```
>>> S1 = 'abc'
>>> S2 = 'xyz123'
>>> list(zip(S1, S2))                          # zip объединяет в пары элементы
                                               #   из итерируемых объектов
[('a', 'x'), ('b', 'y'), ('c', 'z')]

# zip объединяет элементы в пары, усекает до самого короткого
>>> list(zip([-2, -1, 0, 1, 2]))              # Единственная последовательность:
                                               #   1-арные кортежи
[(-2,), (-1,), (0,), (1,), (2,)]
>>> list(zip([1, 2, 3], [2, 3, 4, 5]))        # N последовательностей:
                                               #   N-арные кортежи
[(1, 2), (2, 3), (3, 4)]

# map передает объединенные в пары элементы функции, усекает
>>> list(map(abs, [-2, -1, 0, 1, 2]))         # Единственная последовательность:
                                               #   1-арная функция
[2, 1, 0, 1, 2]
>>> list(map(pow, [1, 2, 3], [2, 3, 4, 5]))   # N последовательностей:
                                               #   N-арная функция, Python 3.X
[1, 8, 81]

# map и zip принимают произвольные итерируемые объекты
>>> list(map(lambda x, y: x + y, open('script2.py'), open('script2.py')))
['import sys\nimport sys\n', 'print(sys.path)\nprint(sys.path)\n', ...и так далее...]

>>> [x + y for (x, y) in zip(open('script2.py'), open('script2.py'))]
['import sys\nimport sys\n', 'print(sys.path)\nprint(sys.path)\n', ...и так далее...]
```

Несмотря на использование для разных целей, если вы достаточно хорошо изучите приводимые примеры, то сможете заметить отношение между результатами `zip` и аргументами функции `map`, которое задействовано в следующем примере.

Написание собственной функции `map(func, ...)`

Хотя встроенные функции `map` и `zip` отличаются высокой скоростью и удобством, их всегда возможно эмулировать в собственном коде. Например, в предыдущей главе мы видели функцию, которая эмулировала встроенную функцию `map` для единственного аргумента в форме последовательности (или другого итерируемого объекта). Она требовала гораздо большей работы, чтобы разрешить передачу множества последовательностей, как делает встроенная функция:

```
# Аналог map(func, seqs...) на основе zip

def mymap(func, *seqs):
    res = []
    for args in zip(*seqs):
        res.append(func(*args))
    return res

print(mymap(abs, [-2, -1, 0, 1, 2]))
print(mymap(pow, [1, 2, 3], [2, 3, 4, 5]))
```

Версия `mymap` в значительной степени опирается на специальный синтаксис передачи аргументов `*аргументы` — она накапливает множество аргументов в виде последовательностей (в действительности итерируемых объектов), распаковывает их как аргументы `zip` для объединения и затем распаковывает результирующие пары `zip` как аргументы для переданной функции. То есть мы задействуем тот факт, что объединение в пары по существу представляет собой вложенную операцию в отображении.