CSSE 304 Day 24

1. Rewrite in CPS form (treat `memq` as substantial, thus we call `memq-cps`)

```
(define intersection
  (lambda (los1 los2)
    (cond [(null? los1) '()]
          [(memq (car los1) los2)
           (cons (car los1)
                 (intersection (cdr los1) los2))]
          [else (intersection (cdr los1) los2)])))
```

2.
```
(define free-vars-cps ; convert to CPS
          (lambda (exp k)
     (cond [(symbol? exp)

                                                ]
           [(eq? (1st exp) 'lambda)



                                                                                ]
           [else



           ])))
```

3.

```
(define union-cps
  (lambda (s1 s2 k)
    (if (null? s1)




)))
```

4.

```
(define remove-cps
  (lambda (element ls k)
    (if (null? ls)









      )))
```

5. *Succeed* and *fail* continuations example: **prod-cps**

```
(define prod-cps ; fill in the continuations at the end
  (lambda (L succeed fail)
    (cond [(null? L) (succeed 1)]
          [(zero? (car L)) (fail)]
          [else (prod-cps (cdr L)
```

```
(define print-list-product
 (lambda (list)
    (prod-cps list
        (lambda (prod)
           (printf "The product is ~s~n" prod))
        (lambda ()
           (printf "zero found, product is 0”)
```

Another similar example, substitute-leftmost (you may want to annotate this:

```
(define substitute-leftmost
  (lambda (new old slist)
    (subst-left-cps
      new
      old
      slist
      (lambda (v) v)    ; changed continuation
      (lambda () slist) ; unchanged continuation
    )))


define subst-left-cps  ; changed and unchanged are continuations
 (lambda (new old slist changed unchanged)
  (let loop ([slist slist] [changed changed] [unchanged unchanged])
   (cond
    [(null? slist) (apply-k unchanged)]
    [(symbol? (car slist))
     (if (eq? (car slist) old)
         (apply-k changed (cons new (cdr slist)))
         (loop (cdr slist)
               (make-k (lambda (substituted-cdr)
                         (apply-k changed
                                  (cons (car slist) substituted-cdr))))
               unchanged))]
      [else ; car is an s-list
       (loop (car slist)
             (make-k (lambda (substituted-car)
                       (apply-k changed (cons substituted-car (cdr slist)))))
             (make-k (lambda ()
                       (loop (cdr slist)
                             (make-k (lambda (substituted-cdr)
                                       (apply-k changed
                                                (cons (car slist)
                                                      substituted-cdr))))
                             unchanged))))])))))
```