CSSE 304 Day 04 Summary

1. `cons` vs. `list` vs. `append` (box-and-pointer diagrams)
   a. ***`(define x '(1 2 3))`*** **`(define y '(4 5))`**

   b. `(cons x y)`

   `(list x y)`

   `(append x y)`

2. **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?]

3. Reflexive pairs.
   A **relation** is a set of ordered pairs; the set of all first elements is the **domain**. The set of all second elements is the **range**.
   We represent a relation by a list of 2-lists. A **2-list** is a list whose length is 2.
   A **reflexive pair** is a 2-list whose first and last elements are the same.
   count-reflexive-pairs (work it out live)

4. **`(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].

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

6. **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).`**

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

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

9. `(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`