Generators and Generator Functions

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)

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

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

>>> list(countdown(5))
[5, 4, 3, 2, 1]
```

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.

```python
def count_partitions(n, m):
    if n == 0:
        return 1
    elif n < 0:
        return 0
    elif m == 0:
        return 0
    else:
        with_m = count_partitions(n-m, m)
        without_m = count_partitions(n, m-1)
        return with_m + without_m

partitions(6, 4)

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 + 2 = 6
1 + 1 + 1 + 1 + 1 = 6

def count_partitions(n, m):
    if n == 0:
        return 1
    elif n < 0:
        return 0
    elif m == 0:
        return 0
    else:
        with_m = count_partitions(n-m, m)
        without_m = count_partitions(n, m-1)
        return with_m + without_m

(partitions(6, 4))
```