
    return (bz2.compress(input), len(input))
  File "/usr/lib/python3.4/bz2.py", line 498, in compress
```

```
    return comp.compress(data) + comp.flush()
TypeError: 'str' does not support the buffer interface

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: encoding with 'bz2' codec failed (TypeError: 'str' does not support the buffe
```

Finally, as the examples above show, these improvements have permitted the restoration of the convenience aliases for the non-Unicode codecs that were themselves restored in Python 3.2. This means that encoding binary data to and from its hexadecimal representation (for example) can now be written as:

```
>>> from codecs import encode, decode
>>> encode(b"hello", "hex")
b'68656c6c6f'
>>> decode(b"68656c6c6f", "hex")
b'hello'
```

The binary and text transforms provided in the standard library are detailed in *binary-transforms* and *text-transforms*.

(Contributed by Nick Coghlan in issue 7475, issue 17827, issue 17828 and issue 19619.)

2.4 PEP 451: A ModuleSpec Type for the Import System

PEP 451 provides an encapsulation of the information about a module that the import machinery will use to load it (that is, a module specification). This helps simplify both the import implementation and several import-related APIs. The change is also a stepping stone for several future import-related improvements.

The public-facing changes from the PEP are entirely backward-compatible. Furthermore, they should be transparent to everyone but importer authors. Key finder and loader methods have been deprecated, but they will continue working. New importers should use the new methods described in the PEP. Existing importers should be updated to implement the new methods. See the *Deprecated* section for a list of methods that should be replaced and their replacements.

2.5 Other Language Changes

Some smaller changes made to the core Python language are:

- Unicode database updated to UCD version 6.3.
- `min()` and `max()` now accept a *default* keyword-only argument that can be used to specify the value they return if the iterable they are evaluating has no elements. (Contributed by Julian Berman in issue 18111.)
- Module objects are now `weakref`'able.
- Module `__file__` attributes (and related values) should now always contain absolute paths by default, with the sole exception of `__main__.__file__` when a script has been executed directly using a relative path. (Contributed by Brett Cannon in issue 18416.)
- All the UTF-* codecs (except UTF-7) now reject surrogates during both encoding and decoding unless the `surrogatepass` error handler is used, with the exception of the UTF-16 decoder (which accepts valid surrogate pairs) and the UTF-16 encoder (which produces them while encoding non-BMP characters). (Contributed by Victor Stinner, Kang-Hao (Kenny) Lu and Serhiy Storchaka in issue 12892.)
- New German EBCDIC *codec* `cp273`. (Contributed by Michael Bierenfeld and Andrew Kuchling in issue 1097797.)
- New Ukrainian *codec* `cp1125`. (Contributed by Serhiy Storchaka in issue 19668.)
- `bytes`.join() and `bytearray`.join() now accept arbitrary buffer objects as arguments. (Contributed by Antoine Pitrou in issue 15958.)

- The `int` constructor now accepts any object that has an `__index__` method for its *base* argument. (Contributed by Mark Dickinson in issue 16772.)
- Frame objects now have a `clear()` method that clears all references to local variables from the frame. (Contributed by Antoine Pitrou in issue 17934.)
- `memoryview` is now registered as a `Sequence`, and supports the `reversed()` builtin. (Contributed by Nick Coghlan and Claudiu Popa in issue 18690 and issue 19078.)
- Signatures reported by `help()` have been modified and improved in several cases as a result of the introduction of Argument Clinic and other changes to the `inspect` and `pydoc` modules.
- `__length_hint__()` is now part of the formal language specification (see **PEP 424**). (Contributed by Armin Ronacher in issue 16148.)

3 New Modules

3.1 asyncio

The new `asyncio` module (defined in **PEP 3156**) provides a standard pluggable event loop model for Python, providing solid asynchronous IO support in the standard library, and making it easier for other event loop implementations to interoperate with the standard library and each other.

For Python 3.4, this module is considered a *provisional API*.

See also:

PEP 3156 – Asynchronous IO Support Rebooted: the "asyncio" Module PEP written and implementation led by Guido van Rossum.

3.2 ensurepip

The new `ensurepip` module is the primary infrastructure for the **PEP 453** implementation. In the normal course of events end users will not need to interact with this module, but it can be used to manually bootstrap `pip` if the automated bootstrapping into an installation or virtual environment was declined.

`ensurepip` includes a bundled copy of `pip`, up-to-date as of the first release candidate of the release of CPython with which it ships (this applies to both maintenance releases and feature releases). `ensurepip` does not access the internet. If the installation has Internet access, after `ensurepip` is run the bundled `pip` can be used to upgrade `pip` to a more recent release than the bundled one. (Note that such an upgraded version of `pip` is considered to be a separately installed package and will not be removed if Python is uninstalled.)

The module is named *ensure*pip because if called when `pip` is already installed, it does nothing. It also has an `--upgrade` option that will cause it to install the bundled copy of `pip` if the existing installed version of `pip` is older than the bundled copy.

3.3 enum

The new `enum` module (defined in **PEP 435**) provides a standard implementation of enumeration types, allowing other modules (such as `socket`) to provide more informative error messages and better debugging support by replacing opaque integer constants with backwards compatible enumeration values.

See also:

PEP 435 – Adding an Enum type to the Python standard library PEP written by Barry Warsaw, Eli Bendersky and Ethan Furman, implemented by Ethan Furman.

3.4 pathlib

The new `pathlib` module offers classes representing filesystem paths with semantics appropriate for different operating systems. Path classes are divided between *pure paths*, which provide purely computational operations without I/O, and *concrete paths*, which inherit from pure paths but also provide I/O operations.

For Python 3.4, this module is considered a *provisional API*.

See also:

PEP 428 – The pathlib module – object-oriented filesystem paths PEP written and implemented by Antoine Pitrou.

3.5 selectors

The new `selectors` module (created as part of implementing **PEP 3156**) allows high-level and efficient I/O multiplexing, built upon the `select` module primitives.

3.6 statistics

The new `statistics` module (defined in **PEP 450**) offers some core statistics functionality directly in the standard library. This module supports calculation of the mean, median, mode, variance and standard deviation of a data series.

See also:

PEP 450 – Adding A Statistics Module To The Standard Library PEP written and implemented by Steven D'Aprano

3.7 tracemalloc

The new `tracemalloc` module (defined in **PEP 454**) is a debug tool to trace memory blocks allocated by Python. It provides the following information:

- Trace where an object was allocated
- Statistics on allocated memory blocks per filename and per line number: total size, number and average size of allocated memory blocks
- Compute the differences between two snapshots to detect memory leaks

See also:

PEP 454 – Add a new tracemalloc module to trace Python memory allocations PEP written and implemented by Victor Stinner

4 Improved Modules

4.1 abc

New function `abc.get_cache_token()` can be used to know when to invalidate caches that are affected by changes in the object graph. (Contributed by Łukasz Langa in issue 16832.)

New class `ABC` has `ABCMeta` as its meta class. Using `ABC` as a base class has essentially the same effect as specifying `metaclass=abc.ABCMeta`, but is simpler to type and easier to read. (Contributed by Bruno Dupuis in issue 16049.)

4.2 aifc

The `getparams()` method now returns a namedtuple rather than a plain tuple. (Contributed by Claudiu Popa in issue 17818.)

`aifc.open()` now supports the context management protocol: when used in a `with` block, the `close()` method of the returned object will be called automatically at the end of the block. (Contributed by Serhiy Storchacha in issue 16486.)

The `writeframesraw()` and `writeframes()` methods now accept any *bytes-like object*. (Contributed by Serhiy Storchaka in issue 8311.)

4.3 argparse

The `FileType` class now accepts *encoding* and *errors* arguments, which are passed through to `open()`. (Contributed by Lucas Maystre in issue 11175.)

4.4 audioop

`audioop` now supports 24-bit samples. (Contributed by Serhiy Storchaka in issue 12866.)

New `byteswap()` function converts big-endian samples to little-endian and vice versa. (Contributed by Serhiy Storchaka in issue 19641.)

All `audioop` functions now accept any *bytes-like object*. Strings are not accepted: they didn't work before, now they raise an error right away. (Contributed by Serhiy Storchaka in issue 16685.)

4.5 base64

The encoding and decoding functions in `base64` now accept any *bytes-like object* in cases where it previously required a `bytes` or `bytearray` instance. (Contributed by Nick Coghlan in issue 17839.)

New functions `a85encode()`, `a85decode()`, `b85encode()`, and `b85decode()` provide the ability to encode and decode binary data from and to `Ascii85` and the git/mercurial `Base85` formats, respectively. The `a85` functions have options that can be used to make them compatible with the variants of the `Ascii85` encoding, including the Adobe variant. (Contributed by Martin Morrison, the Mercurial project, Serhiy Storchaka, and Antoine Pitrou in issue 17618.)

4.6 collections

The `ChainMap.new_child()` method now accepts an *m* argument specifying the child map to add to the chain. This allows an existing mapping and/or a custom mapping type to be used for the child. (Contributed by Vinay Sajip in issue 16613.)

