Как отмечалось в главе 14, в Python 2.X объекты итераторов определяют метод по имени `next`, а не `__next__`. Это касается используемых здесь объектов генераторов. В Python 3.X данный метод переименован в `__next__`. Встроенная функция `next` предоставляется как удобный и переносимый инструмент: `next(I)` — то же самое, что `I.__next__()` в Python 3.X и `I.next()` в Python 2.6/2.7. До версии Python 2.6 в программах вместо ручной итерации просто вызывался `I.next()`.

Генераторные функции в действии

Для иллюстрации основ генераторов давайте перейдем к написанию кода. В следующем коде определяется генераторная функция, которую можно применять для генерации квадратов серии чисел с течением времени:

```
>>> def gensquares(N):
        for i in range(N):
            yield i ** 2    # Позже возобновить здесь выполнение
```

Функция `gensquares` выдает значение и потому возвращает управление вызывающему коду на каждой итерации цикла; при возобновлении ее выполнения восстанавливается предыдущее состояние, включая последние значения переменных `i` и `N`, а управление снова подхватывается непосредственно после оператора `yield`. Скажем, когда `gensquares` используется в теле цикла `for`, первая итерация начинает функцию и получает первый результат; затем на каждой итерации цикла управление возвращается функции после оператора `yield`:

```
>>> for i in gensquares(5):    # Возобновление выполнения функции
        print(i, end=' : ')     # Вывод последнего выданного значения
0 : 1 : 4 : 9 : 16 :
>>>
```

Чтобы закончить генерацию значений, функции либо применяют оператор `return` без значения, либо позволяют потоку управления дойти до конца тела функции[1].

Большинству людей такой процесс на первый взгляд кажется не совсем явным (а то и вообще магическим). Но на самом деле он вполне материален. Если вы действительно хотите увидеть, что происходит внутри `for`, тогда вызовите генераторную функцию напрямую:

```
>>> x = gensquares(4)
>>> x
<generator object gensquares at 0x000000000292CA68>
```

Вы получите обратно *объект генератора*, который поддерживает протокол итерации, встречавшийся в главе 14 — генераторная функция была скомпилирована так, чтобы возвращать его автоматически. В свою очередь возвращенный объект генератора имеет метод `__next__`, который запускает функцию или возобновляет ее выполнение с места, откуда она последний раз выдала значение, и инициирует исключение `StopIteration`, когда достигнут конец серии значений и происходит возврат управления из функции.

---

[1] Формально Python трактует наличие значения в операторе `return` внутри генераторных функций как синтаксическую ошибку в Python 2.X и во всех версиях Python 3.X, предшествующих Python 3.3. Начиная с версии Python 3.3, значение в операторе `return` разрешено и присоединяется к объекту `StopIteration`, но оно игнорируется в контекстах автоматических итераций, а его использование делает код несовместимым со всеми предыдущими выпусками.