Introduction

PEP 380 –Syntax for Delegating to a Subgenerator

The purpose of Generators in Python

Return returns the entire output at once. Yield, which is typically used by generators, yields only one iteration at a time

Code Example of yield

“”‘python def get_primes(number): while True: if is_prime(number): number $=$ yield number number $+=$ 1’“”

Weakness with Yield and Generators

A drawback to yield is that when yield is used in a function, it can only yield back to one caller

Proposal

“”‘python yield from expr’“”

Proposal (cont.)

“”‘python RESULT = yield from EXPR’“”

Comparisons

“”’python *i* = *iter(EXPR) try:* y = next(*i) except StopIteration as* e: *r* = e.value else: while 1: try: *s* = *yield* y except GeneratorExit as *e: try:* m = *i.close except AttributeError: pass else:* m() raise *e except BaseException as* e: *x* = *sys.exc*info() try: *m* = i.throw except AttributeError: raise *e else: try:* y = _m(**x) except StopIteration as* e: *r* = e.value break else: try: if *s is None:* y = next(*i) else:* y = *i.send(*s) except StopIteration as *e:* r = *e.value break RESULT* = r “”’

Syntax

With the new syntax, we can now move around the code with yield in it to a greater degree, making it easier for us to reuse it

Refactoring

Main purpose to move easily between functions and share data

Optimization

Delegating to subgenerators also helps to optimize in recursive calls

Compartmentalization

New syntax allows code to be split up, similar to threads

Similarities to Class

Small-Step Semantics

Counter-points

The proposal, PEP 380, is accepted but disagreed with due to its unusual way of using yield to get outputs

Rejected Automation

Use of automated next() calls not within scope of project

Rejected alternate return from sub-generator

Goes against idea of suspendable functions being like other functions

16

17

18

19

20