вратить сами функции в генераторы вместо построителей списков, снова используйте круглые скобки вместо квадратных. Вот случай для myzip:

```
# Применение генераторов: (...)

def myzip(*seqs):
    minlen = min(len(S) for S in seqs)
    return (tuple(S[i] for S in seqs) for i in range(minlen))

S1, S2 = 'abc', 'xyz123'
print(list(myzip(S1, S2))) # Go!... [('a', 'x'), ('b', 'y'), ('c', 'z')]
```

Для активации генераторов и других итерируемых объектов с целью выпуска ими результатов в этом случае требуется вызов list. Чтобы получить больше сведений, самостоятельно поэкспериментируйте с кодом. Разработка дальнейших альтернативных версий оставляется в качестве упражнения (см. также врезку "Что потребует внимания: одноразовые итерации" ниже).

Дополнительные примеры применения оператора yield предлагаются в главе 30, где он будет использоваться в сочетании с методом перегрузки операции __iter__ для реализации определяемых пользователем итерируемых объектов в автоматическом режиме. Сохранение состояния локальных переменных в такой роли служит альтернативой атрибутам класса в том же духе, как функции замыканий из главы 17; однако, как будет показано, эта методика *комбинирует* классы и функциональные инструменты, а не выдвигает альтернативную парадигму.

Что потребует внимания: одноразовые итерации

В главе 14 мы видели, что некоторые встроенные функции (наподобие map) поддерживают одиночный обход, после чего опустошаются, и было обещано продемонстрировать пример того, как такой факт становится едва различимым, но важным на практике. Теперь, когда были проведены более глубокие исследования темы итерации, наступило время выполнить обещанное. Взгляните на следующую искусную альтернативную версию кода примеров эмуляции zip, рассмотренных в настоящей главе, которая взята из руководств по Python:

```
def myzip(*args):
    iters = map(iter, args)
    while iters:
        res = [next(i) for i in iters]
        yield tuple(res)
```

Поскольку в коде применяются iter и next, он работает с итерируемым объектом любого типа. Обратите внимание, что нет никаких причин перехватывать исключение StopIteration, инициируемое next(i) внутри спискового включения, когда израсходуются элементы в любом итераторе из аргументов — это дает генераторной функции возможность дойти до конца и обеспечивает такой же эффект, как имел бы оператор return. Оператора while iters: достаточно для выполнения цикла, если был передан, по меньшей мере, один аргумент, и избегания бесконечного цикла в противном случае (списковое включение всегда возвращало бы пустой список).

Приведенный код нормально работает в Python 2.X в том виде, как есть:

```
>>> list(myzip('abc', 'lmnop'))
[('a', 'l'), ('b', 'm'), ('c', 'n')]
```