подобно генераторным функциям они разделяют работу по выпуску результатов на небольшие *временные интервалы* – результаты выдаются постепенно вместо того, чтобы заставлять вызывающий код ожидать создания полного набора в единственном вызове.

С другой стороны, на практике генераторные выражения могут выполняться несколько медленнее списковых включений, а потому их лучше всего применять для очень крупных результирующих наборов или в приложениях, которые не могут ожидать генерации полных результатов. Но более авторитетное заявление о производительности придется отложить до написания сценариев для измерения времени выполнения в следующей главе.

Генераторные выражения тоже обладают преимуществами в плане *написания кода*, хотя и более субъективными, как показано в следующем разделе.

Генераторные выражения или map

Один из способов взглянуть на преимущества генераторных выражений в плане написания кода предусматривает их сравнение с другими инструментами для функционального программирования, как мы делали при исследовании списковых включений. Скажем, генераторные выражения часто эквивалентны вызовам `map` из Python 3.X, потому что в обоих случаях результирующие элементы генерируются по запросу. Тем не менее, аналогично списковым включениям генераторные выражения могут быть проще в написании, когда применяемой операцией оказывается не вызов функции. В Python 2.X функция `map` создает временные списки, а генераторные выражения – нет, но сравнение их кода все равно применимо:

```
>>> list(map(abs, (-1, -2, 3, 4)))            # Отображает функцию на кортеж
[1, 2, 3, 4]
>>> list(abs(x) for x in (-1, -2, 3, 4))       # Генераторное выражение
[1, 2, 3, 4]

>>> list(map(lambda x: x * 2, (1, 2, 3, 4)))   # Случай не функции
[2, 4, 6, 8]
>>> list(x * 2 for x in (1, 2, 3, 4))          # Проще как генератор?
[2, 4, 6, 8]
```

То же самое остается справедливым в сценариях использования для обработки текста вроде ранее показанного вызова `join` – списковое включение создает дополнительный временный список результатов, который в таком контексте будет совершенно бессмысленным, поскольку список не сохраняется, а `map` утрачивает преимущество простоты в сравнении с синтаксисом генераторных выражений, когда применяемая операция отличается от вызова:

```
>>> line = 'aaa,bbb,ccc'
>>> ''.join([x.upper() for x in line.split(',')])    # Создает бессмысленный
                                                     #   список
'AAABBBCCC'

>>> ''.join(x.upper() for x in line.split(','))      # Генерирует результаты
'AAABBBCCC'
>>> ''.join(map(str.upper, line.split(',')))         # Генерирует результаты
'AAABBBCCC'

>>> ''.join(x * 2 for x in line.split(','))          # Проще как генератор?
'aaaaaabbbbbbcccccc'
>>> ''.join(map(lambda x: x * 2, line.split(',')))
'aaaaaabbbbbbcccccc'
```