Kelvin Mei
COSI121B
PS5 Write-up

4.6
Simply followed the format given, where I manipulated the expression into the correct lambda format.

4.7
A let* expression can be rewritten as a series of let expressions by getting a list of all the assignments in let and using a loop to write nested let expressions. It is not sufficient to add (eval (let*->nested-lets exp) env) to eval to implement this because there is no way for the eval loop to know if the exp is let* or just regular let. Therefore, it is necessary to explicitly expand let* in terms of non-derived expressions or just have eval perform let* on all instances of let.

4.13
This unbinds variables, but it only does so for the first environment because removing it from other environments can produce errors.
It goes through the variable environment until it finds the expected variable. If found, it removes that variable and the corresponding value from the environment using set-car! and set-cdr!.

4.15
```
(define (run-forever) (run-forever))

(define (try p)
  (if (halts? p p)
      (run-forever)
      'halted))
```

(try try)
Try runs forever if (halts? p p), Try halts if (not (halts? p p)). So in order for try to halt, (not (halts? p p)) must be true. However, (p p) runs forever if (try try) halts and (p p) halts if (try try) runs forever. (try try) is essentially (p p), so it's saying (try try) runs forever if (try try) halts and vice versa. This creates a contradiction from assuming there exists a method (halts? p p), so the assumption must be wrong.

Using cond to define if:
If you try to ignore the if construct and use cond to define if, it does not work because conds are converted to ifs when they are evaluated. If the if construct does not exist, then conds do not work.