CSSE 304 Day 06 Summary

1. Use map and apply for previous programs:

```
(define ms-size
    (lambda (ms)
        (apply + (map cadr ms))))

(define sorted? ; with-normal-less
    (lambda (lon)
       (or (null? lon)
           (apply <= lon))))
```

2. **Map and apply examples**

a. `(map < '(1 5 7) '(2 4 6))`

b. `(map list '(1 5 7) '(2 4 6) '(0 8 3))`

c. `(apply cons '(2 3))`

d. `(list '())`   `(map list '())`   `(apply list '())`

e.
```
(define ms-size
    (lambda (ms) (apply + (map cadr ms))
```

f.
```
(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)
```

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

3. Given the box-and-pointer diagram on the slide, how would Scheme output this object?

Try to write code that creates this object without using `quote`.

More Practice with box-and**-pointer diagrams:** Draw the diagrams for the structures that get created when the following code is executed, then show what it outputs. What if we then do `(set-cdr! v v)`?
Suggestion: Work with another student.

```
(define x '((1 2) 3 (4 5)))
```

```
(define y (cons (car x) ( cdr x)))
```

```
(define z (cons (cdr x ) x))
```

```
(define t (append y x))
```

```
(write x) (newline)

(write y) (newline)

(write z) (newline)

(write t) (newline)

(set-cdr! x x)

(write x) (newline)
```

4. What does the box-and pointer diagram for `'(())` look like?
   How about `'((()))`, `'(((())))`, and `'((())())` ?

5. With another student (pair programming) write `largest-in-lists`, which takes a list of lists of numbers and returns the largest number. Returns `#f` if there are no numbers in any of the lists. Don't use any *separate* recursive helper procedures (instead get practice with `letrec` and/or named `let`). **You may want to test it with some simpler lists before trying the test cases on the PLC server.**

```
(largest-in-lists '((1 3 5) () (4) (2 6 1) (4))) ➔ 6
(largest-in-lists '(() ())) ➔ #f
```