Exercise 3
EECS-111 Fall 2015

Out: Friday, October 23
Due: Friday, October 30, noon

Introduction
In this exercise, you will practice writing recursive procedures.

Problem 1
Write a recursive procedure, **multiply-list**, that takes a list of numbers as input and returns their product. For example, (multiply-list (list 1 2 3 4)) should return 24.

Problem 2
Write a procedure, **multiply-list-iter**, that uses iterative recursion.

Problem 3
Write a recursive procedure, **count**, that takes as inputs, a predicate of one argument, and a list, and returns the number of elements of the list for which the predicate returns true. Thus (count odd? (list 1 2 3 4 5)) should return 3, since there are 3 odd numbers in the list. **Do not use filter** for this problem; you must write it as a recursion.

Problem 4
Now write write count as an **iterative recursion**; call it **count-iter**. If you wrote the last problem as an iterative recursion, that's fine. Just use that answer for this problem and write a new, non-iterative, recursion for the previous problem.

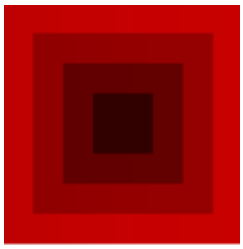
Problem 5
Write **iterated-overlay** as a recursive procedure. Do not use apply, foldl, foldr, or build-list. Test your procedure to make sure it works properly. Do not make it an iterative recursion.

Hint 1: you will need to return a blank image if the count is zero; you can do this by returning empty-image.

Hint 2: remember that (overlay *a b*), which puts *a* on top of *b*, gives you a different picture than (overlay *b a*), which puts *b* on top of *a*. You need to make sure that in the result of (iterated-overlay proc 5), that (proc 0) is on top, then (proc 1) is under it, (proc 2) under that, etc. You can test the ordering of you pictures with a test like this:

```
(iterated-overlay (λ (n)
                    (square (* n 10)
                            "solid"
                            (color (* n 50)
                                   0
                                   0)))
                  5)
```

which should produce an output like this:



Problem 6

A **generalized list**, aka a **recursive list**, is a list that can have other lists inside it, and other lists inside them, etc. So a recursive list of numbers can have numbers in the list or numbers in the sublists, or numbers in the sub-sublists, etc. Formally, we'll say a recursive list of numbers is either:

- A number by itself, or
- A list of recursive lists of numbers

That means all the following are recursive lists of numbers:

- (list 1 2)
- (list 1
    (list 2 3)
    4)
- (list 1
    (list 2
      (list 3 3)
      (list 2

(list 4 5)))
7)

- (list 1
(list 2 1)
1
1)
- 1

Write a recursive procedure (**sum-recursive-list** *x*) that takes a recursive list of numbers and returns the sum of all the numbers in it. So:

- (sum-recursive-list 1) should just return 1
- (sum-recursive-list (list 1 2 3)) should return 6, and
- (sum-recursive-list (list 1 (list 2 (list 3) 4)) should return 10

Note that you can test whether a given value is a number by using the number? procedure. That is, (number? *x*) will return true if *x* is a number and false if it's something else. For this problem, if the argument isn't a number, it will always be a list.

Problem 7

The built-in procedure, max takes 1 or more numbers as input, and returns the largest of them. Write a procedure, **max-recursive-list**, that returns the largest number in a recursive list. Hint: this is effectively the same problem as the previous problem.

Problem 8

Write a procedure, **depth**, that takes a recursive list as input and returns the number of levels of nesting within it. So:

- (depth 1) should return 0 because it's not even a list, but
- (depth (list 1 2 3)) should return 1, since it has one level list nesting, and
- (depth (list 1 (list 2) 3)) should return 2 because it has a list within a list.