And the `ConnectionError` itself has finer-grained subclasses:

- `BrokenPipeError`
- `ConnectionAbortedError`
- `ConnectionRefusedError`
- `ConnectionResetError`

Thanks to the new exceptions, common usages of the `errno` can now be avoided. For example, the following code written for Python 3.2:

```
from errno import ENOENT, EACCES, EPERM

try:
    with open("document.txt") as f:
        content = f.read()
except IOError as err:
    if err.errno == ENOENT:
        print("document.txt file is missing")
    elif err.errno in (EACCES, EPERM):
        print("You are not allowed to read document.txt")
    else:
        raise
```

can now be written without the `errno` import and without manual inspection of exception attributes:

```
try:
    with open("document.txt") as f:
        content = f.read()
except FileNotFoundError:
    print("document.txt file is missing")
except PermissionError:
    print("You are not allowed to read document.txt")
```

See also:

PEP 3151 - Reworking the OS and IO Exception Hierarchy
PEP written and implemented by Antoine Pitrou

PEP 380: Syntax for Delegating to a Subgenerator

PEP 380 adds the `yield from` expression, allowing a generator to delegate part of its operations to another generator. This allows a section of code containing `yield` to be factored out and placed in another generator. Additionally, the subgenerator is allowed to return with a value, and the value is made available to the delegating generator.

While designed primarily for use in delegating to a subgenerator, the `yield from` expression actually allows delegation to arbitrary subiterators.

For simple iterators, `yield from iterable` is essentially just a shortened form of `for item in iterable: yield item`:

```
>>> def g(x):
...     yield from range(x, 0, -1)
...     yield from range(x)
...
>>> list(g(5))
[5, 4, 3, 2, 1, 0, 1, 2, 3, 4]
```

However, unlike an ordinary loop, `yield from` allows subgenerators to receive sent and thrown values directly from the calling scope, and return a final value to the outer generator:

```
>>> def accumulate():
...     tally = 0
...     while 1:
...         next = yield
...         if next is None:
...             return tally
...         tally += next
...
>>> def gather_tallies(tallies):
...     while 1:
...         tally = yield from accumulate()
...         tallies.append(tally)
...
>>> tallies = []
>>> acc = gather_tallies(tallies)
>>> next(acc)  # Ensure the accumulator is ready to accept values
>>> for i in range(4):
...     acc.send(i)
...
>>> acc.send(None)  # Finish the first tally
>>> for i in range(5):
...     acc.send(i)
...
>>> acc.send(None)  # Finish the second tally
>>> tallies
[6, 10]
```

The main principle driving this change is to allow even generators that are designed to be used with the `send` and `throw` methods to be split into multiple subgenerators as easily as a single large function can be split into multiple subfunctions.

See also:

PEP 380 - Syntax for Delegating to a Subgenerator
PEP written by Greg Ewing; implementation by Greg Ewing, integrated into 3.3 by Renaud Blanch, Ryan Kelly and Nick Coghlan; documentation by Zbigniew Jędrzejewski-Szmek and Nick Coghlan

PEP 409: Suppressing exception context

PEP 409 introduces new syntax that allows the display of the chained exception context to be disabled. This allows cleaner error messages in applications that convert between exception