wionych wersji funkcji `mymap` składają wynik w całości, co może mieć konsekwencje w postaci większego zapotrzebowania na pamięć. Dzięki możliwościom *funkcji i wyrażeń generatorów* możemy w prosty sposób przepisać te funkcje tak, aby zwracały wyniki na żądanie.

```
# Użycie generatorów: yield i (...)

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

def mymap(func, *seqs):
    return (func(*args) for args in zip(*seqs))
```

Te wersje nie tworzą całych wyników naraz, ale zwracają generatory obsługujące protokół iteracyjny: pierwsza wersja zwraca wyniki za pomocą instrukcji `yield`, a druga jest jej funkcjonalnym odpowiednikiem dzięki wyrażeniu generatora. Obie wersje zwrócą takie same wyniki, jeśli przekształcimy je na listy za pomocą funkcji `list`.

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

Jednak w tym przypadku różnica jest taka, że generatory nie wykonują żadnej pracy do momentu, gdy zostaje na nich wywołana funkcja `list` aktywująca protokół iteracyjny. Generatory zwrócone z tych funkcji, jak również wersja dla Pythona 3.0 wykorzystująca funkcję `zip`, generują wyniki na żądanie.

Własna wersja funkcji zip(...) i map(None, ...)

Oczywiście spora część magii powyższych przykładów dzieje się dzięki zastosowaniu funkcji wbudowanej `zip` łączącej elementy z kilku sekwencji. Warto również zauważyć, że nasze implementacje funkcji `map` w rzeczywistości kopiują zachowanie funkcji `map` z Pythona 3.0: przycinają wynik do długości najkrótszej sekwencji, zamiast uzupełniać brakujące elementy wartościami `None`, jak to się dzieje w Pythonie 2.X.

```
C:\misc> c:\python26\python
>>> map(None, [1, 2, 3], [2, 3, 4, 5])
[(1, 2), (2, 3), (3, 4), (None, 5)]
>>> map(None, 'abc', 'xyz123')
[('a', 'x'), ('b', 'y'), ('c', 'z'), (None, '1'), (None, '2'), (None, '3')]
```

Wykorzystując narzędzia iteracyjne, możemy zaimplementować własne wersje funkcji `zip` przycinającej wynik do najkrótszej sekwencji (dla 2.6) lub funkcję `map` uzupełniającą brakujące elementy sekwencji wartościami `None` (dla 3.0). Jak się okazuje, funkcje te będą bardzo do siebie podobne:

```
# Własne implementacje funkcji zip(seqs...) oraz map(None, seqs...) w wersji 2.6

def myzip(*seqs):
    seqs = [list(S) for S in seqs]
    res  = []
    while all(seqs):
        res.append(tuple(S.pop(0) for S in seqs))
    return res

def mymapPad(*seqs, pad=None):
    seqs = [list(S) for S in seqs]
    res  = []
    while any(seqs):
```