Title Slide

CSCI 3155 Presentation - Python

- Michael Min
- Andrew Orr
- Devon Connor

Introduction

- Our proposal was PEP 380 –Syntax for Delegating to a Subgenerator
- Generators are made to act like iterators.
- In Python, iterators are objects that use the __ **next** __ method and return a single value per iteration, one little piece at a time
- PEP 380 simply suggests making generators in Python more usable by giving another use to the following keyword:

```
yield
```

What is a Iterator?

```
class firstn(object):
    def __init__(self, n):
        self.n = n
        self.num, self.nums = 0, []
    def __iter__(self):
        return self
    # Python 3 compatibility
    def __next__ (self):
        return self.next()
    def next(self):
        if self.num < self.n:
            cur, self.num = self.num, self.num+1
            return cur
        else:
            raise StopIteration()
```

What is a Generator?

- Generators, on the other hand, do the same thing but are simpler and more readable:

```
def firstn(n):
    num = 0
    while num < n:
        yield num
        num += 1

sum_of_first_n = sum(firstn(1000000))
```

The Purpose of Generators in Python

- `Return`
  - The keyword that returns the entire output at once. Used by iterators
- `Yield`
  - The keyword typically used by generators, which yields only one iteration at a time. Used by generators

Code Example of yield and Generators

```
def get_primes(number):
    while True:
        if is_prime(number):
            number = yield number
        number += 1

get_primes(2)
```

- This function is turned into a generator that will constantly return numbers in its endless loop, one at a time

Weakness and Proposal

- The drawback to this incarnation of yield is that when yield is used in a function, it can only yield back to one caller
- The entire premise of PEP 380 is the following grammar:

```
yield from expr
```

Weakness and Proposal (cont.)

- When used more formally, the syntax is:

```
RESULT = yield from EXPR
```

Process

- The yield runs until EXPR is depleted of iterations, as usual
- The main change with PEP 380 is that it allows for yield to be used out of the function

Comparisons

```
_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()
```

Rejected alternate return from sub-generator

- There's been concerns that the implementation for "yield from" shouldn't use "return"
- This was rejected because that would mean "yield from" would return values fundamentally differently

Further Description of Proposal

- Other than the change with yield being added, no new keywords or symbols are actually added

Further Description of Proposal (cont.)

At one point,

```
yield *
```

was used instead of:

```
yield from
```

but it was ruled that it looked too similar to the yield normally used in functions, like in:

```
def count(number):
    for x in range(0,3):
        number = yield number
        number += 1
```

Preview of Pros

This new implementation helps to add:

- Functionality to generators
- Readability
- Reusability
- Optimization

Refactoring

- Since we can yield values to different locations now, it's easier to connect generators or functions this way

Ease of Use

- It's easy to redirect the result from a generator now:

```
generate1 = yield from add_10(5)

generate2 = yield from add_10(5)

generate3 = yield from add_10(5)
```

Optimization

- Delegating to subgenerators also helps to optimize in recursive calls

Counter-points

- The proposal, PEP 380, is accepted but disagreed with due to its unusual way of using yield to get outputs
- An argument that arises is that “yield from” and the “yield” used in functions get values differently and should not use the same keyword
- Some rejected keywords that were suggested to replace “yield from” were:

```
call    delegate    gcall
```

Similarities to Class

- Small-Step Semantics

Resources

- http://www.cosc.canterbury.ac.nz/greg.ewing/python/yield-from/
- https://www.python.org/dev/peps/pep-0380/
- https://wiki.python.org/moin/Generators