```
PARENTHESES EVERYWHERE FORMAT CODE WELL

scheme
    atoms
        1 "ab"
    lists
        (atom, list)
        either atom or list


procedure
    put it in a list
    takes first element and treats it as a procedure
        tries to evaluate all subsequent items in list
        (+ 1 2)
        3
        (+ 1 2 3 4 5)
        15
        (+ 1 2 3 (+ 99 2) 5)
        112
    natural tree structure to it

scheme lists are really linked lists
"cons cells"
    create single cell with the cons operator

    list by definitione
        last element points to an empty list () atom
        (cons 1 '())

    non list
        (cons 1 2)
        '(1.2)
        . indicates "the rest"

    (quote ( ))
        no evaluation on its arguments, returns the list
        ' is short hand for quote

        (cons '(1 2) '(3 4))
        >'((1 2) 3 4)

    head and tail:
        head: car
        tail: cdr
```

```
> (car '(1 2 3))
1
> (cdr '(1 2 3))
'(2 3)

will not work on the empty list:
    empty list is not technically a list, it's an atom

#t is true

(number? 5)
#t
(symbol? 'a)
#t

pair?
list?

list procedure:
    creates a list after evaluating shit

top level name definitions
(define x 1)
(define y 3)
(+ x y)
4

functions
(define (procedure_name args)
    (program)
)

(lambda (arguments)
    (expressions to run)
)

can use lambda always if we so desire

(if predicate
    then_part
    else_part
)

cond: like switch
```

```
(cond ((predicate) (expr)) ((predicate) (expr)) (else ()))

= numerical equivalence

eq? if objects are the same (ie address is the same)

eqv? type and value

equal? for sequences and strings


(let (bindings) (body))
evaluates to the result of the last one


(define (apnd l e)
    (if (null? l)
        (list e)
        (cons (car l) (apnd (cdr l) e))
    )
)

(define (apendl l1 l2)
    (if (null? l2)
    l1
    (apndl (apnd l1 (car l2)) (cdr l2))
    )
)
    really fucking slow

use library append, this shit too naive
```