1. Write procedure \texttt{(duplicate a lat)} that adds a second instance of atom \texttt{a} every time \texttt{a} is found in the flat list \texttt{lat}. For example, \texttt{(duplicate 'b '(a b a c a b a))} returns \texttt{(a b b a c a b b a)}

\begin{verbatim}
(define duplicate
  (lambda (a lat)
    (cond
      [(null? lat) null]
      [(eq? a (car lat)) (cons a
        (cons a
          (duplicate a (cdr lat)))])
      [else (cons (car lat) (duplicate a (cdr lat)))]))
\end{verbatim}
2. Write procedure (removeDuplicates lat). As usual lat is a flat list of atoms. For each run of identical entries in lat, such as the 3s in (1 2 3 3 3 2 1), this procedure will remove all but one of those entries. So (removeDuplicates '(1 3 3 3 3 4 2 2 1)) returns (1 3 4 2 1), and (removeDuplicates '(1 2 1 2 3)) returns (1 2 1 2 3)

```
(define removeDuplicates
  (lambda (lat)
    (cond
      [(null? lat) null]
      [(null? (cdr lat)) lat]
      [(eq? (car lat) (cadr lat))
        (removeDuplicates (cdr lat))]
      [else (cons (car lat)
                   (removeDuplicates (cdr lat))))]))
```
3. Use foldr or foldl to write (count a lat) which returns the number of instances of atom a in lat, a flat list of atoms. For example, (count 3 '(1 2 3 2 3 2 3 4 3 3)) returns 5

(function)
(define count
  (lambda (a lat)
    (foldr (lambda (x y) (if (eq? x a) (+ y 1) y))
      0
      lat)))
4. Consider the following function:

\[
\text{(define } B \\text{(lambda (L) (cond [(null? L) null] [(atom? L) (if (eq? L 'bob) (list L) null)] [else (apply append (map B L)))]))}
\]

a) What is \((B \ '(1 2 3 \text{bob}))\)?

Answer: \((\text{bob})\)

b) What is \((B \ '[(1 3 \text{bob} (4)) \ (5 ((6)) \ (7 (8 \text{bob} 9)))])\)?

Answer: \((\text{bob bob})\)
5. What does the following expression evaluate to in the top-level environment? Be very explicit:

\[
\begin{align*}
& \text{(let ( [a 5] [b 3] )} \\
& \quad \text{(lambda (x y) (* a (+ b (* x y))))}
\end{align*}
\]

When this is evaluated in the top-level environment, a new environment is created that extends the top-level environment with bindings of a to 5 and b to 4. Call this new environment \( E' \). The let expression then returns the value of its body in \( E' \). Since the body is a lambda expression, it evaluates to a closure with three parts: the parameter list \((x y)\), the lambda’s body \((* a (+ b (* x y)))\) as an unevaluated expression, and the environment \( E' \) (which has the bindings for a and b). This closure is the value of the full let-expression.
6.  
   a. Write (last lat) which returns the last atom in the flat list lat. For example, (last '(a b c d)) returns d. None of the entries of lat will be null.

   (define last
      (lambda (lat)
         (cond
          [(null? lat) null]
          [(null? (cdr lat)) (car lat)]
          [else (last (cdr lat))])))

   b. Write (last* L) which returns the last non-null atom in the general list L. For example, (last* '(a (b (c)) (d (e f)) (())) ) should return f.

   (define last*
      (lambda (L)
         (cond
          [(null? L) null]
          [(atom? L) L]
          [(let ([A (last* (car L))]
                  [B (last* (cdr L))]
                  (if (null? B) A B)])))


7. Write function `(separateNums L)` that returns a list of two flat lists: one containing the numbers of L, the other containing any other atoms of L. Both lists should have their atoms in the same order as L. For example,

(separateNums ‘(a b 3 c 4 2 d 5)) returns ( (3 4 2 5) (a b c d) ) while

(separateNums ‘((a b (c (d 1 2) 3)) ((e 4 5 (f)))) ) returns ( (1 2 3 4 5) (a b c d e f))

```
(define separateNums
  (lambda (L)
    (cond
      [[null? L) (list (list null) (list null))]
      [[atom? L) (if (number? L)
                  (list (list L) null)
                  (list null (list L)))]
      [else (let ([A (map separateNums L)])
                  (list (apply append (map car A))
                        (apply append (map cadr A))))]))
```