Lab 15: Racket Functions

For this exercise, you may use **ONLY** the expressions included in the slides (both sets 1 and 2).

Your functions need to work **ONLY** for the test cases provided.

1. Let ***x*** and ***y*** be **lists**. Write a function ***func1*** that evaluates to the first element of the list ***x*** if the list ***x*** is non-null, or to the cons of ***x*** onto the list ***y*** otherwise.
   Example:
   ```
   (func1 '() '(1 2 3))              => '(() 1 2 3)
   (func1 '(1 2 3) '(4 5 6 7))       => 1
   ```

2. Let ***w*** be a **non-null list** containing at least three elements. Write a function ***new-list*** that evaluates to a new list obtained from ***w*** by exchanging its first and third elements.
   Example:
   ```
   (new-list '(1 2 3))           => '(3 2 1)
   (new-list  '(1 2 3 4))        => '(3 2 1 4)
   (new-list '((1 2) (3) (4)))       => '((4) (3) (1 2))
   ```

3. You will have a question similar to this one on the final exam. Given the input below, trace by hand the following **recursive** functions and write the output as a comment in the .rkt file you are submitting.

   Input: `(this-function 'a '(one () (three)))`

   ```
   (define this-function
     (lambda (a s)
       (cond
         [(empty? s) '()]
         [else (cons a (this-function a (rest s)))])))
   ```

4. You will have a question similar to this one on the final exam. Given the input below, trace by hand the following **recursive** functions and write the output as a comment in the .rkt file you are submitting.

   Input: `(that-function 'b '(a b c))`

   ```
   (define that-function
     (lambda (a s)
       (cond
         [(empty? s) '()]
         [(equal? (first s) a) (cons a s)]
         [else (cons (first s) (that-function a (rest s)))])))
   ```