Ex 2.4

```
(define (cons x y)
  (lambda (m) (m x y)))
(define (car z)
  (z (lambda (p q) p)))
(define (cdr z)
  (z (lambda (p q) q)))
```

```
(car (cons x y))
(car (lambda (m) (m x y))
((lambda (m) (m x y)) (lambda (p q) p))
((lambda (p q) p) x y)
(x)
```

Ex 2.22

```
(define (square-list items)
  (define (iter things answer)
    (if (null? things) answer
        (iter (cdr things) (cons (square (car things))
                                 answer))))
  (iter items nil))
```

because he's doing (cons (square (car things)) answer)
This will square the car and pass it as answer which will place it at the end of the next call.

(iter (cdr things) (cons answer (square (car things))))

The second attempt will print (((() . x) .y) . z) ...
This is becaus he is doing cons of a list with an argument which does not end in null. Instead, he should cons the whole list arguments together and add the extra one after.

✓ Ex 2.25

(1 3 (5 7) 9)

(car (cdr (car (cdr (cdr lst)))))

((7))

(car (car lst))

(1 (2 (3 (4 (5 (6 7))))))

(car (cdr (car (cdr (car (cdr (car (cdr (car (cdr (car (cdr lst)))))))))))))

✓ Ex 2.26

x → (1 2 3)

y → (4 5 6)

(append x y) → (1 2 3 4 5 6)

(cons x y) → ((1 2 3) 4 5 6)

(list x y) → ((1 2 3) (4 5 6))

✓ Ex 2.27

```
(define (deep-reverse lst)
  (cond ((null? lst) lst)
        ((pair? (car lst))
         (append (deep-reverse (cdr lst)) (list (deep-r (car lst)))))
        (else ( " " " " (list (car lst))))))
```

Ex 2.28

```
(define fringe (tr)
  (cond ((null? tr) tr)
        ((pair? (car tr)) (append fridge (car tr) fridge (cdr tr)))
        (else tr)))
```