CSE 341 — Racket Discussion Questions Part 3

These questions deal with macros, delayed evaluation, improper lists, and functions with a variable number of arguments.

1. The lecture notes for macros include a definition for `my-or` that works just like the built-in `or` in Racket.

```
(define-syntax my-or
 (syntax-rules ()
   ((my-or) #f)
   ((my-or e1 e2 ...)
    (let ([temp e1])
      (if temp
          temp
          (my-or e2 ...))))))
```

   Given this definition, if we expand `(my-or (= x 2))` we get

```
(let ([temp (= x 2)]) (if temp temp (my-or)))
```

   This would further expand to

```
(let ([temp (= x 2)]) (if temp temp #f))
```

   Modify the rule so it just expands `(my-or (= x 2))` to `(= x 2)` instead. It should still work correctly for `(my-or)`.

2. Suppose we are writing our own version of the `if` special form, called `my-if`. This can't be a normal function in Racket, since we evaluate the arguments. We can write it as a macro, of course. For this mini-exercise, write it as a function that uses delay to delay evaluating some or all of the arguments. Only delay arguments if need be.

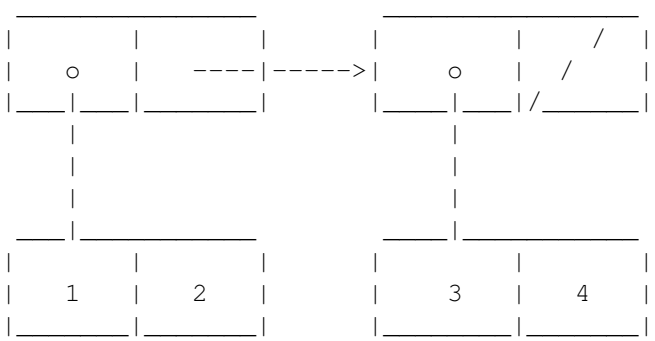
   Now rewrite the following expression using your `my-if` function. (All you need to do is insert the appropriate delays. Use Racket's built-in delay macro.)

```
(if (= 1 1) (+ 2 4) (/ 10 0))
```

3. Draw a box-and-arrow picture for the value of `'(squid . (clam . (octopus . ())))`

4. Draw a box-and-arrow picture for the value of `'(squid . (clam . octopus))`

5. How would you write the following list structure in Scheme?



6. Write a function `my-max` that finds the maximum of its arguments. It needs at least one argument, and can take arbitrarily more. For example

   ```
   (my-max 4 10 2 1)
   ```

   should return 10.

   You can use a helper function if you need to. Bonus points though for a version without a helper function!

   What do these evaluate to?

   ```
   (my-max 3)
   (my-max)
   ```