4.7 colorsys

The number of digits in the coefficients for the RGB — YIQ conversions have been expanded so that they match the FCC NTSC versions. The change in results should be less than 1% and may better match results found elsewhere. (Contributed by Brian Landers and Serhiy Storchaka in issue 14323.)

4.8 contextlib

The new `contextlib.suppress` context manager helps to clarify the intent of code that deliberately suppresses exceptions from a single statement. (Contributed by Raymond Hettinger in issue 15806 and Zero Piraeus in issue 19266.)

The new `contextlib.redirect_stdout()` context manager makes it easier for utility scripts to handle inflexible APIs that write their output to `sys.stdout` and don't provide any options to redirect it. Using the context manager, the `sys.stdout` output can be redirected to any other stream or, in conjunction with `io.StringIO`, to a string. The latter can be especially useful, for example, to capture output from a function that was written to implement a command line interface. It is recommended only for utility scripts because it affects the global state of `sys.stdout`. (Contributed by Raymond Hettinger in issue 15805.)

The `contextlib` documentation has also been updated to include a *discussion* of the differences between single use, reusable and reentrant context managers.

4.9 dbm

`dbm.open()` objects now support the context management protocol. When used in a `with` statement, the `close` method of the database object will be called automatically at the end of the block. (Contributed by Claudiu Popa and Nick Coghlan in issue 19282.)

4.10 dis

Functions `show_code()`, `dis()`, `distb()`, and `disassemble()` now accept a keyword-only *file* argument that controls where they write their output.

The `dis` module is now built around an `Instruction` class that provides object oriented access to the details of each individual bytecode operation.

A new method, `get_instructions()`, provides an iterator that emits the Instruction stream for a given piece of Python code. Thus it is now possible to write a program that inspects and manipulates a bytecode object in ways different from those provided by the `dis` module itself. For example:

```
>>> import dis
>>> for instr in dis.get_instructions(lambda x: x + 1):
...     print(instr.opname)
LOAD_FAST
LOAD_CONST
BINARY_ADD
RETURN_VALUE
```

The various display tools in the `dis` module have been rewritten to use these new components.

In addition, a new application-friendly class `Bytecode` provides an object-oriented API for inspecting bytecode in both in human-readable form and for iterating over instructions. The `Bytecode` constructor takes the same arguments that `get_instruction()` does (plus an optional *current_offset*), and the resulting object can be iterated to produce `Instruction` objects. But it also has a `dis` method, equivalent to calling `dis` on the constructor argument, but returned as a multi-line string:

```
>>> bytecode = dis.Bytecode(lambda x: x +1, current_offset=3)
>>> for instr in bytecode:
...     print('{} ({})'.format(instr.opname, instr.opcode))
LOAD_FAST (124)
LOAD_CONST (100)
BINARY_ADD (23)
RETURN_VALUE (83)
>>> bytecode.dis().splitlines()
['  1           0 LOAD_FAST                0 (x)',
 '      -->     3 LOAD_CONST               1 (1)',
 '              6 BINARY_ADD',
 '              7 RETURN_VALUE']
```

`Bytecode` also has a class method, `from_traceback()`, that provides the ability to manipulate a traceback (that is, `print(Bytecode.from_traceback(tb).dis())` is equivalent to `distb(tb)`).

(Contributed by Nick Coghlan, Ryan Kelly and Thomas Kluyver in issue 11816 and Claudiu Popa in issue 17916.)

New function `stack_effect()` computes the effect on the Python stack of a given opcode and argument, information that is not otherwise available. (Contributed by Larry Hastings in issue 19722.)

4.11 doctest

A new *option flag*, `FAIL_FAST`, halts test running as soon as the first failure is detected. (Contributed by R. David Murray and Daniel Urban in issue 16522.)

The `doctest` command line interface now uses `argparse`, and has two new options, `-o` and `-f`. `-o` allows *doctest options* to be specified on the command line, and `-f` is a shorthand for `-o FAIL_FAST` (to parallel the similar option supported by the `unittest` CLI). (Contributed by R. David Murray in issue 11390.)

`doctest` will now find doctests in extension module `__doc__` strings. (Contributed by Zachary Ware in issue 3158.)

4.12 email

`as_string()` now accepts a *policy* argument to override the default policy of the message when generating a string representation of it. This means that `as_string` can now be used in more circumstances, instead of having to create and use a `generator` in order to pass formatting parameters to its `flatten` method. (Contributed by R. David Murray in issue 18600.)

New method `as_bytes()` added to produce a bytes representation of the message in a fashion similar to how `as_string` produces a string representation. It does not accept the *maxheaderlen* argument, but does accept the *unixfrom* and *policy* arguments. The `Message __bytes__()` method calls it, meaning that `bytes(mymsg)` will now produce the intuitive result: a bytes object containing the fully formatted message. (Contributed by R. David Murray in issue 18600.)

The `Message.set_param()` message now accepts a *replace* keyword argument. When specified, the associated header will be updated without changing its location in the list of headers. For backward compatibility, the default is `False`. (Contributed by R. David Murray in issue 18891.) A pair of new subclasses of `Message` have been added (`EmailMessage` and `MIMEPart`), along with a new sub-module, `contentmanager` and a new `policy` attribute `content_manager`. All documentation is currently in the new module, which is being added as part of email's new *provisional API*. These classes provide a number of new methods that make extracting content from and inserting content into email messages much easier. For details, see the `contentmanager` documentation and the *email-contentmanager-api-examples*. These API additions complete the bulk of the work that was planned as part of the email6 project. The currently provisional API is scheduled to become final in Python 3.5 (possibly with a few minor additions in the area of error handling). (Contributed by R. David Murray in issue 18891.)

4.13 filecmp

A new `clear_cache()` function provides the ability to clear the `filecmp` comparison cache, which uses `os.stat()` information to determine if the file has changed since the last compare. This can be used, for example, if the file might have been changed and re-checked in less time than the resolution of a particular filesystem's file modification time field. (Contributed by Mark Levitt in issue 18149.)

New module attribute `DEFAULT_IGNORES` provides the list of directories that are used as the default value for the *ignore* parameter of the `dircmp()` function. (Contributed by Eli Bendersky in issue 15442.)

4.14 functools

The new `partialmethod()` descriptor brings partial argument application to descriptors, just as `partial()` provides for normal callables. The new descriptor also makes it easier to get arbitrary callables (including `partial()` instances) to behave like normal instance methods when included in a class definition. (Contributed by Alon Horev and Nick Coghlan in issue 4331.) The new `singledispatch()` decorator brings support for single-dispatch generic functions to the Python standard library. Where object oriented programming focuses on

grouping multiple operations on a common set of data into a class, a generic function focuses on grouping multiple implementations of an operation that allows it to work with *different* kinds of data.

See also:

PEP 443 – Single-dispatch generic functions PEP written and implemented by Łukasz Langa.

`total_ordering()` now supports a return value of `NotImplemented` from the underlying comparison function. (Contributed by Katie Miller in issue 10042.)

A pure-python version of the `partial()` function is now in the stdlib; in CPython it is overridden by the C accelerated version, but it is available for other implementations to use. (Contributed by Brian Thorne in issue 12428.)

4.15 gc

New function `get_stats()` returns a list of three per-generation dictionaries containing the collections statistics since interpreter startup. (Contributed by Antoine Pitrou in issue 16351.)

4.16 glob

A new function `escape()` provides a way to escape special characters in a filename so that they do not become part of the globbing expansion but are instead matched literally. (Contributed by Serhiy Storchaka in issue 8402.)

4.17 hashlib

A new `hashlib.pbkdf2_hmac()` function provides the PKCS#5 password-based key derivation function 2. (Contributed by Christian Heimes in issue 18582.)

The `name` attribute of `hashlib` hash objects is now a formally supported interface. It has always existed in CPython's `hashlib` (although it did not return lower case names for all supported hashes), but it was not a public interface and so some other Python implementations have not previously supported it. (Contributed by Jason R. Coombs in issue 18532.)

4.18 hmac

`hmac` now accepts `bytearray` as well as `bytes` for the *key* argument to the `new()` function, and the *msg* parameter to both the `new()` function and the `update()` method now accepts any type supported by the `hashlib` module. (Contributed by Jonas Borgström in issue 18240.)

The *digestmod* argument to the `hmac.new()` function may now be any hash digest name recognized by `hashlib`. In addition, the current behavior in which the value of *digestmod* defaults to `MD5` is deprecated: in a future version of Python there will be no default value. (Contributed by Christian Heimes in issue 17276.)

With the addition of `block_size` and `name` attributes (and the formal documentation of the `digest_size` attribute), the `hmac` module now conforms fully to the **PEP 247** API. (Contributed by Christian Heimes in issue 18775.)

4.19 html

New function `unescape()` function converts HTML5 character references to the corresponding Unicode characters. (Contributed by Ezio Melotti in issue 2927.)

`HTMLParser` accepts a new keyword argument *convert_charrefs* that, when `True`, automatically converts all character references. For backward-compatibility, its value defaults to `False`, but it will change to `True` in a future version of Python, so you are invited to set it explicitly and update your code to use this new feature. (Contributed by Ezio Melotti in issue 13633.)

