Goals of this tutorial

You should be able to...

• write mutually recursive functions based on the relevant templates.
• understand the syntax and semantics of local.
• write functions that consume and/or produce other functions

Group problem - mutually recursive templates

For this example, we are going to use general trees to store employee information for an organization. Consider the following definitions:

```
(define-struct orgnode (name employees))
;; An OrgChart is a (make-orgnode Str (listof OrgChart))
;; requires: the names are unique
```
Group problem - mutually recursive templates

Here is what cs135-staff looks like:

Write a template function for each of OrgChart and (listof OrgChart).
Group Problem - list-employees-at
Using your templates, write a function list-employees-at that consumes a positive integer, n, and an OrgChart. It produces a list of all the employees at level n in the OrgChart, where the root node is at level 1. The list may be in any order.

(list-employees-at cs135-staff 1) ⇒ ("Karen")
(list-employees-at cs135-staff 3) ⇒ ("Dustin" "Zainab" "Vincent")
(list-employees-at cs135-staff 5) ⇒ ()

Group Problem - add-chain-of-command
Write a function add-chain-of-command that consumes a non-empty list of strings, chain, and an OrgChart. chain represents a chain of command of employees in the OrgChart, starting from the topmost employee in the OrgChart. The function should produce the same OrgChart but with any new employees in chain added to it, in that order.

Note that adding the chain may not replace any employees already in the OrgChart and it may not create any duplicate employees.
Group Problem - add-chain-of-command

(add-chain-of-command ("Karen" "Paul" "Vincent" "Ian") cs135-staff) =>

Review: Local Definitions

Recall the special form local which allows us to create local definitions. The syntax for local is as follows:

(local [definition_1 ... definition_n] expression)

where each definition is a define statement, and expression is a Racket expression that uses these definitions.
Stepping Problem - Local

Provide a step-by-step evaluation of the following program. When renaming local definitions, append “.0” if possible, or else “.1”, “.2”, etc. Do not recopy any line that is already in its simplest form.

```
(define (f x y)
  (local
    [[(define a (+ x 3))
      (define y 4)
      (define (g x)
        (+ x a))
      (+ 2 (g y))]])
  (∗ 2 (g y))))
(f 2 6)
```

CS135 Fall 2017 Tutorial 8: Mutual recursion and local
(define a₀ (+ 2 3))
(define y₀ 4)
(define (g₀ x)
  (+ x a₀))
⇒ (∗ 2 (g₀ y₀))
⇒ (∗ 2 (g₀ 4))
(define a 0 5)
(define y 0 4)
(define (g 0 x)
  (+ x a))
⇒ (* 2 (g 0 y))
⇒ (* 2 (g 0 4))
⇒ (* 2 (+ 4 a))
⇒ (* 2 (+ 4 5))
⇒ (* 2 9)
⇒ 18
Clicker Question - Local Definitions
In Intermediate Student, what would this code produce?

```
(define a 10)
(define b
  (local [(define a 5)]
    (add1 a)))
(+ a b)
```

A 10  
B 15  
C 16  
D 21  
E An error
Group Problem (optional) - my-sort

Write a function my-sort that consumes a list and a comparison function, and produce a sorted list in the order according to the comparison function.

Use insertion sort.

(my-sort (list 3 4 1 8 5) <) ⇒ (list 1 3 4 5 8)
(my-sort (list "b" "a" "d" "e" "c") string < ?) ⇒ (list "a" "b" "c" "d" "e")