Notes 2.29 - 2.35

Data Abstraction
- Understand the abstraction and prompt before writing corresponding procedures
- When changing things, only change abstraction not the procedures

Square-list
- Similar thinking to reverse
- Comes up often
- How to solve
  - First condition → empty
  - Second condition → number
  - Else, cons the func invoked on car and then cdr
- Can also abstract to work on several procedures, not just square

Map
- Applies procedure to every element of a list
- Often use lambda instead of defining a helper

Conventional Interfaces
- Use accumulate to replace other functions
  - Map, length, append
  - This was especially hard (**2.33**)
- As well as to define other functions
  - Sum-odd-squares, even-fib, etc.
- Apply op to sequence until it is empty, then use initial

- Accumulate, map, lambda together can be very powerful in creating functions

```
(define (accumulate op initial sequence)
  (if (null? sequence)
      initial
      (op (car sequence)
          (accumulate op initial (cdr sequence)))))
```

Map & Accumulate
- Write a helper function and call it in map
  - In else clause: (op (func (car x)) (func (cdr x)))

- Or use lambda
    - In else clause: (func x)