Python Generators

the 'yield' keyword and some other fun stuff

Scott Newson

```
// A function that returns nothing
>>> def f():
...   print('hello!')
...   return
...
```

```
// A function that returns nothing
>>> def f():
...   print('hello!')
...   return
...
>>> f()
hello!
```

```
// A function that returns nothing
>>> def f():
...   print('hello!')
...   return
...
>>> f()
hello!
>>>
// (actually, you don't need the 'return')
>>> def f():
...   print('hello!')
...
>>> f()
hello!
>>>
```

```
// A function that returns something
>>> def f():
...   print('hello!')
...   return 3
...
>>> x = f()
hello!
>>> print(x)
3
```

```
// A function that returns something
>>> def f():
...   print('before')
...   return 3
...   print('after')
...
>>> x = f()
before
>>> print(x)
3
// Notice that 'after' is never printed
```

```
// A function that yields something
>>> def g():
...   yield 3
...
>>> gen = g()
>>> x = gen.next()
>>> print(x)
3
```

```
// A function that yields two things!
>>> def g():
...   yield 3
...   yield 4
...
>>> gen = g()
>>> print(gen.next())
3
>>> print(gen.next())
4
>>>
```

```
// A function that yields two things!
>>> def g():
...   yield 3
...   yield 4
...
>>> gen = g()
>>> print(gen.next())
3
>>> print(gen.next())
4
>>> print(gen)
<generator object g at 0x10faa8a00>
```

```
// A function that yields two things!
>>> def g():
...   yield 3
...   yield 4
...
>>> gen = g()
>>> print(gen.next())
3
>>> print(gen.next())
4
>>> gen.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
>>>
```

```
// A function that yields something
>>> def g():
...   print('before')
...   yield 3
...   print('after')
...
>>> gen = g()
>>> x = gen.next()
before
>>> print(x)
3
>>> y = gen.next()
after
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
>>>
```

```
// A function that yields Fibonacci numbers
// (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55...)
>>> def g():
...   n = 0
...   while(True):
...     yield fib(n)
...     n = n + 1
...
>>> gen = g()
>>> print(gen.next())
0
>>> print(gen.next())
1
>>> print(gen.next())
1
>>>
// This can go on forever!
```

```
// A function that yields Fibonacci numbers
// (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55...)
>>> def g():
...   n = 0
...   while(True):
...     yield fib(n)
...     n = n + 1
...
>>> gen = g()
>>> print(gen.next())
0
>>> print(gen.next())
1
>>> print(gen.next())
1
>>> print(gen.next())
2
>>> print(gen.next())
3
>>> print(gen.next())
5
>>>
// This can go on forever!
```

```
// A function that yields prime numbers
// (2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31...)
>>> def g():
...   n = 2
...   while(True):
...     if(isPrime(n)):
...       yield n
...     n = n + 1
...
>>> gen = g()
>>> print(gen.next())
2
>>> print(gen.next())
3
>>> print(gen.next())
5
>>> print(gen.next())
7
>>> print(gen.next())
11
>>>
// This can go on forever!
```

```
// Context managers using yield
@contextmanager
def bold():
    print "<b>"
    yield
    print "</b>"

>>> with bold():
...     print "foo"
...
<b>
foo
</b>
```

```
// Stream processing
>>>
```

```
// Sending to a yield
>>> def g():
...   n = 42
...   while(True):
...     n = yield (n + 1)
...
>>> gen = g()
>>> gen.next()
43
>>> gen.send(3)
4
>>> gen.send(4)
5
```

```
// Sending to a yield
>>> def g():
...    n = 42
...    print('enter')
...    while(True):
...      print('before')
...      n = yield (n + 1)
...      print('after')
...
>>> gen = g()
>>>gen.next()
enter
before
43
>>>gen.send(3)
after
before
4
>>>
```

```
// Speaker notes
>>> print(author.name)
Scott Newson
>>> print(author.contact)
scott.g.newson@gmail.com
```