```
def mapCall():
    return list(map((lambda x: x + 10), repslist))    # list() только в Python
3.X

def genExpr():
    return list(x + 10 for x in repslist)           # list() в Python 2.X +
3.X

def genFunc():
    def gen():
        for x in repslist:
            yield x + 10
    return list(gen())                              # list() в Python 2.X + 3.X
...
```

Теперь необходимость вызывать определяемую пользователем функцию для вызова `map` делает его медленнее операторов цикла `for` вопреки тому факту, что версия с операторами цикла крупнее в плане кода. Другими словами, удаление вызовов функций может сделать остальные версии более быстрыми (подробнее об этом в предстоящей врезке "На заметку!"). Вот результаты в Python 3.7:

```
c:\code> c:\python37\python timeseqs2.py
3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)]
forLoop  : 1.78581 => [10...10009]
listComp : 0.90618 => [10...10009]
mapCall  : 1.95757 => [10...10009]
genExpr  : 1.63313 => [10...10009]
genFunc  : 1.63656 => [10...10009]
```

Результаты в CPython также были согласованными. Результаты в Python 3.0 для предыдущего издания книги, полученные на менее производительном компьютере, снова оказались относительно похожими, хотя почти вдвое медленнее из-за отличий компьютеров, где проводилось тестирование (результаты в Python 2.5 снова были в четыре-пять раз медленнее по сравнению с текущими результатами).

Из-за того, что интерпретатор внутренне производит настолько много оптимизаций, анализ производительности кода Python такого вида является крайне сложным делом. Однако без цифр практически невозможно предугадать, какой метод будет выполняться наилучшим образом; все, что вы можете сделать — измерить время выполнения собственного кода на своем компьютере с имеющейся версией Python.

В нашем случае мы можем сказать наверняка лишь то, что в данной версии Python использование определяемой пользователем функции в вызовах `map`, похоже, существенно снижает производительность (хотя операция `+` также может быть медленнее тривиальной функции `abs`), и то, что списковые включения выполняются быстрее (хотя медленнее `map` в других ситуациях). Списковые включения согласованно выглядят в два раза более быстрыми, чем циклы `for`, но даже здесь требуется уточнение — на относительную скорость спискового включения могут повлиять применение дополнительного синтаксиса (например, фильтров `if`), изменения в Python и режимы использования, которые мы не хронометрировали.

Тем не менее, как упоминалось ранее, при написании кода Python производительность не должна быть вашей главной задачей; первое, что вы обязаны делать для оптимизации кода Python — не оптимизировать его! Пишите код изначально с акцентом на *читабельности* и *простоте*, а затем позже оптимизируйте, если и только если в этом есть необходимость. Вполне возможно, что любая из пяти альтернатив окажется достаточно быстрой для наборов данных, которые вашей программе нужно обрабатывать; в таком случае главной целью должна быть понятность программы.