Вызовы map и генераторные выражения могут также быть произвольно *вложенными*, что общеупотребительно в программах и требует вызова list или другого итерационного контекста для инициирования процесса выпуска результатов. Например, приведенное далее списковое включение производит такой же результат, как показанные после него эквиваленты в форме map из Python 3.X и генераторов, но создает два физических списка; остальные генерируют только по одному целому числу за раз с помощью вложенных генераторов, а форма генераторного выражения может более четко отражать свое намерение:

```
>>> [x * 2 for x in [abs(x) for x in (-1, -2, 3, 4)]]   # Вложенные включения
[2, 4, 6, 8]
>>> list(map(lambda x: x * 2, map(abs, (-1, -2, 3, 4))))   # Вложенные отображения
[2, 4, 6, 8]
>>> list(x * 2 for x in (abs(x) for x in (-1, -2, 3, 4)))  # Вложенные генераторы
[2, 4, 6, 8]
```

Хотя результатом всех трех форм является объединение операций, генераторы делают это, не создавая множество временных списков. В Python 3.X следующий пример вкладывает *и* комбинирует генераторы – вложенное генераторное выражение активируется посредством функции map, которая в свою очередь активируется только функцией list:

```
>>> import math
>>> list(map(math.sqrt, (x ** 2 for x in range(4))))  # Вложенные комбинации
[0.0, 1.0, 2.0, 3.0]
```

Говоря формально, функция range справа в Python 3.X тоже представляет собой генератор значений, активируемый самим генераторным выражением – *три уровня* генерации значений, которые производят индивидуальные значения от внутреннего уровня к внешнему только по запросу и которые "просто работают" благодаря инструментам и протоколу итерации Python. На самом деле генераторы можно произвольно смешивать и вкладывать, несмотря на то, что одни могут быть более допустимыми, чем другие:

```
>>> list(map(abs, map(abs, map(abs, (-1, 0, 1)))))  # Вложение пришло в упадок?
[1, 0, 1]
>>> list(abs(x) for x in (abs(x) for x in (abs(x) for x in (-1, 0, 1))))
[1, 0, 1]
```

Последние примеры иллюстрируют, какими могут быть обычные генераторы, но они умышленно записаны в сложной форме, чтобы подчеркнуть тот факт, что генераторные выражения обладают аналогичным потенциалом злоупотребления, как и обсуждаемые ранее списковые включения – вы должны сохранять их простыми, если только они не обязаны быть сложными (позже в главе мы еще вернемся к данной теме).

Однако при правильном использовании генераторные выражения объединяют эффективность списковых включений с преимуществами, касающимися пространства и времени, других итерируемых объектов. Скажем, подходы без вложения предоставляют более простые решения, но по-прежнему задействуют сильные стороны генераторов – согласно девизу Python плоский в целом лучше, чем вложенный:

```
>>> list(abs(x) * 2 for x in (-1, -2, 3, 4))      # Эквиваленты без вложения
[2, 4, 6, 8]
>>> list(math.sqrt(x ** 2) for x in range(4))     # Плоский часто лучше
[0.0, 1.0, 2.0, 3.0]
>>> list(abs(x) for x in (-1, 0, 1))
[1, 0, 1]
```