Week 1 Lecture 2

Theory

Getting Ready

- Feel good about Lecture 1
- Read SICP Seciton 1.2 closely

What's in this lecture?

- Recursion & Iteration
- Building more interesting programs

Resources (these help)

- MIT OpenCourseWare 6.001 2005SP
- Lecture Notes
- Video Lectures
- Development Tools

Scheme: Recursion

```
(define (f) (f)) ;; infinite recursion

(define (f x)  ;; recurses x times
  (if (> x 0)
    (f (- x 1))))

(define (factorial n)
  (if (= n 1) 1
    (* n (factorial (- n 1)))))
```

Scheme: Iteration 1

```
(define (factorial n)
  (fact-iter 1 1 n))

(define (fact-iter prod count max-count)
  (if (> count max-count)
    prod
    (fact-iter
      (* count prod)  ;; new product
      (+ count 1)     ;; new count
      max-count)))    ;; max count is constant
```

Scheme: Let

```
(define (f x y)          ;; f(x) = (x*(x+1)) + (y*(y+1))
  (let ((x1 (+ x 1))
        (y1 (+ y 1)))
    (+ (* x x1) (* y y1)))))
```

```
(define (awesome person num-times)
  (if (> num-times 0)
    (let () ;; no bindings
      (write-line
        (string-append person " is awesome!"))
      (awesome person (- num-times 1)))))
```

Scheme: Iteration 2

```
(define (f x y)
  (f-iter 0 x 0 y))

(define (f-iter i x j y)
  (let ()
    (write-line ".")
    (if (< j y)
      (cond ((< i x) (f-iter (+ i 1) x j y))
        ((= i x) (f-iter 0 x (+ j 1) y))))))
```

Sample: Fib 1

```
;; tree recursive form
(define (fib n)
  (cond ((= n 0) 0))
        ((= n 1) 1))
        (else (+ (fib (- n 1))
                 (fib (- n 2))))))
```

Scheme: Tail Recursion

```
;; 2-dimensional iteration
(define (f x y)
  (f-iter 0 x 0 y))

(define (f-i i x j y)
  (let ()
    (write-line ".")
    (if (< j y)
      (cond ((< i x) (f-i (+ i 1) x j y))
        ((= i x) (f-i 0 x (+ j 1) y))))))
```

Sample: Fib 2

```
;; tail-recursive form
(define (fib n) (fib-iter  1 0 n))

(define (fib-iter a b count)
  (if (= count 0) b
      (fib-iter (+ a b) a (- count 1))))
```

Scheme: Orders of Growth

- How many calls to fib in tree-recursive form?
- How many calls to fib-iter in tail-recursive form?
- How many calls to f-i in 2-dimensional iteration case?

Sample: Infinite Game

```
(define (game) (center-boat))

(define (center-boat)
  (let ()
    (write-line "You are in the center of a boat. Move left or right?")
    (let ((d (read-line)))
      (cond ((string=? d "left") (side-boat d))
            ((string=? d "right") (side-boat d))
            (else (center-boat))))))

(define (side-boat side)
  (let ((otherside (if (string=? side "left") "right" "left")))
    (write-line
      (string-append "You are at the " side " of the boat. Move left or right?"))
    (let ((d (read-line)))
      (cond ((string=? d side) (ocean))
            (else (side-boat otherside))))))

(define (ocean)
  (write-line "You fell in the ocean! Game Over."))
```

Exercises

- SICP 1.9, 1.11, 1.12, 1.16, 1.20