PEP 380: Синтаксис делегации субгенератору

**Аннотация:**  
Данный синтаксис предназначен для того, чтобы генератор мог передать часть своих функций другому генератору. Другими словами, часть кода, содержащую yield можно вынести и поместить в отдельную функцию. Кроме того, субгенератор может возвращать значение с помощью return, и это значение будет доступно делегирующему генератору. Благодаря этому синтаксису можно добиться оптимизации в случае, если один генератор должен выдавать значения другого генератора.  
  
  
**Мотивация:**  
Не смотря на то, что генератор является формой сопрограммы, он ограничен тем, что может выдавать свои результаты только тому, кто его непосредственно вызывает.Другими словами, часть функции, содержащая yield не может быть просто вынесена в отдельную функцию, как любой другой код. Иначе вызывающая функция сама становится генератором и нам придется явным образом проводить по ней итерацию и выдавать те значения, которые она производит.  
Конечно, если выдача значений — единственная задача субгенератора, то его можно организовать в простой цикл:  
for v in g:  
    yield v  
Однако как только нам потребуется взаимодействие вызывающей функции и генератора, то есть мы начнём пользоваться send(), throw() и close(), ситуация станет не такой простой. Как мы можно понять, код станет гораздо сложнее, так как придётся обрабатывать все возможные вызовы и исключения.  
Новый синтаксис позволяет легко решить данную проблему. В простейшем случае он аналогичен циклу, приведенному выше, но кроме этого он так же обрабатывает любое поведение генератора и позволяет провести рефакторинг кода без особых мучений.  
Предложение:  
В версии 3.3 доступен такой синтаксис:  
yield from <expr>  
где <expr> - выражение, вычисление которого даёт нам итерируемый объект, из которого и вычленяется итератор. Итератор, в свою очередь проходится до истощения и значения, выдаваемые им передаются напрямую вызывающей функции и принимаются от неё же, то есть "мимо" делегирующего генератора.  
Более того, как уже упоминалось, если итератор сам является генератором, то он может использовать return для передачи значения в делегирующий генератор (в таком случае выражение yield from принимает это значение)  
Если говорить терминами протокола генераторов, то всё это можно описать так:

* Любое значение, выдаваемое итератором, передаётся напрямую вызывающей функции
* Любые значения, посылаемые делегирующему генератору с использованием send() так же передаются напрямую итератору. Если посылается значение None, то вызывается метод \_\_next\_\_() итератора, иначе вызывается его метод send(). Если вызов приводит к возбуждению исключения StopIteration, то выполняются действия делегирующего оператора, идущие после yield from. Любые другие исключения распространяются делегирующему генератору.
* Исключения, кроме GeneratorExit, вброшенные в делегирующий генератор, передаются методом throw() итератору. Если это приводит к вызову исключения, то они обрабатываются как в предыдущем пункте.
* Если в делегирующий генератор вбрасывается GeneratorExit  исключение или вызывается его метод close(), в таком случае вызывается аналогичный метод итератора, если он у него есть.
* Значение выражения  yield from — первый аргумент исключения StopIneration, возбуждаемого при завершении итератора. (но я этого не обнаружил, смотрите листинг)
* Выражение return expr в генераторе возбуждает исключение StopIteration(exrp) при выходе из генератора (аналогично)

Дополнительно важные детали:

* Для удобства, как уже упоминалось, исключению StopIteration был добавлен атрибут value, в котором хранится его первый аргумент или None.
* Исключение  StopIteration **не**перехватывается в делегирующем итераторе!!
* Исключение  StopIteration  может быть представлено таким образом:
* Замечания по поводу рефакторинга кода:
  + Блок кода, который перехватывает исключение GeneratorExit без последующего его возбуждения не может быть корректно преобразован с сохранением функционала
  + Поведение составного итератора может не совпадать с поведением единого итератора, если возбуждается исключение  StopIteration  в делегирующем генераторе.
* Обсуждение реализаций Python которые не отслеживают ссылки привело к решению, что завершение делегирующего генератора приводит к завершению субгенератора. По этой причине не стоит создавать общие субитераторы, чтобы они не закрывались преждевременно.