The *strict* argument of `HTMLParser` is now deprecated. (Contributed by Ezio Melotti in issue 15114.)

4.20 http

`send_error()` now accepts an optional additional *explain* parameter which can be used to provide an extended error description, overriding the hardcoded default if there is one. This extended error description will be formatted using the `error_message_format` attribute and sent as the body of the error response. (Contributed by Karl Cow in issue 12921.)

The `http.server` *command line interface* now has a `-b/--bind` option that causes the server to listen on a specific address. (Contributed by Malte Swart in issue 17764.)

4.21 importlib

The `InspectLoader` ABC defines a new method, `source_to_code()` that accepts source data and a path and returns a code object. The default implementation is equivalent to `compile(data, path, 'exec', dont_inherit=True)`. (Contributed by Eric Snow and Brett Cannon in issue 15627.)

`InspectLoader` also now has a default implementation for the `get_code()` method. However, it will normally be desirable to override the default implementation for performance reasons. (Contributed by Brett Cannon in issue 18072.)

The `reload()` function has been moved from `imp` to `importlib` as part of the `imp` module deprecation. (Contributed by Berker Peksag in issue 18193.)

`importlib.util` now has a `MAGIC_NUMBER` attribute providing access to the bytecode version number. This replaces the `get_magic()` function in the deprecated `imp` module. (Contributed by Brett Cannon in issue 18192.)

New `importlib.util` functions `cache_from_source()` and `source_from_cache()` replace the same-named functions in the deprecated `imp` module. (Contributed by Brett Cannon in issue 18194.)

The `importlib` bootstrap `NamespaceLoader` now conforms to the `InspectLoader` ABC, which means that `runpy` and `python -m` can now be used with namespace packages. (Contributed by Brett Cannon in issue 18058.)

`importlib.util` has a new function `decode_source()` that decodes source from bytes using universal newline processing. This is useful for implementing `InspectLoader.get_source()` methods.

`importlib.machinery.ExtensionFileLoader` now has a `get_filename()` method. This was inadvertently omitted in the original implementation. (Contributed by Eric Snow in issue 19152.)

4.22 inspect

The `inspect` module now offers a basic *command line interface* to quickly display source code and other information for modules, classes and functions. (Contributed by Claudiu Popa and Nick Coghlan in issue 18626.)

`unwrap()` makes it easy to unravel wrapper function chains created by `functools.wraps()` (and any other API that sets the `__wrapped__` attribute on a wrapper function). (Contributed by Daniel Urban, Aaron Iles and Nick Coghlan in issue 13266.)

As part of the implementation of the new `enum` module, the `inspect` module now has substantially better support for custom `__dir__` methods and dynamic class attributes provided through metaclasses. (Contributed by Ethan Furman in issue 18929 and issue 19030.)

`getfullargspec()` and `getargspec()` now use the `signature()` API. This allows them to support a much broader range of callables, including those with `__signature__` attributes, those with metadata provided by argument clinic, `functools.partial()` objects and more. Note that, unlike `signature()`, these functions still ignore `__wrapped__` attributes, and report the already bound first argument for bound methods, so it is still necessary to update your code to use `signature()` directly if those features are desired. (Contributed by Yury Selivanov in issue 17481.)

`signature()` now supports duck types of CPython functions, which adds support for functions compiled with Cython. (Contributed by Stefan Behnel and Yury Selivanov in issue 17159.)

4.23 ipaddress

`ipaddress` was added to the standard library in Python 3.3 as a *provisional API*. With the release of Python 3.4, this qualification has been removed: `ipaddress` is now considered a stable API, covered by the normal standard library requirements to maintain backwards compatibility.

A new `is_global` property is `True` if an address is globally routeable. (Contributed by Peter Moody in issue 17400.)

4.24 logging

The `TimedRotatingFileHandler` has a new *atTime* parameter that can be used to specify the time of day when rollover should happen. (Contributed by Ronald Oussoren in issue 9556.)

`SocketHandler` and `DatagramHandler` now support Unix domain sockets (by setting *port* to `None`). (Contributed by Vinay Sajip in commit ce46195b56a9.)

`fileConfig()` now accepts a `configparser.RawConfigParser` subclass instance for the *fname* parameter. This facilitates using a configuration file when logging configuration is just a part of the overall application configuration, or where the application modifies the configuration before passing it to `fileConfig()`. (Contributed by Vinay Sajip in issue 16110.)

Logging configuration data received from a socket via the `logging.config.listen()` function can now be validated before being processed by supplying a verification function as the argument to the new *verify* keyword argument. (Contributed by Vinay Sajip in issue 15452.)

4.25 marshal

The default `marshal` version has been bumped to 3. The code implementing the new version restores the Python2 behavior of recording only one copy of interned strings and preserving the interning on deserialization, and extends this "one copy" ability to any object type (including handling recursive references). This reduces both the size of `.pyc` files and the amount of memory a module occupies in memory when it is loaded from a `.pyc` (or `.pyo`) file. (Contributed by Kristján Valur Jónsson in issue 16475, with additional speedups by Antoine Pitrou in issue 19219.)

4.26 mmap

mmap objects can now be `weakref`ed. (Contributed by Valerie Lambert in issue 4885.)

4.27 multiprocessing

On Unix two new *start methods*, (`spawn` and `forkserver`, have been added for starting processes using `multiprocessing`. These make the mixing of processes with threads more robust, and the `spawn` method matches the semantics that multiprocessing has always used on Windows. New function `get_all_start_methods()` reports all start methods available on the platform, `get_start_method()` reports the current start method, and `set_start_method()` sets the start method. (Contributed by Richard Oudkerk in issue 8713.)

`multiprocessing` also now has the concept of a `context`, which determines how child processes are created. New function `get_context()` returns a context that uses a specified start method. It has the same API as the `multiprocessing` module itself, so you can use it to create `Pools` and other objects that will operate within that context. This allows a framework and an application or different parts of the same application to use multiprocessing without interfering with each other. (Contributed by Richard Oudkerk in issue 18999.)

Except when using the old *fork* start method, child processes no longer inherit unneeded handles/file descriptors from their parents (part of issue 8713).

multiprocessing now relies on runpy (which implements the -m switch) to initialise __main__ appropriately in child processes when using the spawn or forkserver start methods. This resolves some edge cases where combining multiprocessing, the -m command line switch, and explicit relative imports could cause obscure failures in child processes. (Contributed by Nick Coghlan in issue 19946.)

4.28 operator

New function length_hint() provides an implementation of the specification for how the __length_hint__() special method should be used, as part of the **PEP 424** formal specification of this language feature. (Contributed by Armin Ronacher in issue 16148.)

There is now a pure-python version of the operator module available for reference and for use by alternate implementations of Python. (Contributed by Zachary Ware in issue 16694.)

4.29 os

There are new functions to get and set the *inheritable flag* of a file descriptor (os.get_inheritable(), os.set_inheritable()) or a Windows handle (os.get_handle_inheritable(), os.set_handle_inheritable()).

New function cpu_count() reports the number of CPUs available on the platform on which Python is running (or None if the count can't be determined). The multiprocessing.cpu_count() function is now implemented in terms of this function). (Contributed by Trent Nelson, Yogesh Chaudhari, Victor Stinner, and Charles-François Natali in issue 17914.)

os.path.samestat() is now available on the Windows platform (and the os.path.samefile() implementation is now shared between Unix and Windows). (Contributed by Brian Curtin in issue 11939.)

os.path.ismount() now recognizes volumes mounted below a drive root on Windows. (Contributed by Tim Golden in issue 9035.)

os.open() supports two new flags on platforms that provide them, O_PATH (un-opened file descriptor), and O_TMPFILE (unnamed temporary file; as of 3.4.0 release available only on Linux systems with a kernel version of 3.11 or newer that have uapi headers). (Contributed by Christian Heimes in issue 18673 and Benjamin Peterson, respectively.)

4.30 pdb

pdb has been enhanced to handle generators, yield, and yield from in a more useful fashion. This is especially helpful when debugging asyncio based programs. (Contributed by Andrew Svetlov and Xavier de Gaye in issue 16596.)

The print command has been removed from pdb, restoring access to the Python print() function from the pdb command line. Python2's pdb did not have a print command; instead, entering print executed the print statement. In Python3 print was mistakenly made an alias for the pdb p command. p, however, prints the repr of its argument, not the str like the Python2 print command did. Worse, the Python3 pdb print command shadowed the Python3 print function, making it inaccessible at the pdb prompt. (Contributed by Connor Osborn in issue 18764.)

4.31 pickle

pickle now supports (but does not use by default) a new pickle protocol, protocol 4. This new protocol addresses a number of issues that were present in previous protocols, such as the serialization of nested classes, very large strings and containers, and classes whose __new__() method takes keyword-only arguments. It also provides some efficiency improvements.

See also:

PEP 3154 – Pickle protocol 4 PEP written by Antoine Pitrou and implemented by Alexandre Vassalotti.

