Pairs
What do each of the following evaluate to?

(define u (cons 2 3)) (define w (cons 5 6)) (define x (cons u w))
(define y (cons w x)) (define z (cons 3 y))

1. u, w, x, y, z (write them out in Schemes notation)
   u: (2 . 3)
   w: (5 . 6)
   x: ((2 . 3) 5 . 6)
   y: ((5 . 6) (2 . 3) 5 . 6)
   z: (3 (5 . 6) (2 . 3) 5 . 6)

2. (car y)
   (5 . 6)

3. (car (car y))
   5

4. (cdr (car (cdr (cdr z))))
   3

5. (+ (cdr (car y)) (cdr (car (cdr z))))
   12

6. (cons z u)
   ((3 (5 . 6) (2 . 3) 5 . 6) 2 . 3)

7. (cons (car (cdr y)) (cons (car (car x)) (car (car (cdr z)))))
   ((2 . 3) 2 . 5)

Lists
1. Define a procedure length that takes in a list and returns the number of elements within the list.

   (define (length ls)
     (if (null? ls)
         0
         (+ 1 (length (cdr ls)))))

2. Define a procedure (insert-after item mark ls) which inserts item after mark in ls.

   (define (insert-after item mark ls)
     (cond ((null? ls) '())
           ((equal? (car ls) mark)
            (cons (car ls) (cons item (cdr ls))))
           (else (cons (car ls) (insert-after item mark (cdr ls))))))

3. Define a procedure list? that takes in something and returns true if its a list, false otherwise. A list is a ”chain of pairs terminated by the end-of-list marker” (SICP)

   (define (list? ls)
     (or (null? ls)
         (and (pair? ls)
             (list? (cdr ls))))))

4. Suppose we have x bound to a mysterious element. All we know is this:
(list? x) ==> #t
(pair? x) ==> #f

What is x?

'() - the null list, empty list, end-of-list marker

5. Add in procedure calls to get the desired results. The blanks dont need to have anything:

(cons   'a         '(b c d e) )
==> (a b c d e)

(append    '(cs61a is)  (list 'cool) )
==> (cs61a is cool)

(cons       '(back to)       '(save the universe) )
==> ((back to) save the universe)

(cons   '(I keep the wolf)   (car   '((from the door))) )
==> ((I keep the wolf) from the door)

Deep Lists
1. Define a procedure (depth ls) that calculates how maximum levels of sublists there are in ls. For example,

(depth ' (1 2 3 4)) ==> 1  
(depth ' (1 2 (3 4) 5)) ==> 2  
(depth ' (1 2 (3 4 5 (6 7) 8) 9 (10 11) 12)) ==> 3

Remember that there's a procedure called max that takes in two numbers and returns the greater of the two.

(define (depth ls)
  (if (atom? ls)
      0
      (max (+ 1 (depth (car ls))) (depth (cdr ls)))))

2. Define a procedure (count-of item ls) that returns how many times a given item occurs in a given list; it could also be in a sublist. So,

(count-of 'a ' (a b c a a (b d a c (a e) a) b (a))) ==> 7

(define (count-of item ls)
  (cond ((null? ls) 0)
       ((pair? (car ls))
        (+ (count-of item (car ls))
           (count-of item (cdr ls))))
       ((equal? item (car ls)) (+ 1 (count-of item (cdr ls))))
       (else (count-of item (cdr ls)))))

3. Define a procedure (interleave ls1 ls2) that takes in two lists and returns one list with elements from both lists interleaved. So,

(interleave 'a b c d ' (1 2 3 4 5 6 7)) ==> (a 1 b 2 c 3 d 4 5 6 7)

(define (interleave ls1 ls2)
  (cond ((null? ls1) ls2)
       ((null? ls2) ls1)
       (else (cons (car ls1) (interleave ls2 (cdr ls1))))))