Yield From

Итак, у нас есть функция _connect. Она принимает sock, host, port, к которым нужно подключиться.

```
class Fetcher:
    def fetch(self):
        self.sock = socket.socket()
        self.sock.setblocking(False)
        for f in connect(self.sock, 'xkcd.com', 80):
            yield f
```

Задача — отрефакторить код так, чтобы могли использовать функцию connect в любой момент.

Мы находимся в нашей функции fetch. Мы создали сокет и итерируемся по всем Future, которые за-yield'ит функция connect().

Например, connect() подключается к сокету и начинает в него писать, то есть connect делает один yield для соединения и один yield для записи. Мы выступаем в роли какого-то посредника — connect() отдает нам сколько-то Future, а мы отдаем их дальше.

Это выглядит не очень удобно — любой генератор, у которого есть хотя бы два yield в теле, обязывает нас использовать цикл как в примере. А если в генераторе тоже используется цикл, то тем более.

Поэтому в Python ввели синтаксический сахар *yield from.*

Пример:

```
class Fetcher:
    def fetch(self):
        self.sock = socket.socket()
        self.sock.setblocking(False)
        yield from connect(self.sock, 'xkcd.com', 80)
```

(пример можно найти в examples/generators/yield_from):

Первый и второй примеры работают почти идентично. Единственное — но важное — отличие видно при завершении работы генератора. Если мы будем просто идти циклом по генератору, однажды генератор закончится и кинет ошибку StopIteration. Просто yield выкинет эту ошибку дальше, а yield_from перехватит и вернет последнее возвращенное значение *(это значение хранится в ошибке).* Т.е. когда в конце генератора мы вызываем return 1, выбрасывает исключение StopIteration со *StopIteration.value = 1.*

Поэтому чтобы примеры работали идентично, первый надо переписать:

```
class Fetcher:
    def fetch(self):
        self.sock = socket.socket()
        self.sock.setblocking(False)
        try:
            for f in connect(self.sock, 'xkcd.com', 80):
                yield f
        except StopIteration as e:
            return e.value
```