4.32 plistlib

`plistlib` now has an API that is similar to the standard pattern for stdlib serialization protocols, with new `load()`, `dump()`, `loads()`, and `dumps()` functions. (The older API is now deprecated.) In addition to the already supported XML plist format (`FMT_XML`), it also now supports the binary plist format (`FMT_BINARY`). (Contributed by Ronald Oussoren and others in issue 14455.)

4.33 poplib

Two new methods have been added to `poplib`: `capa()`, which returns the list of capabilities advertised by the POP server, and `stls()`, which switches a clear-text POP3 session into an encrypted POP3 session if the POP server supports it. (Contributed by Lorenzo Catucci in issue 4473.)

4.34 pprint

The `pprint` module's `PrettyPrinter` class and its `pformat()`, and `pprint()` functions have a new option, *compact*, that controls how the output is formatted. Currently setting *compact* to `True` means that sequences will be printed with as many sequence elements as will fit within *width* on each (indented) line. (Contributed by Serhiy Storchaka in issue 19132.)

Long strings are now wrapped using Python's normal line continuation syntax. (Contributed by Antoine Pitrou in issue 17150.)

4.35 pty

`pty.spawn()` now returns the status value from `os.waitpid()` on the child process, instead of `None`. (Contributed by Gregory P. Smith.)

4.36 pydoc

The `pydoc` module is now based directly on the `inspect.signature()` introspection API, allowing it to provide signature information for a wider variety of callable objects. This change also means that `__wrapped__` attributes are now taken into account when displaying help information. (Contributed by Larry Hastings in issue 19674.)

The `pydoc` module no longer displays the `self` parameter for already bound methods. Instead, it aims to always display the exact current signature of the supplied callable. (Contributed by Larry Hastings in issue 20710.)

In addition to the changes that have been made to `pydoc` directly, its handling of custom `__dir__` methods and various descriptor behaviours has also been improved substantially by the underlying changes in the `inspect` module.

As the `help()` builtin is based on `pydoc`, the above changes also affect the behaviour of `help()`.

4.37 re

New `fullmatch()` function and `regex.fullmatch()` method anchor the pattern at both ends of the string to match. This provides a way to be explicit about the goal of the match, which avoids a class of subtle bugs where `$` characters get lost during code changes or the addition of alternatives to an existing regular expression. (Contributed by Matthew Barnett in issue 16203.)

The repr of *regex objects* now includes the pattern and the flags; the repr of *match objects* now includes the start, end, and the part of the string that matched. (Contributed by Hugo Lopes Tavares and Serhiy Storchaka in issue 13592 and issue 17087.)

4.38 resource

New `prlimit()` function, available on Linux platforms with a kernel version of 2.6.36 or later and glibc of 2.13 or later, provides the ability to query or set the resource limits for processes other than the one making the call. (Contributed by Christian Heimes in issue 16595.)

On Linux kernel version 2.6.36 or later, there are there are also some new Linux specific constants: `RLIMIT_MSGQUEUE`, `RLIMIT_NICE`, `RLIMIT_RTPRIO`, `RLIMIT_RTTIME`, and `RLIMIT_SIGPENDING`. (Contributed by Christian Heimes in issue 19324.)

On FreeBSD version 9 and later, there some new FreeBSD specific constants: `RLIMIT_SBSIZE`, `RLIMIT_SWAP`, and `RLIMIT_NPTS`. (Contributed by Claudiu Popa in issue 19343.)

4.39 select

`epoll` objects now support the context management protocol. When used in a `with` statement, the `close()` method will be called automatically at the end of the block. (Contributed by Serhiy Storchaka in issue 16488.)

`devpoll` objects now have `fileno()` and `close()` methods, as well as a new attribute `closed`. (Contributed by Victor Stinner in issue 18794.)

4.40 shelve

`Shelf` instances may now be used in `with` statements, and will be automatically closed at the end of the `with` block. (Contributed by Filip Gruszczyński in issue 13896.)

4.41 shutil

`copyfile()` now raises a specific `Error` subclass, `SameFileError`, when the source and destination are the same file, which allows an application to take appropriate action on this specific error. (Contributed by Atsuo Ishimoto and Hynek Schlawack in issue 1492704.)

4.42 smtpd

The `SMTPServer` and `SMTPChannel` classes now accept a *map* keyword argument which, if specified, is passed in to `asynchat.async_chat` as its *map* argument. This allows an application to avoid affecting the global socket map. (Contributed by Vinay Sajip in issue 11959.)

4.43 smtplib

`SMTPException` is now a subclass of `OSError`, which allows both socket level errors and SMTP protocol level errors to be caught in one try/except statement by code that only cares whether or not an error occurred. (Contributed by Ned Jackson Lovely in issue 2118.)

4.44 socket

The socket module now supports the `CAN_BCM` protocol on platforms that support it. (Contributed by Brian Thorne in issue 15359.)

Socket objects have new methods to get or set their *inheritable flag*, `get_inheritable()` and `set_inheritable()`.

The `socket.AF_*` and `socket.SOCK_*` constants are now enumeration values using the new `enum` module. This allows meaningful names to be printed during debugging, instead of integer "magic numbers".

The `AF_LINK` constant is now available on BSD and OSX.

inet_pton() and inet_ntop() are now supported on Windows. (Contributed by Atsuo Ishimoto in issue 7171.)

4.45 sqlite3

A new boolean parameter to the connect() function, *uri*, can be used to indicate that the *database* parameter is a uri (see the SQLite URI documentation). (Contributed by poq in issue 13773.)

4.46 ssl

PROTOCOL_TLSv1_1 and PROTOCOL_TLSv1_2 (TLSv1.1 and TLSv1.2 support) have been added; support for these protocols is only available if Python is linked with OpenSSL 1.0.1 or later. (Contributed by Michele Orrù and Antoine Pitrou in issue 16692.) New function create_default_context() provides a standard way to obtain an SSLContext whose settings are intended to be a reasonable balance between compatibility and security. These settings are more stringent than the defaults provided by the SSLContext constructor, and may be adjusted in the future, without prior deprecation, if best-practice security requirements change. The new recommended best practice for using stdlib libraries that support SSL is to use create_default_context() to obtain an SSLContext object, modify it if needed, and then pass it as the *context* argument of the appropriate stdlib API. (Contributed by Christian Heimes in issue 19689.)

SSLContext method load_verify_locations() accepts a new optional argument *cadata*, which can be used to provide PEM or DER encoded certificates directly via strings or bytes, respectively. (Contributed by Christian Heimes in issue 18138.)

New function get_default_verify_paths() returns a named tuple of the paths and environment variables that the set_default_verify_paths() method uses to set OpenSSL's default cafile and capath. This can be an aid in debugging default verification issues. (Contributed by Christian Heimes in issue 18143.)

SSLContext has a new method, cert_store_stats(), that reports the number of loaded X.509 certs, X.509 CA certs, and certificate revocation lists (crls), as well as a get_ca_certs() method that returns a list of the loaded CA certificates. (Contributed by Christian Heimes in issue 18147.)

If OpenSSL 0.9.8 or later is available, SSLContext has an new attribute verify_flags that can be used to control the certificate verification process by setting it to some combination of the new constants VERIFY_DEFAULT, VERIFY_CRL_CHECK_LEAF, VERIFY_CRL_CHECK_CHAIN, or VERIFY_X509_STRICT. OpenSSL does not do any CRL verification by default. (Contributed by Christien Heimes in issue 8813.)

New SSLContext method load_default_certs() loads a set of default "certificate authority" (CA) certificates from default locations, which vary according to the platform. It can be used to load both TLS web server authentication certificates (purpose=SERVER_AUTH) for a client to use to verify a server, and certificates for a server to use in verifying client certificates (purpose=CLIENT_AUTH). (Contributed by Christian Heimes in issue 19292.) Two new windows-only functions, enum_certificates() and enum_crls() provide the ability to retrieve certificates, certificate information, and CRLs from the Windows cert store. (Contributed by Christian Heimes in issue 17134.) Support for server-side SNI (Server Name Indication) using the new ssl.SSLContext.set_servername_callback() method. (Contributed by Daniel Black in issue 8109.)

The dictionary returned by SSLSocket.getpeercert() contains additional X509v3 extension items: crlDistributionPoints, calIssuers, and OCSP URIs. (Contributed by Christian Heimes in issue 18379.)

4.47 stat

The stat module is now backed by a C implementation in _stat. A C implementation is required as most of the values aren't standardized and are platform-dependent. (Contributed by Christian Heimes in issue 11016.)

The module supports new ST_MODE flags, S_IFDOOR, S_IFPORT, and S_IFWHT. (Contributed by Christian Hiemes in issue 11016.)

4.48 struct

New function `iter_unpack` and a new `struct.Struct.iter_unpack()` method on compiled formats provide streamed unpacking of a buffer containing repeated instances of a given format of data. (Contributed by Antoine Pitrou in issue 17804.)

4.49 subprocess

`check_output()` now accepts an *input* argument that can be used to provide the contents of `stdin` for the command that is run. (Contributed by Zack Weinberg in issue 16624.)

