CSSE 304 Day 07 Summary

1.
```
(define cube (lambda(x) (* x x x)))

(define apply-many

(lambda (functions arg)
     (map (lambda (function)
           (apply function (list arg)))
          functions))
(apply-many (list - cube (lambda (x) (/ x 2))) 3)

(apply-many '(- cube (lambda (x) (/ x 2))) 3)
(apply-many `(,- ,cube ,(lambda (x) (/ x 2))) 3)

(apply + 1 2 '(3 4 5)) ; a different form of apply?
```

2. After class: Look over the solutions for matrix-transpose, minimize-interval-list, pascal-triangle, and largest-in-lists. Write down questions that you have about them and ask someone (possibly me) sometime.

3. Write `(all-positive? lon)` using `map`. Why is this hard? How do we get around this?

4. What do we mean by the *syntax* of a programming language?

   What do we mean by the *semantics* of a programming language?

   The left-hand side of a context-free (BNF) grammar production always contains a single _____ symbol.

   A string that is in the language of a grammar consists of (circle one):
   terminals only nonterminals only both terminals and nonterminals

5. A **BNF Grammar** (a.k.a. Context-free grammar) example:
   a. **Nonterminals:** <exp> <term> <factor> <number> <digit>
   b. **Terminals:** + *) (0 1 2 3 4 5 6 7 8 9
   c. **Start Symbol:** <exp>
   d. Productions:
      i. <exp> ::= <exp> + <term> | <term>
      ii. <term> ::= <term> * <factor> | <factor>
      iii. <factor> ::= (<exp>) | <number>
      iv. <number> ::= <number> <digit> | <digit>
      v. <digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

6. Show a derivation and a derivation tree for `1 * (2 + 34)` from `<exp>` [you will need to write small]

What is the meaning of Kleene * ? Kleene $^{+}$?

7. The slides contain (all in one place, the **collection of grammars** used in chapter 1 of EoPL.
   e. <list-of-numbers> ::= ({<number>}*)
   f. <s-list> ::= ({<symbol-exp>}*)
      <symbol-exp> ::= <symbol> | <s-list>
   g. <bintree> ::= <number> | (<symbol> <bintree> <bintree>)
   h. <BST> ::= ( ) | (<number> <BST> <BST>
   i. <datum> ::= <number> | <symbol> | <string> | <boolean> | <dotted-datum> | <list> | <vector>
      <list> ::= ({<datum>}*)
      <dotted-datum> ::= ({<datum>}$^{+}$. <datum>)
      <vector> ::= # <list>
   j. <LcExp> ::= Identifier | ; variable reference
      (lambda (Identifier) <LcExp>) | ; abstraction
      (<LcExp> <LcExp>) ; application

8. The s-list grammar can also be written as <s-list> ::= () | (<s-exp> . <s-list>)
   <s-exp> ::= <symbol> | <s-list>

9. (probably tomorrow) we will write and test some of these procedures:
   a. contains?
   b. count-occurrences
   c. notate-depth
   d. flatten
   e. subst