1. What would Scheme print? Draw box-and-pointer diagrams to help determine this.

(a) (cons (cons 1 nil) (cons 2 (cons (cons 3 (cons 4 5)) (cons 6 nil))))

(b) (define a 4)
   ((lambda (x y) (+ a)) 1 2)

(c) ((lambda (x y z) (y x)) 2 / 2)

(d) ((lambda (x) (x x)) (lambda (y) 4))

(e) (define boom1 (/ 1 0))

(f) boom1

(g) (define boom2 (lambda () (/ 1 0)))

(h) (boom2)
(i) Why/How are the two “boom” definitions above different?

(j) How can we rewrite boom2 without using the lambda operator?

2. What will Scheme output?
   (a) `(if (/ 1 0) 1 0)`
   (b) `(if 1 1 (/ 1 0))`
   (c) `(if 0 (/ 1 0) 1)`
   (d) `(and 1 #f (/ 1 0))`
   (e) `(and 1 2 3)`
   (f) `(or #f #f 0 #f (/ 1 0))`
   (g) `(or #f #f (/ 1 0) 3 4)`
   (h) `(and (and) (or))`

(i) Given the lines above, what can we say about interpreting `if` expressions and booleans in Scheme?
3. The following line of code does not work. Why? Write the lambda equivalent of the let expressions.

```scheme
(let ((foo 3)
       (bar (+ foo 2)))
    (+ foo bar))
```

2. Scoping

4. What is the difference between dynamic and lexical scoping?

5. What would this print using lexical scoping? What would it print using dynamic scoping?

```python
a = 2
def foo():
    a = 10
    return lambda x: x + a
bar = foo()
bar(10)
```

6. How would you modify an environment diagram to represent dynamic scoping?
7. **Implement** `waldo`. `waldo` **returns** `#t` if the symbol `waldo` is in a list. You may assume that the list passed in is well-formed.

```scheme
scm> (waldo '(1 4 waldo))
#t
scm> (waldo '())
#f
scm> (waldo '(1 4 9))
#f
```

**Extra challenge:** Define `waldo` so that it returns the index of the list where the symbol `waldo` was found (if `waldo` is not in the list, return `#f`).

```scheme
scm> (waldo '(1 4 waldo))
2
scm> (waldo '())
#f
scm> (waldo '(1 4 9))
#f
```
8. (Optional) From CS61A Fall 2017 Discussion 6: The quicksort sorting algorithm is an efficient and commonly used algorithm to order the elements of a list. We choose one element of the list to be the pivot element and partition the remaining elements into two lists: one of elements less than the pivot and one of elements greater than the pivot. We recursively sort the two lists, which gives us a sorted list of all the elements less than the pivot and all the elements greater than the pivot, which we can then combine with the pivot for a completely sorted list.

Implement quicksort in Scheme. Choose the first element of the list as the pivot. You may assume that all elements are distinct. Hint: you may want to use a helper function.

```
scm> (quicksort (list 5 2 4 3 12 7))
(2 3 4 5 7 12)
```