`getstatus()` and `getstatusoutput()` now work on Windows. This change was actually inadvertently made in 3.3.4. (Contributed by Tim Golden in issue 10197.)

4.50 sunau

The `getparams()` method now returns a namedtuple rather than a plain tuple. (Contributed by Claudiu Popa in issue 18901.)

`sunau.open()` now supports the context management protocol: when used in a `with` block, the `close` method of the returned object will be called automatically at the end of the block. (Contributed by Serhiy Storchaka in issue 18878.)

`AU_write.setsampwidth()` now supports 24 bit samples, thus adding support for writing 24 sample using the module. (Contributed by Serhiy Storchaka in issue 19261.)

The `writeframesraw()` and `writeframes()` methods now accept any *bytes-like object*. (Contributed by Serhiy Storchaka in issue 8311.)

4.51 sys

New function `sys.getallocatedblocks()` returns the current number of blocks allocated by the interpreter. (In CPython with the default `--with-pymalloc` setting, this is allocations made through the `PyObject_Malloc()` API.) This can be useful for tracking memory leaks, especially if automated via a test suite. (Contributed by Antoine Pitrou in issue 13390.)

When the Python interpreter starts in *interactive mode*, it checks for an `__interactivehook__` attribute on the `sys` module. If the attribute exists, its value is called with no arguments just before interactive mode is started. The check is made after the `PYTHONSTARTUP` file is read, so it can be set there. The `site` module *sets it* to a function that enables tab completion and history saving (in `~/.python-history`) if the platform supports `readline`. If you do not want this (new) behavior, you can override it in `PYTHONSTARTUP`, `sitecustomize`, or `usercustomize` by deleting this attribute from `sys` (or setting it to some other callable). (Contributed by Éric Araujo and Antoine Pitrou in issue 5845.)

4.52 tarfile

The `tarfile` module now supports a simple *tarfile-commandline* when called as a script directly or via `-m`. This can be used to create and extract tarfile archives. (Contributed by Berker Peksag in issue 13477.)

4.53 textwrap

The `TextWrapper` class has two new attributes/constructor arguments: `max_lines`, which limits the number of lines in the output, and `placeholder`, which is a string that will appear at the end of the output if it has been truncated because of *max_lines*. Building on these capabilities, a new convenience function `shorten()` collapses all of the whitespace in the input to single spaces and produces a single line of a given *width* that ends with the *placeholder* (by default, `[...]`). (Contributed by Antoine Pitrou and Serhiy Storchaka in issue 18585 and issue 18725.)

4.54 threading

The `Thread` object representing the main thread can be obtained from the new `main_thread()` function. In normal conditions this will be the thread from which the Python interpreter was started. (Contributed by Andrew Svetlov in issue 18882.)

4.55 traceback

A new `traceback.clear_frames()` function takes a traceback object and clears the local variables in all of the frames it references, reducing the amount of memory consumed. (Contributed by Andrew Kuchling in issue 1565525.)

4.56 types

A new `DynamicClassAttribute()` descriptor provides a way to define an attribute that acts normally when looked up through an instance object, but which is routed to the *class* `__getattr__` when looked up through the class. This allows one to have properties active on a class, and have virtual attributes on the class with the same name (see `Enum` for an example). (Contributed by Ethan Furman in issue 19030.)

4.57 urllib

`urllib.request` now supports `data:` URLs via the `DataHandler` class. (Contributed by Mathias Panzenböck in issue 16423.)

The http method that will be used by a `Request` class can now be specified by setting a `method` class attribute on the subclass. (Contributed by Jason R Coombs in issue 18978.)

`Request` objects are now reusable: if the `full_url` or `data` attributes are modified, all relevant internal properties are updated. This means, for example, that it is now possible to use the same `Request` object in more than one `OpenerDirector.open()` call with different *data* arguments, or to modify a `Request`'s `url` rather than recomputing it from scratch. There is also a new `remove_header()` method that can be used to remove headers from a `Request`. (Contributed by Alexey Kachayev in issue 16464, Daniel Wozniak in issue 17485, and Damien Brecht and Senthil Kumaran in issue 17272.)

`HTTPError` objects now have a `headers` attribute that provides access to the HTTP response headers associated with the error. (Contributed by Berker Peksag in issue 15701.)

4.58 unittest

The `TestCase` class has a new method, `subTest()`, that produces a context manager whose `with` block becomes a "sub-test". This context manager allows a test method to dynamically generate subtests by, say, calling the `subTest` context manager inside a loop. A single test method can thereby produce an indefinite number of separately-identified and separately-counted tests, all of which will run even if one or more of them fail. For example:

```
class NumbersTest(unittest.TestCase):
    def test_even(self):
        for i in range(6):
            with self.subTest(i=i):
                self.assertEqual(i % 2, 0)
```

will result in six subtests, each identified in the unittest verbose output with a label consisting of the variable name `i` and a particular value for that variable (`i=0`, `i=1`, etc). See *subtests* for the full version of this example. (Contributed by Antoine Pitrou in issue 16997.)

`unittest.main()` now accepts an iterable of test names for *defaultTest*, where previously it only accepted a single test name as a string. (Contributed by Jyrki Pulliainen in issue 15132.)

If `SkipTest` is raised during test discovery (that is, at the module level in the test file), it is now reported as a skip instead of an error. (Contributed by Zach Ware in issue 16935.)

`discover()` now sorts the discovered files to provide consistent test ordering. (Contributed by Martin Melin and Jeff Ramnani in issue 16709.)

`TestSuite` now drops references to tests as soon as the test has been run, if the test is successful. On Python interpreters that do garbage collection, this allows the tests to be garbage collected if nothing else is holding a reference to the test. It is possible to override this behavior by creating a `TestSuite` subclass that defines a custom `_removeTestAtIndex` method. (Contributed by Tom Wardill, Matt McClure, and Andrew Svetlov in issue 11798.)

A new test assertion context-manager, `assertLogs()`, will ensure that a given block of code emits a log message using the `logging` module. By default the message can come from any logger and have a priority of `INFO` or higher, but both the logger name and an alternative minimum logging level may be specified. The object returned by the context manager can be queried for the `LogRecord`s and/or formatted messages that were logged. (Contributed by Antoine Pitrou in issue 18937.)

Test discovery now works with namespace packages (Contributed by Claudiu Popa in issue 17457.)

`unittest.mock` objects now inspect their specification signatures when matching calls, which means an argument can now be matched by either position or name, instead of only by position. (Contributed by Antoine Pitrou in issue 17015.)

`mock_open()` objects now have `readline` and `readlines` methods. (Contributed by Toshio Kuratomi in issue 17467.)

4.59 venv

`venv` now includes activation scripts for the `csh` and `fish` shells. (Contributed by Andrew Svetlov in issue 15417.)

`EnvBuilder` and the `create()` convenience function take a new keyword argument *with_pip*, which defaults to `False`, that controls whether or not `EnvBuilder` ensures that `pip` is installed in the virtual environment. (Contributed by Nick Coghlan in issue 19552 as part of the **PEP 453** implementation.)

4.60 wave

The `getparams()` method now returns a namedtuple rather than a plain tuple. (Contributed by Claudiu Popa in issue 17487.)

`wave.open()` now supports the context management protocol. (Contributed by Claudiu Popa in issue 17616.)

`wave` can now *write output to unseekable files*. (Contributed by David Jones, Guilherme Polo, and Serhiy Storchaka in issue 5202.)

The `writeframesraw()` and `writeframes()` methods now accept any *bytes-like object*. (Contributed by Serhiy Storchaka in issue 8311.)

4.61 weakref

New `WeakMethod` class simulates weak references to bound methods. (Contributed by Antoine Pitrou in issue 14631.)

New `finalize` class makes it possible to register a callback to be invoked when an object is garbage collected, without needing to carefully manage the lifecycle of the weak reference itself. (Contributed by Richard Oudkerk in issue 15528.)

The callback, if any, associated with a `ref` is now exposed via the `__callback__` attribute. (Contributed by Mark Dickinson in issue 17643.)

4.62 xml.etree

A new parser, `XMLPullParser`, allows a non-blocking applications to parse XML documents. An example can be seen at *elementtree-pull-parsing*. (Contributed by Antoine Pitrou in issue 17741.)

The `xml.etree.ElementTree` `tostring()` and `tostringlist()` functions, and the `ElementTree` `write()` method, now have a *short_empty_elements keyword-only parameter* providing control over whether elements with no content are written in abbreviated (`<tag />`) or expanded (`<tag></tag>`) form. (Contributed by Ariel Poliak and Serhiy Storchaka in issue 14377.)

4.63 zipfile

The `writepy()` method of the `PyZipFile` class has a new *filterfunc* option that can be used to control which directories and files are added to the archive. For example, this could be used to exclude test files from the archive. (Contributed by Christian Tismer in issue 19274.)

The *allowZip64* parameter to `ZipFile` and `PyZipfile` is now `True` by default. (Contributed by William Mallard in issue 17201.)

