1 Recursion

Definition: **recursion** is an algorithmic technique where a function, in order to accomplish a task, calls itself with some part of the task.

Structure: every recursive solution involves two major parts:

(1) **base case**, where the problem is simple enough to be solved directly

(2) **recursive case**, which has three components

 (a) **divide** problem into one or more simpler or smaller parts of the problem,

 (b) **call** the function (recursively) on at least one part, and

 (c) **combine** the solutions of the parts into a solution for the problem

Sometimes there is more than one base case. Sometimes there is more than one recursive case.

2 Downup

The function `downup` takes a string and prints out a pattern. For example, `downup('howdy')` prints this:

```
howdy
howd
how
ho
h
ho
how
howd
howdy
```

The pattern can be described in a self-referential (or recursive) way. The downup pattern for "howdy" is the word "howdy," followed by the downup pattern for "howd," followed by "howdy" again.

Here is a recursive approach that *prints* the parts of the pattern as it goes.

```
def downup(s):
    if len(s) <= 1:
        print s
    else:
        print s
        downup(s[:-1])
        print s
```

Here is a visualization of what happens when `downup` is called on `'hey!'`. Indentation is used to show the levels of recursion.

```
downup('hey!')
    print 'hey!'
    downup('hey')
        print 'hey'
        downup('he')
            print 'he'
            downup('h')
                print 'h'
            print 'he'
        print 'hey'
    print 'hey!'
```

Here is an alternative approach, also recursive, that *returns* a string that contains the entire downup pattern. The reason for showing you this second approach is that the structure of this code very closely matches the structure of recursive solution outlined at the beginning of this handout.

```python
def downup(s):
    if len(s) <= 1:        # (1) base case
        return s
    else:                  # (2) recursive case
        smaller = s[:-1]                          # (a) divide
        result = downup(smaller)                  # (b) call
        sandwich = s + '\n' + result + '\n' + s   # (c) combine
        return sandwich
```

3 Factorial

Here is a recursive approach for calculating the factorial of a number. The factorial of 4 is $4! = 4 \times 3 \times 2 \times 1 = 24$. In general, the factorial of $n! = n \times (n-1) \times (n-2) \times \cdots \times 2 \times 1$.

The recursive "insight" is to see that $n! = n \times (n-1)!$ except when $n = 1$ in which case $1! = 1$.

```python
def fact(n):
    '''(int) -> int
    Returns n! where n is expected to be a
    positive integer.
    '''
    if n == 1:             # (1) base case
        return 1
    else:                  # (2) recursive case
        result = fact(n-1) #     (a) divide and (b) call
        return n * result  #     (c) combine
```

Definition of recursion adapted from NIST, `http://xlinux.nist.gov/dads//HTML/recursion.html`. The downup example adapted from Brian Harvey.