Week 2 Lecture 4

Theory

Getting Ready

- Feel good about Lecture 3
- Read SICP Section 2.1 closely

What's in this lecture?

- Using Data Abstractions

Abstraction

- Abstraction creates a boundary between different parts of an application
- Components use a simplified interface, rather than having to know the internal details of operation
- Choosing abstractions wisely is an *art form*

Abstraction 1: Vehicle

- Vehicle has: accelerator, brake, steering wheel, automatic transmission "functions"
- Details of "electric vs. internal combustion," "disc vs. caliper," etc. are hidden

Abstraction 2: Cell Phone

- Cell Phone supports the "telephone" interface: call a phone #, receive a call, hang up
- Also supports the "SMS" interface: send and receive SMS messages
- The details of radio transmission, data encoding, cell tower handoff are hidden

Sample Abstractions

- 2-D Vector ([x, y] relative to [0,0])
- Rational number (expressible as a/b)
- Complex number (expressible as x + iy)

Pairs

```
> (cons 1 2)          ;; creates a new pair [1,2]
;Value 11: (1 . 2)

> (car (cons 3 4))   ;; returns first element
;Value: 3

> (cdr (cons 5 6)) ;; returns second element
;Value: 6
```

Length Function

- Length of Vector [x,y] is (thanks Pythagoras):

  (sqrt (+ (square x) (square y)))

- But! We don't want to expose details...

  (define (length v) ...) ;; where v is vector

Vector Representation

```
(define (make-vector x y) (cons x y))

(define (length v)
  (let ((x (car v))
        (y (cdr v)))
    (sqrt (+ (* x x) (* y y)))))
```

Now you try...

```
(define (add v1 v2) ...)  ;; sum of 2 vectors
(define (sub v1 v2) ...)  ;; diff of 2 vectors


;; angle separating 2 vectors
(define (angle-sep v1 v2) ...)
```

Exercises

- SICP 2.1, 2.2, 2.3, 2.4