5 CPython Implementation Changes

5.1 PEP 445: Customization of CPython Memory Allocators

PEP 445 adds new C level interfaces to customize memory allocation in the CPython interpreter.

See also:

PEP 445 – Add new APIs to customize Python memory allocators PEP written and implemented by Victor Stinner.

5.2 PEP 442: Safe Object Finalization

PEP 442 removes the current limitations and quirks of object finalization in CPython. With it, objects with `__del__()` methods, as well as generators with `finally` clauses, can be finalized when they are part of a reference cycle.

As part of this change, module globals are no longer forcibly set to `None` during interpreter shutdown in most cases, instead relying on the normal operation of the cyclic garbage collector. This avoids a whole class of interpreter-shutdown-time errors, usually involving `__del__` methods, that have plagued Python since the cyclic GC was first introduced.

See also:

PEP 442 – Safe object finalization PEP written and implemented by Antoine Pitrou.

5.3 PEP 456: Secure and Interchangeable Hash Algorithm

PEP 456 follows up on earlier security fix work done on Python's hash algorithm to address certain DOS attacks to which public facing APIs backed by dictionary lookups may be subject. (See issue 14621 for the start of the current round of improvements.) The PEP unifies CPython's hash code to make it easier for a packager to substitute a different hash algorithm, and switches Python's default implementation to a SipHash implementation on platforms that have a 64 bit data type. Any performance differences in comparison with the older FNV algorithm are trivial.

The PEP adds additional fields to the `sys.hash_info` struct sequence to describe the hash algorithm in use by the currently executing binary. Otherwise, the PEP does not alter any existing CPython APIs.

5.4 PEP 436: Argument Clinic

"Argument Clinic" (**PEP 436**) is now part of the CPython build process and can be used to simplify the process of defining and maintaining accurate signatures for builtins and standard library extension modules implemented in C.

Some standard library extension modules have been converted to use Argument Clinic in Python 3.4, and `pydoc` and `inspect` have been updated accordingly.

It is expected that signature metadata for programmatic introspection will be added to additional callables implemented in C as part of Python 3.4 maintenance releases.

Note: The Argument Clinic PEP is not fully up to date with the state of the implementation. This has been deemed acceptable by the release manager and core development team in this case, as Argument Clinic will not be made available as a public API for third party use in Python 3.4.

See also:

PEP 436 – The Argument Clinic DSL PEP written and implemented by Larry Hastings.

5.5 Other Build and C API Changes

- The new `PyType_GetSlot()` function has been added to the stable ABI, allowing retrieval of function pointers from named type slots when using the limited API. (Contributed by Martin von Löwis in issue 17162.)
- The new `Py_SetStandardStreamEncoding()` pre-initialization API allows applications embedding the CPython interpreter to reliably force a particular encoding and error handler for the standard streams. (Contributed by Bastien Montagne and Nick Coghlan in issue 16129.)
- Most Python C APIs that don't mutate string arguments are now correctly marked as accepting `const char *` rather than `char *`. (Contributed by Serhiy Storchaka in issue 1772673.)
- A new shell version of `python-config` can be used even when a python interpreter is not available (for example, in cross compilation scenarios).
- `PyUnicode_FromFormat()` now supports width and precision specifications for `%s`, `%A`, `%U`, `%V`, `%S`, and `%R`. (Contributed by Ysj Ray and Victor Stinner in issue 7330.)
- New function `PyStructSequence_InitType2()` supplements the existing `PyStructSequence_InitType()` function. The difference is that it returns `0` on success and `-1` on failure.
- The CPython source can now be compiled using the address sanity checking features of recent versions of GCC and clang: the false alarms in the small object allocator have been silenced. (Contributed by Dhiru Kholia in issue 18596.)
- The Windows build now uses Address Space Layout Randomization and Data Execution Prevention. (Contributed by Christian Heimes in issue 16632.)
- New function `PyObject_LengthHint()` is the C API equivalent of `operator.length_hint()`. (Contributed by Armin Ronacher in issue 16148.)

5.6 Other Improvements

- The *python* command has a new *option*, `-I`, which causes it to run in "isolated mode", which means that `sys.path` contains neither the script's directory nor the user's `site-packages` directory, and all `PYTHON*` environment variables are ignored (it implies both `-s` and `-E`). Other restrictions may also be applied in the future, with the goal being to isolate the execution of a script from the user's environment. This is appropriate, for example, when Python is used to run a system script. On most POSIX systems it can and should be used in the `#!` line of system scripts. (Contributed by Christian Heimes in issue 16499.)

- Tab-completion is now enabled by default in the interactive interpreter on systems that support `readline`. History is also enabled by default, and is written to (and read from) the file `~/.python-history`. (Contributed by Antoine Pitrou and Éric Araujo in issue 5845.)
- Invoking the Python interpreter with `--version` now outputs the version to standard output instead of standard error (issue 18338). Similar changes were made to `argparse` (issue 18920) and other modules that have script-like invocation capabilities (issue 18922).
- The CPython Windows installer now adds `.py` to the `PATHEXT` variable when extensions are registered, allowing users to run a python script at the windows command prompt by just typing its name without the `.py` extension. (Contributed by Paul Moore in issue 18569.)
- A new `make` target coverage-report will build python, run the test suite, and generate an HTML coverage report for the C codebase using `gcov` and lcov.
- The `-R` option to the *python regression test suite* now also checks for memory allocation leaks, using `sys.getallocatedblocks()`. (Contributed by Antoine Pitrou in issue 13390.)
- `python -m` now works with namespace packages.
- The `stat` module is now implemented in C, which means it gets the values for its constants from the C header files, instead of having the values hard-coded in the python module as was previously the case.
- Loading multiple python modules from a single OS module (`.so`, `.dll`) now works correctly (previously it silently returned the first python module in the file). (Contributed by Václav Šmilauer in issue 16421.)
- A new opcode, `LOAD_CLASSDEREF`, has been added to fix a bug in the loading of free variables in class bodies that could be triggered by certain uses of *__prepare__*. (Contributed by Benjamin Peterson in issue 17853.)
- A number of MemoryError-related crashes were identified and fixed by Victor Stinner using his **PEP 445**-based `pyfailmalloc` tool (issue 18408, issue 18520).
- The *pyvenv* command now accepts a `--copies` option to use copies rather than symlinks even on systems where symlinks are the default. (Contributed by Vinay Sajip in issue 18807.)
- The *pyvenv* command also accepts a `--without-pip` option to suppress the otherwise-automatic bootstrapping of pip into the virtual environment. (Contributed by Nick Coghlan in issue 19552 as part of the **PEP 453** implementation.)
- The encoding name is now optional in the value set for the `PYTHONIOENCODING` environment variable. This makes it possible to set just the error handler, without changing the default encoding. (Contributed by Serhiy Storchaka in issue 18818.)
- The `bz2`, `lzma`, and `gzip` module `open` functions now support `x` (exclusive creation) mode. (Contributed by Tim Heaney and Vajrasky Kok in issue 19201, issue 19222, and issue 19223.)

5.7 Significant Optimizations

- The UTF-32 decoder is now 3x to 4x faster. (Contributed by Serhiy Storchaka in issue 14625.)
- The cost of hash collisions for sets is now reduced. Each hash table probe now checks a series of consecutive, adjacent key/hash pairs before continuing to make random probes through the hash table. This exploits cache locality to make collision resolution less expensive. The collision resolution scheme can be described as a hybrid of linear probing and open addressing. The number of additional linear probes defaults to nine. This can be changed at compile-time by defining LINEAR_PROBES to be any value. Set LINEAR_PROBES=0 to turn-off linear probing entirely. (Contributed by Raymond Hettinger in issue 18771.)
- The interpreter starts about 30% faster. A couple of measures lead to the speedup. The interpreter loads fewer modules on startup, e.g. the `re`, `collections` and `locale` modules and their dependencies are no longer imported by default. The marshal module has been improved to load compiled Python code faster. (Contributed by Antoine Pitrou, Christian Heimes and Victor Stinner in issue 19219, issue 19218, issue 19209, issue 19205 and issue 9548.)

- `bz2.BZ2File` is now as fast or faster than the Python2 version for most cases. `lzma.LZMAFile` has also been optimized. (Contributed by Serhiy Storchaka and Nadeem Vawda in issue 16034.)
- `random.getrandbits()` is 20%-40% faster for small integers (the most common use case). (Contributed by Serhiy Storchaka in issue 16674.)
- By taking advantage of the new storage format for strings, pickling of strings is now significantly faster. (Contributed by Victor Stinner and Antoine Pitrou in issue 15596.)
- A performance issue in `io.FileIO.readall()` has been solved. This particularly affects Windows, and significantly speeds up the case of piping significant amounts of data through `subprocess`. (Contributed by Richard Oudkerk in issue 15758.)
- `html.escape()` is now 10x faster. (Contributed by Matt Bryant in issue 18020.)
- On Windows, the native `VirtualAlloc` is now used instead of the CRT `malloc` in `obmalloc`. Artificial benchmarks show about a 3% memory savings.
- `os.urandom()` now uses a lazily-opened persistent file descriptor so as to avoid using many file descriptors when run in parallel from multiple threads. (Contributed by Antoine Pitrou in issue 18756.)

