CSSE 304 Day 04 Summary

1. Factorial examples

```
> (define fact
     (lambda (n)
       (cond
        [(zero? n)  1]
        [else (* n (fact
                (- n 1)))])))
> (fact 4)
24
> (fact  -2)
  C-c C-c
break>q
```

Note that pressing Ctrl-c multiple times will break Scheme out of an infinite loop

```
> (trace
fact)
(fact)
> (fact 4)
|(fact 4)
| (fact 3)
| |(fact 2)
| | (fact 1)
| | |(fact 0)
| | |1
| | 1
| |2
| 6
|24
24
```

```
> (define fact2
   (lambda (n)
    (if (or (negative? n)
            (not (integer? n)))
        "error"
        (fact-acc n 1))))
> (define fact-acc
   (lambda (n acc)
    (if (zero? n)
        acc
        (fact-acc (- n 1)
                  (* n acc)))))
```

```
> (trace fact fact2
fact-acc)
(fact fact2 fact-
acc)
> (fact2 4)
|(fact2 4)
|(fact-acc 4 1)
|(fact-acc 3 4)
|(fact-acc 2 12)
|(fact-acc 1 24)
|(fact-acc 0 24)
|24
24
```

A place for your notes on the fact examples:

2. Make-adder example:

```
> (define make-adder
    (lambda (m)
      (lambda (n)
         (+ m n))))
> (define add5 (make-adder 5))
> add5
#<procedure>
> (add5 8)
13
> ((make-adder 5) 8)
13
> (((lambda (m)
     (lambda (n)
      (+ m n)))
     5)
   8)
13
```

3. `cons` vs. `list` vs. `append` (box-and-pointer diagrams)

```
(define x '(1 2 3))     (define y '(4 5))   (define z '(6 7))
```

```
(cons x y)
```

```
(list x y)
```

```
(append x y z)
```

4. **apply** applies a procedure to the elements of a list: **(apply cons '(2 4))** is the same as **(cons 2 4)**.
   Other examples: [if apply did not exist, could we write it?]

5. **(make-list n obj)** returns a list of **n** "copies" of obj. [If **obj** is a "by-reference" object, such as a list, it makes **n** copies of the reference].

6. **(firsts '((a b) (c d) (e f))) ➔ (a c e)**

7. **Map-unary** applies a unary procedure to every element of a list and returns the list of the results**.**
   (map-unary positive? '(2 -1 3 4)) ➔ (#t #f #t #t).

8. Use **map-unary** to rewrite **firsts**. [Note: **map-unary** has the same interface as built-in procedure **map**]

9. **(positives '(1 -3 6 0 2 -1 7)) ➔ (1 6 2 7)**

10. `(sorted? <= '(3 4 2 6)) ➔ #f` ; Hint: Use `or` and `and`.
    `(sorted? >= '(4 3 2 1)) ➔ #t` ; assume 2nd arg is a list of numbers