6.090, Building Programming Experience
Ben Vandiver
Lecture 2: Procedures and Recursion

Review:
Self-evaluating expression
  #t, 5, "yay"
Name expression
  +, foo, bar

N	V
foo	4

combinations (+ 3 4) to evaluate a combination, sequence of steps:
Evaluate operator
Evaluate arguments
Apply operator value to argument values
Order of evaluation of arguments unspecified, but scheme does it right-to-left

Special forms:
define
  (define name value)

  (define bar (+ 3 3))

if
  (if test consequent alternative)

Make mental pictures for better gasp abstract more difficult to hold

Lambda:

((lambda (x) (* x x)) 5) compound procedure in a combination

↓

(* 5 5) => 25

Lambda procedures are lengthy; in order to reuse over and over again with simplicity, give name.

```
(define square
->(lambda (x)
--->(* x x)))
```
Notice indentations

All code now must be tabbed accordingly: newly opened paren is where tab should begin. In scheme, simply hit the return key and then hit the tab key, and the correct tab will automatically appear in the code.

Writing Procedures

Not takes one argument, it negates what you see
For example:
Not true-ish is false
Not false is true

```
(define not
  (lambda (s)
    (if s
        #f
        #t)))
```

Evaluation of (not #f):

- (not #f)
- ((lambda (s) (if s #f #t)) #f)
- (if #f #f #t)
- #t

When you come across a problem → break into smaller, more manageable problems

Factorial
n! = (n – 1)! * n
0! = 1

```
(define fact
  (lambda (n)
    (if (= n 0)
        1
        (* n (fact (- n 1))))))
```

Use substitution model to investigate this
If scheme gets stuck in an infinite loop, exit with C-c, C-c.

- (fact 2)
- (if (= 2 0) 1 (* 2 (fact (- 2 1)))
- (* 2 (fact 1))
- (* 2 (if (= 1 0) 1 (* 1 (fact (- 1 1)))))
- (* 2 (* 1 (fact 0)))

- (* 2 (* 1 (if (= 0 0) 1 (* 0 (fact (- 0 1))))))
- (* 2 (* 1 1))
- (* 2 1)
- 2

Defined factorial in terms of itself: this is known as recursion.

Exponentiation
x^n = x^(n-1)*x
X^0 = 1

```
(define expt
  (lambda (x n)
    (if (= n o)
        1
        (* x (expt x (- n 1))))))
```

Arithmetic operator facts:
Arithmetic operators take as many arguments as you want to enter.

(/ 5) → 1/5
(/) → error
(-) → error
(+) → 0
(*) → 1

In writing recursion, steps to take:
Base case:
Recursive case:

Recursion like Proof by Induction
If we have the sum from i=0 to i=n of 2^i, the solution is $2^{n+1} - 1$. To prove this, we choose a base case:

the sum from i=0 to i=0 of 2^i
this gives 2^0, which is 1
putting 0 into the expression $2^{(n+1)} - 1$, we get $2 - 1 = 1$, which agrees with $2^0 = 1$

We assume that we know the following:

i=0 to i=n of $2^i = 2^{n+1} - 1$

Now we need to show that:

i=0 to i=(n+1) of $2^i = 2^{n+2} - 1$

Using what we know, and combining with 2^{n+1}, we have
i=0 to i=n of $2^i + 2^{n+1} = 2^{n+1} - 1 + 2^{n+1} = 2*2^{n+1} - 1 = 2^{n+2} - 1$

Homework
Do all the problems on the lecture 2 handout. Solutions are posted under lecture 2 solutions on the server.