6 Deprecated

This section covers various APIs and other features that have been deprecated in Python 3.4, and will be removed in Python 3.5 or later. In most (but not all) cases, using the deprecated APIs will produce a `DeprecationWarning` when the interpreter is run with deprecation warnings enabled (for example, by using `-Wd`).

6.1 Deprecations in the Python API

- As mentioned in *PEP 451: A ModuleSpec Type for the Import System*, a number of `importilb` methods and functions are deprecated: `importlib.find_loader()` is replaced by `importlib.util.find_spec()`; `importlib.machinery.PathFinder.find_module()` is replaced by `importlib.machinery.PathFinder.find_spec()`; `importlib.abc.MetaPathFinder.find_module()` is replaced by `importlib.abc.MetaPathFinder.find_spec()`; `importlib.abc.PathEntryFinder.find_loader()` and `find_module()` are replaced by `importlib.abc.PathEntryFinder.find_spec()`; all of the `xxxLoader` ABC `load_module` methods (`importlib.abc.Loader.load_module()`, `importlib.abc.InspectLoader.load_module()`, `importlib.abc.FileLoader.load_module()`, `importlib.abc.SourceLoader.load_module()`) should no longer be implemented, instead loaders should implement an `exec_module` method (`importlib.abc.Loader.exec_module()`, `importlib.abc.InspectLoader.exec_module()` `importlib.abc.SourceLoader.exec_module()`) and let the import system take care of the rest; and `importlib.abc.Loader.module_repr()`, `importlib.util.module_for_loader()`, `importlib.util.set_loader()`, and `importlib.util.set_package()` are no longer needed because their functions are now handled automatically by the import system.
- The `imp` module is pending deprecation. To keep compatibility with Python 2/3 code bases, the module's removal is currently not scheduled.
- The `formatter` module is pending deprecation and is slated for removal in Python 3.6.
- `MD5` as the default *digestmod* for the `hmac.new()` function is deprecated. Python 3.6 will require an explicit digest name or constructor as *digestmod* argument.
- The internal `Netrc` class in the `ftplib` module has been documented as deprecated in its docstring for quite some time. It now emits a `DeprecationWarning` and will be removed completely in Python 3.5.
- The undocumented *endtime* argument to `subprocess.Popen.wait()` should not have been exposed and is hopefully not in use; it is deprecated and will mostly likely be removed in Python 3.5.
- The *strict* argument of `HTMLParser` is deprecated.

- The `plistlib` `readPlist()`, `writePlist()`, `readPlistFromBytes()`, and `writePlistToBytes()` functions are deprecated in favor of the corresponding new functions `load()`, `dump()`, `loads()`, and `dumps()`. `Data()` is deprecated in favor of just using the `bytes` constructor.
- The `sysconfig` key `SO` is deprecated, it has been replaced by `EXT_SUFFIX`.
- The `U` mode accepted by various `open` functions is deprecated. In Python3 it does not do anything useful, and should be replaced by appropriate uses of `io.TextIOWrapper` (if needed) and its *newline* argument.
- The *parser* argument of `xml.etree.ElementTree.iterparse()` has been deprecated, as has the *html* argument of `XMLParser()`. To prepare for the removal of the latter, all arguments to `XMLParser` should be passed by keyword.

6.2 Deprecated Features

- Running *idle* with the `-n` flag (no subprocess) is deprecated. However, the feature will not be removed until issue 18823 is resolved.
- The site module adding a "site-python" directory to sys.path, if it exists, is deprecated (issue 19375).

7 Removed

7.1 Operating Systems No Longer Supported

Support for the following operating systems has been removed from the source and build tools:

- OS/2 (issue 16135).
- Windows 2000 (changeset e52df05b496a).
- Windows systems where `COMSPEC` points to `command.com` (issue 14470).
- VMS (issue 16136).

7.2 API and Feature Removals

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

- The unmaintained `Misc/TextMate` and `Misc/vim` directories have been removed (see the devguide for suggestions on what to use instead).
- The `SO` makefile macro is removed (it was replaced by the `SHLIB_SUFFIX` and `EXT_SUFFIX` macros) (issue 16754).
- The `PyThreadState.tick_counter` field has been removed; its value has been meaningless since Python 3.2, when the "new GIL" was introduced (issue 19199).
- `PyLoader` and `PyPycLoader` have been removed from `importlib`. (Contributed by Taras Lyapun in issue 15641.)
- The *strict* argument to `HTTPConnection` and `HTTPSConnection` has been removed. HTTP 0.9-style "Simple Responses" are no longer supported.
- The deprecated `urllib.request.Request` getter and setter methods `add_data`, `has_data`, `get_data`, `get_type`, `get_host`, `get_selector`, `set_proxy`, `get_origin_req_host`, and `is_unverifiable` have been removed (use direct attribute access instead).
- Support for loading the deprecated `TYPE_INT64` has been removed from `marshal`. (Contributed by Dan Riti in issue 15480.)
- `inspect.Signature`: positional-only parameters are now required to have a valid name.

- `object.__format__()` no longer accepts non-empty format strings, it now raises a `TypeError` instead. Using a non-empty string has been deprecated since Python 3.2. This change has been made to prevent a situation where previously working (but incorrect) code would start failing if an object gained a __format__ method, which means that your code may now raise a `TypeError` if you are using an `'s'` format code with objects that do not have a __format__ method that handles it. See issue 7994 for background.
- `difflib.SequenceMatcher.isbjunk()` and `difflib.SequenceMatcher.isbpopular()` were deprecated in 3.2, and have now been removed: use `x in sm.bjunk` and `x in sm.bpopular`, where *sm* is a `SequenceMatcher` object (issue 13248).

7.3 Code Cleanups

- The unused and undocumented internal `Scanner` class has been removed from the `pydoc` module.
- The private and effectively unused `_gestalt` module has been removed, along with the private `platform` functions `_mac_ver_lookup`, `_mac_ver_gstalt`, and `_bcd2str`, which would only have ever been called on badly broken OSX systems (see issue 18393).
- The hardcoded copies of certain `stat` constants that were included in the `tarfile` module namespace have been removed.

8 Porting to Python 3.4

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

8.1 Changes in 'python' Command Behavior

- In a posix shell, setting the `PATH` environment variable to an empty value is equivalent to not setting it at all. However, setting `PYTHONPATH` to an empty value was *not* equivalent to not setting it at all: setting `PYTHONPATH` to an empty value was equivalent to setting it to `.`, which leads to confusion when reasoning by analogy to how `PATH` works. The behavior now conforms to the posix convention for `PATH`.
- The [X refs, Y blocks] output of a debug (`--with-pydebug`) build of the CPython interpreter is now off by default. It can be re-enabled using the `-X showrefcount` option. (Contributed by Ezio Melotti in issue 17323.)
- The python command and most stdlib scripts (as well as `argparse`) now output `--version` information to `stdout` instead of `stderr` (for issue list see *Other Improvements* above).

8.2 Changes in the Python API

- The ABCs defined in `importlib.abc` now either raise the appropriate exception or return a default value instead of raising `NotImplementedError` blindly. This will only affect code calling `super()` and falling through all the way to the ABCs. For compatibility, catch both `NotImplementedError` or the appropriate exception as needed.
- The module type now initializes the `__package__` and `__loader__` attributes to `None` by default. To determine if these attributes were set in a backwards-compatible fashion, use e.g. `getattr(module, '__loader__', None) is not None`. (issue 17115.)
- `importlib.util.module_for_loader()` now sets `__loader__` and `__package__` unconditionally to properly support reloading. If this is not desired then you will need to set these attributes manually. You can use `importlib.util.module_to_load()` for module management.
- Import now resets relevant attributes (e.g. `__name__`, `__loader__`, `__package__`, `__file__`, `__cached__`) unconditionally when reloading. Note that this restores a pre-3.3 behavior in that it means a module is re-found when re-loaded (issue 19413).

- Frozen packages no longer set `__path__` to a list containing the package name, they now set it to an empty list. The previous behavior could cause the import system to do the wrong thing on submodule imports if there was also a directory with the same name as the frozen package. The correct way to determine if a module is a package or not is to use `hasattr(module, '__path__')` (issue 18065).
- Frozen modules no longer define a `__file__` attribute. It's semantically incorrect for frozen modules to set the attribute as they are not loaded from any explicit location. If you must know that a module comes from frozen code then you can see if the module's `__spec__.location` is set to `'frozen'`, check if the loader is a subclass of `importlib.machinery.FrozenImporter`, or if Python 2 compatibility is necessary you can use `imp.is_frozen()`.
- `py_compile.compile()` now raises `FileExistsError` if the file path it would write to is a symlink or a non-regular file. This is to act as a warning that import will overwrite those files with a regular file regardless of what type of file path they were originally.
- `importlib.abc.SourceLoader.get_source()` no longer raises `ImportError` when the source code being loaded triggers a `SyntaxError` or `UnicodeDecodeError`. As `ImportError` is meant to be raised only when source code cannot be found but it should, it was felt to be over-reaching/overloading of that meaning when the source code is found but improperly structured. If you were catching ImportError before and wish to continue to ignore syntax or decoding issues, catch all three exceptions now.
