CSE 341 — More Scheme Discussion Questions

1. Using `map` and `lambda`, define a function `averages` that accepts two lists of numbers, and returns a list of the average of each pair. For example:
   ```
   (averages '(1 2 3) '(11 12 13))   =>  (6 7 8)
   ```

2. What is the value of `x` and `y` after evaluating the following expressions?

   (a)
   ```
   (define x '(1 2 3 4))
   (define y x)
   (set-car! (cdr x) 100)
   ```

   (b)
   ```
   (define x '(1 2 3 4))
   (define y x)
   (set! x '(100 200))
   ```

3. Aloysius Q. Hacker, 341 student, is puzzled by the following code.

   ```
   (define incr)
   (define get)

   (let ((n 0))
      (set! incr (lambda (i) (set! n (+ n i))))
      (set! get (lambda () n)))
   ```

   Aloysius is unsure how the functions `incr` and `get` can possibly work . . . if `n` is in a stack frame, why do `incr` and `get` still work correctly even though we're done with evaluating the `let`?

   Is there something special about `let` at the top level? So Aloysius tries an experiment:

   ```
   (define newincr)
   (define newget)

   (define (test k)
     (let ((n 0))
        (set! newincr (lambda (i) (set! n (+ n i k))))
        (set! newget (lambda () n))))
   ```

   Then he evaluates `(test 100)`. This time the `let` is embedded in a function, and so (Aloysius reasons) certainly its stack frame will go away when `test` returns.

   What is the result when Aloysius evaluates each of the following expressions in turn?

   ```
   (newget)
   (newincr 10)
   (newget)
   ```

   Explain (or at least make some reasonable hypotheses).

4. Define a tail-recursive version of "map" for 1-argument functions. (Avoid side effects if possible, but use them if necessary.)