Goals of this tutorial

You should be able to...

- Modify fields of Structures.
- Understand Binary Trees (BT) and Binary Search Trees (BST).
- Write a Template for a BT/BST.
- Traverse a BT/BST.
Announcements

- Office hours have been changed starting this week.
- New times and locations of office hours are posted on the “Help” → ”Office and Consulting Hours” page of the course website.

Please email us at cs135@uwaterloo.ca to set up an appointment outside of these hours.
Clicker Question: Debugging Structure Templates

How many errors are there in the following template and data definition?

```
(define-struct student (name id average))
;; A Student is a (list Str Nat Num)
;; id has eight digits, 0 ≤ average ≤ 100

;;;(define (student-template student)
;;; (cond
;;; [... (first student) ...]
;;; [else (... (student-id student)... (student-average))]))
```

A  It looks perfect!
B  2
C  4
D  6
E  Too many to count.
Problem 1: Update Structures

UW now allows students to go by their preferred name! Write a function update-name that consumes a (listof student) los, a Nat id, and a Str new-name. If there is an element in los that has the same id as id, update-name updates the name field of it to new-name. You may assume that the id field in los is unique. Design recipe is not required. Examples:

(define Roger (make-student "Roger" 20760000 77))
(define Alice (make-student "Alice" 20729947 99.9))
(define Aksha (make-student "Aksha" 20760000 77))
(define Eavan (make-student "Eavan" 20859999 100))
(update-name (list Roger Alice Eavan) 20760000 "Aksha") → (list Aksha Alice Eavan)
(update-name empty 20760000 "Aksha") → empty
Problem 2: Binary Trees

Using the data definition provided write a template for a binary tree.

```
(define-struct bt (key left right))
;; A binary tree (BT) is any of:
;; * empty
;; * (make-bt Nat BT BT)
```
Problem 2: Replace

Using template for a bt write one function called `replace-bt` that consumes a binary tree and two numbers called `old-key` and `new-key`. It will replace every occurrence of `old-key` with `new-key`. 
Problem 2: Design Recipe Part 1

;;; (replace-bt tree old-key new-key) produces a new tree with all
;;; old-key in it replaced with the new-key.
;;; replace-bt: BT Nat Nat → BT
Problem 2: Design Recipe Part 2

;; Example:
(check-expect (replace-bt (make-bt 1 (make-bt 2 empty empty) (make-bt 3 empty empty)) 3 0) (make-bt 1 (make-bt 2 empty empty) (make-bt 0 empty empty)))

;; Test:
(check-expect (replace-bt (make-bt 1 empty empty) 2 0) (make-bt 1 empty empty))
(define-struct bst (key left right))

;; A BST is one of:

;; * empty

;; * (make-bst Nat BST BST)

;; requires:

;; key > all left BST keys

;; key < all right BST keys
Clicker Question 3: Identify the BST

A)  
\[ \begin{array}{c}
3 \\
2 \\
1 
\end{array} \]

B)  
\[ \begin{array}{c}
1 \\
2 \\
3 
\end{array} \]

C)  
\[ \begin{array}{c}
2 \\
1 \\
3 \\
4 
\end{array} \]

D)  
\[ \begin{array}{c}
4 \\
2 \\
3 \\
5 
\end{array} \]

E)  
\[ \begin{array}{c}
3 \\
2 \\
4 \\
4 
\end{array} \]
Problem 3: Tree Range

The full range of a tree is defined by the difference between its smallest and its largest keys. Define a function that calculates the difference between the minimum and maximum keys found in a non-empty BST.

Your first step should be to write a helper function that finds the smallest key in a BST, call this function tree-min.
Extra Practice: Boolean Expressions

Boolean expressions can be represented by nested lists. Consider the following data definition for a boolean expression:

```scheme
;; A boolean expression (BoolExp) is one of:
;; * (anyof 'true 'false)
;; * (cons (anyof 'and 'or) (listof BoolExp))
```

Examples:

'true

'(or true (and true true false))

Write template functions for BoolExp and (listof BoolExp).