

Python

Generator and Generator Expressions

Stéphane Vialette

LIGM, Université Paris-Est Marne-la-Vallée

October 13, 2009

- 1 Introduction
- 2 Generators
- 3 Refactoring example
- 4 Enhanced generator features

Outline

- 1 Introduction
- 2 Generators
- 3 Refactoring example
- 4 Enhanced generator features

Introducing generator expressions

Description

- Generator expressions extend naturally from list comprehensions.
- Instead of building a list with values, they return a generator that yields after processing each item.
- Generator expressions are much more memory efficient by performing lazy evaluation.

List comprehension

```
[expr for item_var in iterable if cond_expr]
```

Generator expression

```
(expr for item_var in iterable if cond_expr)
```

Outline

- 1 Introduction
- 2 Generators**
- 3 Refactoring example
- 4 Enhanced generator features

Introducing generators

Example

```
In [1]: def gen():  
...:     print "gen: 1"  
...:     yield 1  
...:     print "gen: 2"  
...:     yield 2  
...:     return  
  
In [2]: my_gen = gen()  
In [3]: my_gen  
Out[3]: <generator object gen at 0x86eda54>  
In [4]: my_gen.next()  
gen: 1  
Out[4]: 1  
In [5]: my_gen.next()  
gen: 2  
Out[5]: 2  
In [6]: my_gen.next()
```

StopIteration Traceback (most recent call last)

Introducing generators

Example

```
In [1]: def gen():  
...:     yield 1  
...:     yield 2  
...:     return  
...:  
In [2]: my_gen = gen()  
In [3]: for val in my_gen:  
...:     print val  
...:  
1  
2  
In [4]: for val in gen():  
...:     print val  
...:  
1  
2  
In [5]:
```

Fibonacci

Example

```

#  $f_n = f_{n-1} + f_{n-2}$ 
def fib():
    fn2 = 1 # " $f_{n-2}$ "
    fn1 = 1 # " $f_{n-1}$ "
    while True:
        (fn1, fn2, oldfn2) = (fn1+fn2, fn1, fn2)
        yield oldfn2

genexpr = (val for val in fib())
for val in genexpr: # print 1 1 2 3 5 8 13
    if val > 20:
        break
    else:
        print val,

```


Representing infinite sequences

Example

a generator for 1, 2, ...

```
def infseq():
```

```
    i = 1
```

```
    while True:
```

```
        yield i
```

```
        i = i+1
```

This series converges to PI

```
def pi_series():
```

```
    sum, i, j = 0, 1.0, 1
```

```
    while(1):
```

```
        sum = sum + j/i
```

```
        yield 4*sum
```

```
        i, j = i + 2, j*-1
```

A convenient function that return the first n integers

Example

```
def pi_series():
    sum, i, j = 0, 1.0, 1
    while(1):
        sum = sum + j/i
        yield 4*sum
        i, j = i + 2, j*-1

def firstn(g, n):
    for i in range(n):
        yield g.next()

print list(firstn(pi_series(), 8))
# [4.0, 2.6666666666666667, 3.4666666666666668, 2.8952380952380956,
3.3396825396825403, 2.9760461760461765, 3.2837384837384844,
3.0170718170718178]
```

The Eratosthenes sieve

Example

```
def intsfrom(i):  
    while 1:  
        yield i  
        i = i + 1  
  
def exclude_multiples(n, ints):  
    for i in ints:  
        if (i % n):  
            yield i  
  
def sieve(ints):  
    while True:  
        prime = ints.next()  
        yield prime  
        ints = exclude_multiples(prime, ints)  
  
print list(firstn(exclude_multiples(2, intsfrom(1)), 5))  
# [1, 3, 5, 7, 9].
```

Convert a generator

Example

```
def listify(gen):  
    "Convert a generator into a function which returns a list"  
    def patched(*args, **kwargs):  
        return list(gen(*args, **kwargs))  
    return patched  
  
@listify  
def f(x):  
    for i in range(x):  
        yield "item" + str(i)  
  
print f(5) # print ['item0', 'item1', 'item2', 'item3', 'item4']
```

Outline

- 1 Introduction
- 2 Generators
- 3 Refactoring example**
- 4 Enhanced generator features

Dealing with large files: Refactoring example 1/5

Example

```
def longest_line_in_file(filename):  
    f = open(filename, 'r')  
    longest_line = 0  
    while True:  
        line_len = len(f.readline().strip())  
        if not line_len:  
            break  
        if line_len > longest_line:  
            longest_line = line_len  
    f.close()  
    return longest_line
```

Dealing with large files: Refactoring example 2/5

Example

```
def longest_line_in_file(filename):  
    f = open(filename, 'r')  
    all_lines = f.readlines()  
    f.close()  
    longest_line = 0  
    for line in all_lines:  
        line_len = len(line.strip())  
        if line_len > longest_line:  
            longest_line = line_len  
    return longest_line
```

Dealing with large files: Refactoring example 3/5

Example

```
def longest_line_in_file(filename):  
    f = open(filename, 'r')  
    all_lines = [line.strip() for line in f.readlines()]  
    f.close()  
    longest_line = 0  
    for line in all_lines:  
        line_len = len(line.strip())  
        if line_len > longest_line:  
            longest_line = line_len  
    return longest_line
```


Dealing with large files: Refactoring example 4/5

Example

```
def longest_line_in_file(filename):  
    f = open(filename, 'r')  
    all_len = [len(line.strip()) for line in f]  
    f.close()  
    return max(all_len)
```

Dealing with large files: Refactoring example 5/5

Example

```
def longest_line_in_file(filename):  
    f = open(filename, 'r')  
    longest_line = max(len(line.strip()) for line in f)  
    f.close()  
    return longest_line
```

Outline

- 1 Introduction
- 2 Generators
- 3 Refactoring example
- 4 Enhanced generator features**

A simple counter

Example

```
def counter(maximum):
```

```
    i = 0
```

```
    while i < maximum:
```

```
        yield i
```

```
        i += 1
```

```
for i in counter(10): # print 0 1 2 3 4 5 6 7 8 9
```

```
    print i,
```

In Python 2.5, yield is now an expression

Example

```
def counter(maximum):
```

```
    i = 0
```

```
    while i < maximum:
```

```
        val = (yield i)
```

```
        if val is not None:
```

```
            i = val
```

```
        else:
```

```
            i += 1
```

```
my_counter = counter(10)
```

```
print my_counter.next() # print 0
```

```
print my_counter.next() # print 1
```

```
print my_counter.send(5) # print 5
```

```
print my_counter.next() # print 6
```

```
print my_counter.send(1) # print 1
```

```
print my_counter.next() # print 2
```

```
print my_counter.send(11) # raise StopIteration
```