- `functools.update_wrapper()` and `functools.wraps()` now correctly set the `__wrapped__` attribute to the function being wrapped, even if that function also had its `__wrapped__` attribute set. This means `__wrapped__` attributes now correctly link a stack of decorated functions rather than every `__wrapped__` attribute in the chain referring to the innermost function. Introspection libraries that assumed the previous behaviour was intentional can use `inspect.unwrap()` to access the first function in the chain that has no `__wrapped__` attribute.
- `inspect.getfullargspec()` has been reimplemented on top of `inspect.signature()` and hence handles a much wider variety of callable objects than it did in the past. It is expected that additional builtin and extension module callables will gain signature metadata over the course of the Python 3.4 series. Code that assumes that `inspect.getfullargspec()` will fail on non-Python callables may need to be adjusted accordingly.
- `importlib.machinery.PathFinder` now passes on the current working directory to objects in `sys.path_hooks` for the empty string. This results in `sys.path_importer_cache` never containing `''`, thus iterating through `sys.path_importer_cache` based on `sys.path` will not find all keys. A module's `__file__` when imported in the current working directory will also now have an absolute path, including when using `-m` with the interpreter (except for `__main__.__file__` when a script has been executed directly using a relative path) (Contributed by Brett Cannon in issue 18416). is specified on the command-line) (issue 18416).
- The removal of the *strict* argument to `HTTPConnection` and `HTTPSConnection` changes the meaning of the remaining arguments if you are specifying them positionally rather than by keyword. If you've been paying attention to deprecation warnings your code should already be specifying any additional arguments via keywords.
- Strings between `from __future__ import ...` statements now *always* raise a `SyntaxError`. Previously if there was no leading docstring, an interstitial string would sometimes be ignored. This brings CPython into compliance with the language spec; Jython and PyPy already were. (issue 17434).
- `ssl.SSLSocket.getpeercert()` and `ssl.SSLSocket.do_handshake()` now raise an `OSError` with `ENOTCONN` when the `SSLSocket` is not connected, instead of the previous behavior of raising an `AttributeError`. In addition, `getpeercert()` will raise a `ValueError` if the handshake has not yet been done.
- `base64.b32decode()` now raises a `binascii.Error` when the input string contains non-b32-alphabet characters, instead of a `TypeError`. This particular `TypeError` was missed when the other `TypeErrors` were converted. (Contributed by Serhiy Storchaka in issue 18011.) Note: this change was also inadvertently applied in Python 3.3.3.
- The `file` attribute is now automatically closed when the creating `cgi.FieldStorage` instance is garbage collected. If you were pulling the file object out separately from the `cgi.FieldStorage` in-

stance and not keeping the instance alive, then you should either store the entire `cgi.FieldStorage` instance or read the contents of the file before the `cgi.FieldStorage` instance is garbage collected.

- Calling `read` or `write` on a closed SSL socket now raises an informative `ValueError` rather than the previous more mysterious `AttributeError` (issue 9177).
- `slice.indices()` no longer produces an `OverflowError` for huge values. As a consequence of this fix, `slice.indices()` now raises a `ValueError` if given a negative length; previously it returned nonsense values (issue 14794).
- The `complex` constructor, unlike the `cmath` functions, was incorrectly accepting `float` values if an object's `__complex__` special method returned one. This now raises a `TypeError`. (issue 16290.)
- The `int` constructor in 3.2 and 3.3 erroneously accepts `float` values for the *base* parameter. It is unlikely anyone was doing this, but if so, it will now raise a `TypeError` (issue 16772).
- Defaults for keyword-only arguments are now evaluated *after* defaults for regular keyword arguments, instead of before. Hopefully no one wrote any code that depends on the previous buggy behavior (issue 16967).
- Stale thread states are now cleared after `fork()`. This may cause some system resources to be released that previously were incorrectly kept perpetually alive (for example, database connections kept in thread-local storage). (issue 17094.)
- Parameter names in `__annotations__` dicts are now mangled properly, similarly to `__kwdefaults__`. (Contributed by Yury Selivanov in issue 20625.)
- `hashlib.hash.name` now always returns the identifier in lower case. Previously some builtin hashes had uppercase names, but now that it is a formal public interface the naming has been made consistent (issue 18532).
- Because `unittest.TestSuite` now drops references to tests after they are run, test harnesses that re-use a `TestSuite` to re-run a set of tests may fail. Test suites should not be re-used in this fashion since it means state is retained between test runs, breaking the test isolation that `unittest` is designed to provide. However, if the lack of isolation is considered acceptable, the old behavior can be restored by creating a `TestSuite` subclass that defines a `_removeTestAtIndex` method that does nothing (see `TestSuite.__iter__()`) (issue 11798).
- `unittest` now uses `argparse` for command line parsing. There are certain invalid command forms that used to work that are no longer allowed; in theory this should not cause backward compatibility issues since the disallowed command forms didn't make any sense and are unlikely to be in use.
- The `re.split()`, `re.findall()`, and `re.sub()` functions, and the `group()` and `groups()` methods of `match` objects now always return a *bytes* object when the string to be matched is a *bytes-like object*. Previously the return type matched the input type, so if your code was depending on the return value being, say, a `bytearray`, you will need to change your code.
- `audioop` functions now raise an error immediately if passed string input, instead of failing randomly later on (issue 16685).
- The new *convert_charrefs* argument to `HTMLParser` currently defaults to `False` for backward compatibility, but will eventually be changed to default to `True`. It is recommended that you add this keyword, with the appropriate value, to any `HTMLParser` calls in your code (issue 13633).
- Since the *digestmod* argument to the `hmac.new()` function will in the future have no default, all calls to `hmac.new()` should be changed to explicitly specify a *digestmod* (issue 17276).
- Calling `sysconfig.get_config_var()` with the `SO` key, or looking `SO` up in the results of a call to `sysconfig.get_config_vars()` is deprecated. This key should be replaced by `EXT_SUFFIX` or `SHLIB_SUFFIX`, depending on the context (issue 19555).
- Any calls to `open` functions that specify `U` should be modified. `U` is ineffective in Python3 and will eventually raise an error if used. Depending on the function, the equivalent of its old Python2 behavior can be achieved using either a *newline* argument, or if necessary by wrapping the stream in `TextIOWrapper` to use its *newline* argument (issue 15204).

- If you use *pyvenv* in a script and desire that pip *not* be installed, you must add `--without-pip` to your command invocation.
- The default behavior of `json.dump()` and `json.dumps()` when an indent is specified has changed: it no longer produces trailing spaces after the item separating commas at the ends of lines. This will matter only if you have tests that are doing white-space-sensitive comparisons of such output (issue 16333).
- `doctest` now looks for doctests in extension module `__doc__` strings, so if your doctest test discovery includes extension modules that have things that look like doctests in them you may see test failures you've never seen before when running your tests (issue 3158).
- The `collections.abc` module has been slightly refactored as part of the Python startup improvements. As a consequence of this, it is no longer the case that importing `collections` automatically imports `collections.abc`. If your program depended on the (undocumented) implicit import, you will need to add an explicit `import collections.abc` (issue 20784).

8.3 Changes in the C API

- `PyEval_EvalFrameEx()`, `PyObject_Repr()`, and `PyObject_Str()`, along with some other internal C APIs, now include a debugging assertion that ensures they are not used in situations where they may silently discard a currently active exception. In cases where discarding the active exception is expected and desired (for example, because it has already been saved locally with `PyErr_Fetch()` or is being deliberately replaced with a different exception), an explicit `PyErr_Clear()` call will be needed to avoid triggering the assertion when invoking these operations (directly or indirectly) and running against a version of Python that is compiled with assertions enabled.
- `PyErr_SetImportError()` now sets `TypeError` when its **msg** argument is not set. Previously only `NULL` was returned with no exception set.
- The result of the `PyOS_ReadlineFunctionPointer` callback must now be a string allocated by `PyMem_RawMalloc()` or `PyMem_RawRealloc()`, or *NULL* if an error occurred, instead of a string allocated by `PyMem_Malloc()` or `PyMem_Realloc()` (issue 16742)
- `PyThread_set_key_value()` now always set the value. In Python 3.3, the function did nothing if the key already exists (if the current value is a non-NULL pointer).
- The `f_tstate` (thread state) field of the `PyFrameObject` structure has been removed to fix a bug: see issue 14432 for the rationale.

9 Changed in 3.4.3

9.1 PEP 476: Enabling certificate verification by default for stdlib http clients

`http.client` and modules which use it, such as `urllib.request` and `xmlrpc.client`, will now verify that the server presents a certificate which is signed by a CA in the platform trust store and whose hostname matches the hostname being requested by default, significantly improving security for many applications.

For applications which require the old previous behavior, they can pass an alternate context:

```
import urllib.request
import ssl

# This disables all verification
context = ssl._create_unverified_context()

# This allows using a specific certificate for the host, which doesn't need
# to be in the trust store
context = ssl.create_default_context(cafile="/path/to/file.crt")

urllib.request.urlopen("https://invalid-cert", context=context)
```

Index

E

P