Below are several exercises that we will be covering in the upcoming tutorial (Friday, Mar 4). We will release these exercises in advance of the tutorial so that you get a chance to attempt the exercises yourself before we discuss them in tutorial. The course staff running the tutorial will go through each problem (time permitting) and show their process for how they would complete these questions in an assignment/exam setting. Being familiar with the questions before attending tutorial will help to ensure that you get the most out of our explanations.

Note that tutorials are not mandatory in CS135. We never cover any new material that you won’t have seen in lectures, and instead choose to highlight techniques and concepts from the most recent lectures to give you some extra practice. If you are very comfortable with the material that we covered and could easily complete the below questions, then you aren’t obligated to attend tutorials. Be very cautious with this though since sometimes questions can seem easy until you actually sit down and attempt them.

**Question 1: Counting Keys in Binary Search Trees**

Note: In [Module 11 (Slide 22)](#), you learned about binary search trees. You should use the data definitions for BST and Node defined on Slide 23 when solving this problem.

Write a function `(count-range bst low high)` which consumes a binary search tree bst and two natural numbers low and high. The function should produce the number of keys in the binary tree whose value is at least low and at most high. Try to solve this problem efficiently (i.e., without searching paths that cannot contain a valid key).

*Examples:*

```lisp
(define example-tree (make-node 5

    (make-node 1 empty empty)

    (make-node 7

        (make-node 6 empty empty)

        (make-node 14 empty empty)))))

(count-range example-tree 6 7) → 2

(count-range example-tree 0 100) → 5
```

**Question 2: Decorating Cookies**
You’ve been hired as a cookie decorator for children’s birthday parties. Each cookie is a simple shape, either a triangle or a square. Your clients give you an exact list of the cookies they would like you to decorate, but you’re not sure you’ll have enough icing to do the job.

Write a function \( \textit{area-to-decorate cookies} \) which consumes a list of cookies and produces the sum of the surface areas of all cookies: the amount of icing you will need to decorate them. You may use the following definitions when writing this function:

- The area of a square is equal to its length squared.
- The area of a triangle is \((\text{base} \times \text{height})/2\).

You may also use the following code and data definitions to start writing your solution:

```scheme
;; A Cookie is a Triangle or a Square.
(define-struct triangle (base height))

;; A Triangle is a (make-triangle Num Num).
;; Requires: base > 0, height > 0
(define-struct square (length))

;; A Square is a (make-square Num).
;; Requires: length > 0
```

\textit{Example:}

```scheme
(define example-triangle (make-triangle 2 4))
(define example-square (make-square 1))
(area-to-decorate (list example-triangle example-square example-square)) → 6
```