Generators
Announcements
Generators
Generators and Generator Functions

>>> def plus_minus(x):
...     yield x
...     yield -x

>>> t = plus_minus(3)
>>> next(t)
3
>>> next(t)
-3
>>> t
<generator object plus_minus ...>

A generator function is a function that yields values instead of returning them.

A normal function returns once; a generator function can yield multiple times.

A generator is an iterator created automatically by calling a generator function.

When a generator function is called, it returns a generator that iterates over its yields.

(Demo)
Generators & Iterators
Generator Functions can Yield from Iterables

A `yield from` statement yields all values from an iterator or iterable (Python 3.3)

```python
>>> list(a_then_b([3, 4], [5, 6]))
[3, 4, 5, 6]

def a_then_b(a, b):
    for x in a:
        yield x
    for x in b:
        yield x

def countdown(k):
    if k > 0:
        yield k
        yield from countdown(k-1)
```

(Demo)
Example: Partitions
Yielding Partitions

A partition of a positive integer \( n \), using parts up to size \( m \), is a way in which \( n \) can be expressed as the sum of positive integer parts up to \( m \) in increasing order.

\[
\text{partitions}(6, 4)
\]

\[
\begin{align*}
2 + 4 & = 6 \\
1 + 1 + 4 & = 6 \\
3 + 3 & = 6 \\
1 + 2 + 3 & = 6 \\
1 + 1 + 1 + 3 & = 6 \\
2 + 2 + 2 & = 6 \\
1 + 1 + 2 + 2 & = 6 \\
1 + 1 + 1 + 1 + 2 & = 6 \\
1 + 1 + 1 + 1 + 1 + 1 & = 6
\end{align*}
\]

\[
\text{def count_partitions}(n, m):
\quad \text{if } n == 0:
\quad \quad \text{return } 1
\quad \text{elif } n < 0:
\quad \quad \text{return } 0
\quad \text{elif } m == 0:
\quad \quad \text{return } 0
\quad \text{else:}
\quad \quad \text{with}_m = \text{count_partitions}(n-m, m)
\quad \quad \text{without}_m = \text{count_partitions}(n, m-1)
\quad \quad \text{return } \text{with}_m + \text{without}_m
\]

(Demo)