ACCUMULATE (AKA REDUCE, AKA FOLD)

Here is the definition of accumulate from SICP:

```
(define (accumulate op initial sequence)
  (if (null? sequence)
      initial
      (op (car sequence)
           (accumulate op initial (cdr sequence)))))
```

**Question 1.** The following expression will compute the sum of the list '(1 2 3). Show how this is accomplished by expanding the recursion using the Substitution Model.

(accumulate + 0 '(1 2 3))
Question 2. Accumulations don’t have to reduce a list to a single value. E.g., if the op function is cons, they can reproduce lists.

What will be returned by this expression?

\[(accumulate \text{ cons} \ () \ '(1\ 2\ 3\ 4))\]

Question 3. The op function must be a procedure of two parameters (e.g., + and cons). It’s helpful to recognize that the first parameter will be the single item being worked on at each step of the recursion, and the second parameter will be the result of the prior accumulation. This latter thing may be a single value (e.g., in an accumulation using +) or a list (e.g., in an accumulation using cons).

What is the result of evaluating the following expression? Use Substitution Model if you are not sure.

\[(accumulate (\lambda (v\ l) (\text{cons} (\text{add1}\ v)\ l))\ ()\ '(1\ 2\ 3\ 4))\]

More practice problems.

(EASY) Use an accumulation to count the number of elements in a list.

(EASY) Use an accumulation to sum up only the even numbers in a list.

(EASY) Use an accumulation to scale a list by 10.

(EASY) Use an accumulation to filter a list to keep even numbers only.

(INTEGERMEDIATE) Use an accumulation to compute the average of a list. Note: You’ll have to pass back a compound object that includes sum-so-far and count-of-elements-so-far (e.g., a list of two numbers).

(HARD) Use an accumulation to work through an ordered list of natural numbers, and return a list containing only one instance of each number. E.g. from \'(1\ 1\ 3\ 3\ 3\ 5)\) to \'(1\ 3\ 5).