CSSE 304   Day 31 Summary

1. Back to writing code in CPS: Now with continuations represented as data structures instead of Scheme procedures.

Review of environment representations:

1. An environment is represented by a Scheme procedure

2. An environment is represented by a datatype.

Continuations as a datatype

- Continuation ADT
    a. Interface
        i. Various constructors (in our new implementation, will be created by `define-datatype`)
        ii. `(apply-continuation k val)` (where `k` is a representation of a continuation)
    b. We'll go through a process that is similar to what we did with environments
    c. First represent a continuation by a Scheme procedure (most of the work is in the constructors)
    d. Then define a continuation datatype (most of the work is in `apply-k`)
        i. Continuation constructors are treated as primitive (in the CPS sense)
- Example of the transformation to data-structure continuations (live coding).

```
(define read-flatten-print
  (lambda ()
    (display "enter slist to flatten: ")
    (let ([slist (read)])
      (unless (eq? slist 'exit)
        (flatten-cps slist
                     (make-k (lambda (val)
                               (pretty-print val)
                               (read-flatten-print)))))))))

(define append-cps
  (lambda (L1 L2 k)
    (if (null? L1)
        (apply-k k L2)
        (append-cps (cdr L1)
                    L2
                    (make-k
                     (lambda (appended-cdr)
                          (apply-k k (cons (car L1)
                                      appended-cdr))))))))


(define flatten-cps
  (lambda (ls k)
    (if (null? ls)
        (apply-k k ls)
        (flatten-cps (cdr ls)
          (make-k (lambda (v)
                   (if (list? (car ls))
                    (flatten-cps (car ls)
                      (make-k (lambda (u) (append-cps u v k))))
                    (apply-k k (cons (car ls) v)))))))))


(define apply-k (lambda (k . vals) (apply k vals)))

(define make-k (lambda (v) v))
```