Taka sama sytuacja ma miejsce w przypadku funkcji generatorów — poniższy przykład prezentuje funkcję generatora, której iterator wyczerpuje się po pierwszym przejściu:

```
>>> def timesfour(S):
...     for c in S:
...         yield c * 4
...
>>> G = timesfour('jajko')          # Funkcje generatorów działają w ten sam sposób
>>> iter(G) is G
True
>>> I1, I2 = iter(G), iter(G)
>>> next(I1)
'jjjj'
>>> next(I1)
'aaaa'
>>> next(I2)                        # I2 znajduje się na tej samej pozycji co I1
'jjjj'
```

To znacząca różnica w porównaniu z niektórymi typami wbudowanymi, które obsługują wielokrotne iteracje i przejścia, a zmiany w ich wartości są odzwierciedlane w aktywnych iteratorach:

```
>>> L = [1, 2, 3, 4]
>>> I1, I2 = iter(L), iter(L)
>>> next(I1)
1
>>> next(I1)
2
>>> next(I2)                        # Listy obsługują wielokrotną iterację
1
>>> del L[2:]                       # Zmiany w liście są odzwierciedlane w iteratorach
>>> next(I1)
StopIteration
```

Gdy będziemy tworzyć własne iteratory w części VI, przekonamy się, że to od programisty zależy, ile równoległych iteracji chce obsłużyć, jeśli w ogóle.

Emulacja funkcji zip i map za pomocą narzędzi iteracyjnych

Aby zobaczyć możliwości narzędzi iteracyjnych, przyjrzyjmy się bardziej zaawansowanym przykładom użycia. Uzbrojeni w wiedzę o listach składanych, generatorach i innych narzędziach iteracyjnych możemy pokusić się o zaemulowanie niektórych rozszerzeń funkcyjnych Pythona. To zadanie okaże się o tyle proste co pouczające.

Mieliśmy już okazję zaobserwować, w jaki sposób funkcje `zip` i `map` przetwarzają obiekty iterowane z użyciem funkcji wykonywanych na ich elementach. W przypadku zastosowania kilku argumentów sekwencyjnych funkcja `map` wykonuje funkcje wieloargumentowe na kolejnych elementach każdej z list, natomiast `zip` łączy w nowe sekwencje odpowiadające sobie elementy każdej z list.

```
>>> S1 = 'abc'
>>> S2 = 'xyz123'
>>> list(zip(S1, S2))                          # zip łączy w pary elementy iteratorów
[('a', 'x'), ('b', 'y'), ('c', 'z')]

# zip łączy elementy i przycina wynik do najkrótszej sekwencji
>>> list(zip([ 2, -1, 0, 1, 2]))               # Jedna sekwencja: zwraca krotki 1-elementowe
[( 2,), ( 1,), (0,), (1,), (2,)]
```