На самом деле os.walk в Python реализована как рекурсивная функция в стандартном библиотечном файле os.py, находящемся в C:\Python37\Lib на компьютере Windows. Поскольку для возвращения результатов в ней применяется yield (и yield from вместо for, начиная с версии Python 3.3), она является нормальной генераторной функцией и потому итерируемым объектом:

```
>>> G = os.walk(r'C:\code\pkg')
>>> iter(G) is G    # Однопроходный итератор: вызов iter(G) необязателен
True
>>> I = iter(G)
>>> next(I)
('C:\\code\\pkg', ['__pycache__'], ['eggs.py', 'eggs.pyc', 'main.py', ...и
так далее...])
>>> next(I)
('C:\\code\\pkg\\__pycache__', [], ['eggs.cpython-37.pyc', ...и так далее...])
>>> next(I)
StopIteration
```

За счет выдачи результатов по мере продвижения инструмент прохода по каталогам не требует от своих клиентов ждать, пока не будет пройдено все дерево. Дополнительные сведения о данном инструменте ищите в руководствах по Python и в книге *Programming Python* (http://www.oreilly.com/catalog/9780596158101). Кроме того, в главе 14 и других главах упоминается os.popen — связанный итерируемый объект, используемый для запуска команды оболочки и чтения ее вывода.

Генераторы и применение функций

В главе 18 отмечалось, что аргументы со звездочкой способны распаковывать *итерируемый объект* в индивидуальные аргументы. Ознакомившись с генераторами, мы можем также посмотреть, что это означает в коде. Вот пример для Python 3.X и 2.X (хотя range в Python 2.X дает список):

```
>>> def f(a, b, c): print('%s, %s, and %s' % (a, b, c))

>>> f(0, 1, 2)                    # Нормальные позиционные аргументы
0, 1, and 2
>>> f(*range(3))                  # Распаковка значений range:
                                  #   итерируемый объект в Python 3.X
0, 1, and 2
>>> f(*(i for i in range(3)))     # Распаковка значений генераторного выражения
0, 1, and 2
```

Прием применим также к словарям и представлениям (хотя dict.values — список в Python 2.X, а при передаче значений по ключам порядок произволен):

```
>>> D = dict(a='Bob', b='dev', c=40.5); D
{'b': 'dev', 'c': 40.5, 'a': 'Bob'}
>>> f(a='Bob', b='dev', c=40.5)    # Нормальные ключевые аргументы
Bob, dev, and 40.5
>>> f(**D)                        # Распаковка словаря: ключ=значение
Bob, dev, and 40.5
>>> f(*D)                         # Распаковка итератора ключей
b, c, and a
>>> f(*D.values())                # Распаковка итератора представления:
                                  #   итерируемый объект в Python 3.X
dev, 40.5, and Bob
```