Ex 4.6

ve body

(let ((v1 e1) (v2 e2) ...) (...))

((lambda (v1 v2 ...) (...)) e1 e2 ...)

```
(define (let->combination exp)
        (cons (list 'lambda (map car (ve exp)) (body exp))
              (map cadr (ve exp))))
(define (ve exp) (cadr exp))
(define (body exp) (cddr exp))
(define (let? exp) (tagged-list? exp 'let))
```

tagged-list? ↓ if list beg with let

→ Add to eval:

((let? exp) (eval (let->combination exp) env))

(let ((x 3)) (let ((y 4)) (let ((z 2)) (body))

Ex 4.7

```
(define (let*->nested-lets exp)
    (define (letloop ve) (if (null? ve) (car (body exp))
                             (cons (list 'let (car ve))
                                   (letloop (cdr ve)))))
    (letloop (ve exp)))
```

(let* ((v1 e1) (v2 e2) ...) (body))

=> (let ((v1 e1)) (let ((v2 e2)) ... (body))))...

yes it is enough since it will convert to a let expression and be evaluated accordingly.

Ex 4.13

```
(define (make-unbound! var env)
  (define (scan vars)
    (cond ((null? vars) #t)
          ((eq? var (car vars)) (begin
            (set-car! vars null)) (set-car! vals null))
          (else (scan (cdr vars) (cdr vals)))))
  (let ((frame (first-frame env)))
    (scan (frame-variables frame))))
```

Ex 4.15

(p a) —halts→ value

—not→ error / forever

(halts? p a)

```
(define (try p)
  (if (halts? p p) (run forever) → if halts p on p
      'halted))
```

try does the inverse behavior of the procedure.

Assume (halts? try try) halts then (try try) will run forever.

Assume (halts? try try) runs forever then (try try) will halt.

These 2 assumption + outcome contradrict each other.

They also assume existance of halts?, So halts? is not possible.

- if / cond

```
(if (= n 1) 1 (* n (fact (- n 1))))
(cond ((= n 1) 1)
      (else (* n (fact (- n 1)))))
```

```
(factorial 3)
(if (= 3 1) 1 (* 3 (factorial (2))))
                   if (= 2 1) 1 (factorial 1)
                                if (= 1 1) 1 (factorial 0)
```

forever

since if is a special form, it evaluates only one of its parts. if we redefine it and send the function the 2 parts, it will try to evaluate both parts before calling on the function if. we will get to an infinite loop.