The Little Schemer, Ch 1

Concepts:

- atom
- list
- S–expression
- car
- cdr
- cons
- null?
- atom?
- eq?
The Laws

The Law of car: The primitive *car* is defined only for non–empty lists.

The Law of cdr: The primitive *cdr* is defined only for non–empty lists. The *cdr* of a any non–empty list is always another list.

The Law of cons: The primitive *cons* always takes two arguments. The second argument must be a list. The result is a list.
The Laws, cont.

The Law of null?: The primitive null? is defined only for lists.

The Law of eq?: The primitive eq? takes two arguments. Each must be a non–numeric atom.
(define lat?
  (lambda (l)
    (cond
      ((null? l) #t)
      ((atom? (car l)) (lat? (cdr l)))
      (else #f)))))
Functions

(define member?
  (lambda (a lat)
    (cond
      ((null? lat) #f)
      (else (or (eq? a (car lat))
                (member? a (cdr lat)))))))
Functions

(define member?
  (lambda (a lat)
    (cond
      ((null? lat) #f)
      ((eq? a (car lat) #t)
       (else (member? a (cdr lat))))))))
(define function
  (lambda (parameters)
    (cond
      (q1 ans)
      (q2 ans)
      ...
      (else ans))
  )
)
The First Commandment

( preliminary )

Always ask null? As the first question in expressing an function.
The Second Commandment

Use *cons* to build lists.
The Third Commandment

When building a list, describe the first typical element, and then cons it onto the natural recursion.
The C Method

```c
main (char a[], char lat[][[]]) {
    for (int i = 0; i < alen(lat); i++) {
        if (strcmp(a, lat[i]))
            return 1;
    }
    return 0;
}
```
Eating a Loaf of Bread

Are you done? Then Stop.
Else, cut a slice and eat it.
Repeat.

(define eat-a-loaf
  (lambda (loaf)
    (cond
      ((null? loaf) 'done)
      (else (begin (eat (car loaf))
                    (eat-a-loaf (cdr loaf)))))))
(define rember
  (lambda (a lat)
    (cond
      ((null? lat) '())
      ((eq? a (car lat)) (cdr lat))
      (else (cons (car lat)
                   (rember a (cdr lat))))))))