Worksheet #3
Michael Collins

======================================================================
Draw box and pointer diagrams for the following expressions:

(cons 1 2)
(list 1 2)
(cons 1 (cons 2 nil))
(cons 1 nil)
(cons 1 (cons 2 3))
(cons (cons 1 2) nil)
(list 1 2 3 4)
(list 1 (cons 2 3) (list 4 5))

======================================================================
In scheme, we will often want to access elements deep in a cons structure. Therefore, the following accessors have been designed to help us out:

(cadr x) = (car (cdr x))
(caddr x) = (car (cdr (cdr x)))
(cdaar x) = (cdr (car (car x)))
(cddr x) = (cdr (cdr x))
etc. etc.

say we define:

(define x (list 1 2 3 4 5 6 7 8 9 10))

How could you get the values 1, 2, 3, 4 using the c???r functions on x?

======================================================================
Here’s a procedure which appends two lists together:

(define (append a b)
  (if (null? a)
    b
    (cons (car a) (append (cdr a) b)))))

Trace what happens (using box and pointer diagrams), when we apply

(append (list 1 2) (list 3 4 5))
Write a procedure that sums the elements in a list. The procedure should generate a recursive process.

Write a procedure ‘square-ls’ that takes a list as input, and returns a new list with all values within the list squared. For example

(square-ls (list (1 2 3 4))) => (1 4 9 16)

Write a procedure that sums the elements in a list. The procedure should generate an iterative process.

Take the following definition for a procedure ‘foo’:

(define foo (lambda (x)
    (foo-h x nil)))

(define foo-h (lambda (x y)
    (if (null? x)
        y
        (foo-h (cdr x) (cons (car x) y)))))

Trace the process generated by (foo (list 1 2 3 4)). What does foo do to a list?

Can you think of a definition of foo which achieves the same thing, but creates a recursive process? (hint: it’s ok to use append)