1 Tree Recursion

1.1 Consider a special version of the count_stairways problem, where instead of taking 1 or 2 steps, we are able to take up to and including $k$ steps at a time.

Write a function `count_k` that figures out the number of paths for this scenario.

```python
def count_k(n, k):
    """
    >>> count_k(3, 3) # 3, 2 + 1, 1 + 2, 1 + 1 + 1
    4
    >>> count_k(4, 4)
    8
    >>> count_k(10, 3)
    274
    >>> count_k(300, 1) # Only one step at a time
    1
    """
```
2 Mutable Linked Lists and Trees

2.1 Write a recursive function `flip_two` that takes as input a linked list `lnk` and mutates `lnk` so that every pair is flipped.

```python
def flip_two(lnk):
    """
    >>> one_lnk = Link(1)
    >>> flip_two(one_lnk)
    >>> one_lnk
    Link(1)
    >>> lnk = Link(1, Link(2, Link(3, Link(4, Link(5)))))
    >>> flip_two(lnk)
    >>> lnk
    Link(2, Link(1, Link(4, Link(3, Link(5)))))
    """
```

2.2 Write a function `flatten` that given a Tree `t`, will return a linked list of the elements of `t`, ordered by level. Entries on the same level should be ordered from left to right. For example, the following tree will return the linked list `<1 2 3 4 5 6 7>`.

```
def flatten(t):
```
3 Streams

3.1 (Fall 2014) Implement `cycle` which returns a Stream repeating the digits 1, 3, 0, 2, and 4, forever. Hint: \((3+2) \mod 5 == 0\).

```python
def cycle(start=1):
    # Return a stream repeating 1, 3, 0, 2, 4 forever.

    >>> stream_to_list(cycle(), n=12)
    [1, 3, 0, 2, 4, 1, 3, 0, 2, 4, 1, 3]
```

3.2 Write a function `merge` that takes 2 sorted Streams `s1` and `s2`, and returns a new sorted Stream which contains all the elements from `s1` and `s2`. 
4 Generators

4.1 Write a generator function that yields functions that are repeated applications of a one-argument function \( f \). The first function yielded should apply \( f \) 0 times (the identity function), the second function yielded should apply \( f \) once, etc.

```python
def repeated(f):
    """
    >>> [g(1) for _, g in
       ... zip(range(5), repeated(double))]
    [1, 2, 4, 8, 16]
    """

    g = ______________________________________________________

    while True:
        ______________________________________________________
        ____________________________________________________

4.2 Ben Bitdiddle proposes the following alternate solution. Does it work?

```python
def ben_repeated(f):
    g = lambda x: x
    while True:
        yield g
        g = lambda x: f(g(x))
```

4.3 Implement accumulate, which takes in an iterable and a function \( f \) and yields each accumulated value from applying \( f \) to the running total and the next element.

```python
from operator import add, mul

def accumulate(iterable, f=add):
    """Return running totals
    >>> list(accumulate([1,2,3,4,5]))
    [1, 3, 6, 10, 15]
    >>> list(accumulate([1,2,3,4,5], mul))
    [1, 2, 6, 24, 120]
    """
    it = iter(iterable)
```