Goals of this lab:

- More practice with recursion
- Learn how to use \textit{eval} expression

1. Write a Scheme function \textit{code-modifier} that takes three arguments, two atoms \textit{a}, and \textit{b}, and a list \textit{expr} that holds an expression. Your function should evaluate to a list which holds another expression such that all occurrence(s) of the first atomic argument, \textit{a}, is/are replaced with the second atomic argument, \textit{b}. You can use the following \textit{atom?} function that we had defined in lab2 as a helper function.

\begin{verbatim}
(define atom? (lambda (x)
    (and (not (null? x))
        (not (pair? x))))
)
\end{verbatim}

Examples:

\begin{verbatim}
(code-modifier '3 '7 '(* 3 (+ 3 5))) => (* 7 (+ 7 5))
(code-modifier '+ '* '(lambda (x) (+ x x))) => (lambda (x) (* x x))
\end{verbatim}

Solution:

\begin{verbatim}
(define code-modifier (lambda (a b expr)
    (cond
        ((null? expr) '()) ; base case
        ((atom? expr) (if (eqv? a expr) b expr)) ; base case
        (else
            (let ((x (code-modifier a b (car expr)))
                  (y (code-modifier a b (cdr expr))))
                (cons x y))))))
\end{verbatim}
2. eval expression. (Switch to "Standard (R5RS)" language since eval expression is not defined in "Intermediate Student with lambda". In DrScheme environment, Click Language -> choose Language -> Standard (R5RS).)

Syntax:
(eval expression)

Evaluates expression and returns its value. expression must be a valid Scheme expression represented as data such as a list.

Examples:

(eval '(+ 7 (+ 7 5)))
=> 84

(let ((f (eval '(* 5 (+ 5 5))))
     (f * 5)) ; let body
=> 25

What do the following expressions evaluate to? Please, fill in the blanks.

(eval '(- 7 12 5)) => _-10_
(eval (cons '- '(7 12 5))) => _-10_
(eval (cons '- (cdr '(7 12 5)))) => __7__

Following expressions evaluate to 13, and 15, respectively. However, the expressions are not complete. Please, fill in the missing code.

(eval (cons '+ (cons (eval '(+ 3 4)) (cons (eval '(* 2 3)) _'()_))) => 13
(eval (cons '+ (cons (eval '(* 2 3)) (cons (eval '(+ 3 4)) _'(2)_))) => 15

Now, using the eval expression, and your code-modifier function, write an expression that modifies the expression, (lambda (x y) (+ x y)) to (lambda (x y) (* x y)) and evaluates this modified function with the arguments 3, and 5. The result should be 15. (Hint: Think about building a convenient expression before passing it to the eval expression as an argument.)

Solution:

(eval (cons (code-modifier '+ '* 'lambda (x y) (+ x y))) '(3 5)))