Assignment 6

Programming Languages

1. [20 pts, Scheme]

   Write a function count so that `(count a lat)` counts how often the atom `a` appears in the list of atoms `lat`.

```
> (count 'a '(d a t a))
2
> (count 's '(m i s s i s s i p p i))
4
```

2. [20 pts, Scheme]

   Write a function `(count-all lat)` that produces a list pairing every individual atom in the list of atoms `lat` with a count of how often the atom appears in `lat`.

```
> (count-all '(m i s s i s s i p p i))
((m 1) (i 4) (s 4) (p 2))
> (count-all '(to be or not to be that is the question))
((to 2) (be 2) (or 1) (not 1) (that 1) (is 1) (the 1) (question 1))
```

   The order of the items in the output list does not matter. Make sure to include the code if you use any auxiliary functions (helper functions). Note that this function is computing a histogram.

3. [20 pts, Scheme] Consider the following code:

```
(define mystery
  (lambda (x)
    (cond
     ((null? x) (list x))
     (else (append
            (mystery (cdr x))
            (map (lambda (y)
                   (cons (car x)
                         y))
                 (mystery
                  (cdr x)))))))))
```

   (a.) What is the value of `(mystery '(a b c))`?

   (b.) Suggest a better name for mystery that better conveys its purpose.

   (c.) What is the value of `(length (mystery '(a b c d e f g h i j)))`?

4. [50 pts, Scheme] (including 10 pts bonus)

   Recall that in lexical binding (or scoping) a variable reference is associated with the nearest lexically enclosing binding of the variable. Dynamic binding (or scoping) resolves a variable reference in the body of a function in the environment at the point of call (extended with bindings of actuals to formals). Below are expressions in a Scheme-like syntax. Evaluate each using obvious meanings in both a lexical and a dynamic setting. If something happens to prevent straightforward evaluation, explain the "error" succinctly. You may assume that each of these is to be evaluated in a pristine initial environment: no previous definitions have been made.

   For each of the code pieces below write down Lexical and Dynamic evaluations of the expressions.

   (a.)

```
(let ((x 22))
  (let ((f (lambda (y) x)))
    (let ((x 30))
      (f 42))))
```

   (b.)

```
(let ((f (lambda (y) x)))
  (let ((x 33))
    (f 420)))
```

   (c.)

```
(let ((f (lambda (x)
           (if (= x 0)
               0
               (+ x (f (1- x)))))))
  (f 10))
```

   (d.)

```
(let ((a 2))
  (let ((f (lambda () a)))
    (let ((g (lambda (x) (f))))
      (let ((a 4))
        (g 1)))))
```

   (e.)

```
(let ((a 2))
  (let ((f (lambda () a)))
    (let ((g (lambda (a) (f))))
      (let ((a 4))
        (g 1)))))
```