5.2.10.1. Generator-iterator methods

This subsection describes the methods of a generator iterator. They can be used to control the execution of a generator function.

Note that calling any of the generator methods below when the generator is already executing raises a `ValueError`exception.

generator.**next**()

Starts the execution of a generator function or resumes it at the last executed `yield`expression. When a generator function is resumed with a `next()`method, the current `yield`expression always evaluates to`None`. The execution then continues to the next `yield`expression, where the generator is suspended again, and the value of the `expression_list` is returned to `next()`‘s caller. If the generator exits without yielding another value, a `StopIteration`exception is raised.

generator.**send**(*value*)

Resumes the execution and “sends” a value into the generator function. The `value`argument becomes the result of the current `yield` expression. The`send()` method returns the next value yielded by the generator, or raises `StopIteration` if the generator exits without yielding another value. When `send()` is called to start the generator, it must be called with `None` as the argument, because there is no`yield` expression that could receive the value.

generator.**throw**(*type*[, *value*[,*traceback*]])

Raises an exception of type `type`at the point where generator was paused, and returns the next value yielded by the generator function. If the generator exits without yielding another value, a`StopIteration` exception is raised. If the generator function does not catch the passed-in exception, or raises a different exception, then that exception propagates to the caller.

generator.**close**()

Raises a `GeneratorExit` at the point where the generator function was paused. If the generator function then raises`StopIteration` (by exiting normally, or due to already being closed) or `GeneratorExit` (by not catching the exception), close returns to its caller. If the generator yields a value, a `RuntimeError` is raised. If the generator raises any other exception, it is propagated to the caller. `close()` does nothing if the generator has already exited due to an exception or normal exit.

Here is a simple example that demonstrates the behavior of generators and generator functions:

```
>>> def echo(value=None):
...     print "Execution starts when 'next()' is called for the first time."
...     try:
...         while True:
...             try:
...                 value = (yield value)
...             except Exception, e:
...                 value = e
...     finally:
...         print "Don't forget to clean up when 'close()' is called."
...
>>> generator = echo(1)
>>> print generator.next()
```

```
Execution starts when 'next()' is called for the first time.
1
>>> print generator.next()
None
>>> print generator.send(2)
2
>>> generator.throw(TypeError, "spam")
TypeError('spam',)
>>> generator.close()
Don't forget to clean up when 'close()' is called.
```