

# Infinite Sequences in Python

- Bob Miller
- 
- Iterators and Generators
- A Puzzle

# Part 1

- Iterators
- Iteration Protocol
- Generators
- Generators as Iterators
- Infinite Generators

# Iteration

## A simple loop

```
>>> vehicles = ['car', 'bike', 'bus']
>>> for ride in vehicles:
...     print ride
...
car
bike
bus
>>>
```

# Iteration Protocol

## How Python compiles it

```
iterator = vehicles.__iter__()  
while True:  
    try:  
        ride = iterator.next()  
    except StopIteration:  
        break  
    print ride
```

# Iteration Protocol

An *iterable* object

- `__iter__()` method returns an iterator.

An *iterator* object

- `next()` method returns next value
- raises `StopIteration` exception when done

Duck typing in action.

# An Iterator Class

```
class MyIterator:

    def __init__(self, sequence):
        self.sequence = sequence
        self.index = 0

    def next(self):
        try:
            r = self.sequence[self.index]
            self.index += 1
            return r
        except IndexError:
            raise StopIteration
```

# An Iterable

```
class MyIterable:  
  
    def __iter__(self):  
        return MyIterator(self)  
  
    # other methods...
```

# Generators

A *generator* is a function that *yields* a value and keeps going.

```
>>> def gen():  
...     yield 'car'  
...     yield 'bike'  
...     yield 'bus'  
...  
>>>
```

But how do you get all the values?

# Generators

```
>>> g = gen()
>>> g.next()
'car'
>>> g.next()
'bike'
>>> g.next()
'bus'
>>> g.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
>>>
```

# Generators as Iterators

```
>>> for ride in gen():  
...     print ride  
car  
bike  
bus  
>>>
```

# Infinite Sequences

A generator doesn't have to stop...

```
>>> def natural_numbers():
...     i = 1
...     while True:
...         yield i
...         i += 1
...
>>> from itertools import islice
>>> islice(natural_numbers(), 5)
<itertools.islice object at 0x8339acc>
>>> list(islice(natural_numbers(), 5))
[1, 2, 3, 4, 5]
```

# Part 2

- A Puzzle from NewScientist magazine

“Triangular Triples”

No. 1434

Richard England

Vol 193 No 2595

March 17-23, 2007

page 56

# Triangular Triples

A triangular number is an integer that fits the formula  $\frac{1}{2}n(n+1)$  such as 1, 3, 6, 10, 15.

45 is not only a triangular number ( $\frac{1}{2} * 9 * 10$ ) but also the product of three different factors ( $1 * 3 * 15$ ) each of which is itself a triangular number. But Harry, Tom and I don't regard that as a satisfactory example since one of those factors is 1; so we have been looking for triangular numbers that are the product of three different factors each of which is a triangular number other than 1.

We have each found a different 3-digit triangular number that provides a solution. Of the factors we have used, one appears only in Harry's solution and another only in Tom's solution. What are the three triangular number factors of my solution?

£15 will go to the sender of the first correct answer[...]

# Triangular Numbers

*“A triangular number is an integer that fits the formula  $\frac{1}{2} n (n + 1)$ ...”*

```
>>> def triangulars():  
...     for n in natural_numbers():  
...         yield n * (n + 1) / 2  
...  
>>> list(islice(triangulars(), 5))  
[1, 3, 6, 10, 15]
```

# Triples

*"... the product of three different numbers, each of which is a triangular number"*

```
>>> def triplit(seq):
...     '''triple iterator.'''
...     for i, iv in enumerate(seq):
...         for j, jv in enumerate(islice(seq, i):
...             for k, kv in enumerate(islice(seq, j)):
...                 yield kv, jv, iv
...
>>> list(triplit(['car', 'bike', 'bus', 'canoe']))
[('car', 'bike', 'bus'),
 ('car', 'bike', 'canoe'),
 ('car', 'bus', 'canoe'),
 ('bike', 'bus', 'canoe')]
```

# Triples

But this doesn't work.

```
>>> list(islice(triplit(triangulars()), 3))  
[(28, 21, 10), (66, 55, 36), (91, 78, 36)]
```

Oops.

# Triangular Numbers again

```
class Triangulars:

    def __init__(self):
        self.n = 1
        self.seq = []
        self.set = set()
        self._expand()

    def __iter__(self):
        i = 0
        while True:
            while len(self.seq) <= i:
                self._expand()
            yield self.seq[i]
            i += 1

# continued...
```

# Triangular Numbers again

```
# class Triangular continued...

def _expand(self):
    n = self.n
    t = n * (n + 1) / 2
    self.n += 1
    self.seq.append(t)
    self.set.add(t)

def __contains__(self, n):
    while self.seq[-1] < n:
        self._expand()
    return n in self.set

triangulars = Triangulars() # singleton instance
```

# Triangular Numbers again

Now it works.

```
>>> list(islice(triplit(Triangulars()), 3))  
[(1, 3, 6), (1, 3, 10), (1, 6, 10)]
```

# Triangular Triples

*“... the product of three different numbers, each of which is a triangular number”*

```
>>> class Triples:
...     def __iter__(self):
...         for a, b, c in triplit(triangulars):
...             if (1 not in (a, b, c) and
...                 a * b * c in triangulars):
...                 yield a, b, c
...
>>> list(islice(Triples(), 3))
[(3, 6, 21), (3, 10, 21), (6, 15, 36)]
```

# Which Triples?

*"... 3-digit triangular number..."*

*"...one appears only in Harry's and another only in Tom's..."*

Presumably Dick's solution has no unique factors.

# Finding suitable triples

```
from operator import mul

def is_suitable(trio0, trio1, trio2):
    too_big = [reduce(mul, t) > 999
                for t in (trio0, trio1, trio2)]
    if all(too_big):
        raise SystemExit      # Yikes!
    if any(too_big):
        return False
    return (freqs(trio0 + trio1 + trio2)
            == [1, 1, 2, 2, 3])

def freqs(seq):
    d = collections.defaultdict(int)
    for i in seq:
        d[i] += 1
    return sorted(d.values())
```

# Solve the Puzzle

```
>>> def solve():
...     for triple_trio in triplit(Triples()):
...         if is_suitable(*triple_trio):
...             for i, j, k in triple_trio:
...                 print ('%d = %d * %d * %d'
...                         % (i * j * k, i, j, k))
...
>>> solve()
378 = 3 * 6 * 21
630 = 3 * 10 * 21
990 = 3 * 6 * 55
>>>
```

# Solution

Hey, there's a unique solution!  
(Puzzle writers are lucky.)

$$378 = 3 * 6 * 21$$

$$630 = 3 * 10 * 21$$

$$990 = 3 * 6 * 55$$

Tom and Harry's unique factors: 10, 55  
Dick's number: 378.

**End**

Thank you.