```
#lang racket

; LECTURE 3

; PRACTICE - Pairs

(define u (cons 2 3))
(define w (cons 5 6))
(define x (cons u w))
(define y (cons w x))
(define z (cons 3 y))

; Draw box-and-pointer diagrams for u, w, x, y, z

; Evaluate:
; [#1]   (car y)
; [#2]   (car (car y))
; [#3]   (cdr (car (cdr (cdr z))))
; [#4]   (+ (cdr (car y)) (cdr (car (cdr z))))

; PRACTICE - Lists

; Draw box-and-pointer diagrams for the following lists:

; (define a '(1 2 3))
; (define b '((1 2) 3))
; (define c '((1 2) (3)))
; (define d '(1 2 '()))

; Evaluate:
; [#1]   (car a)
; [#2]   (cdr a)
; [#3]   (cdr (cdr a))
; [#4]   (cdr (cdr (cdr a)))

; [#5]   (car b)
; [#6]   (cdr b)

; [#7]   (car c)
; [#8]   (cdr c)

; [#9]   (cdr (car (cdr '(((1) 3) (4 (5 6))))))

; Evaluate and draw box-and-pointer diagrams:

; (cons '((1 a) (2 b)) '(3 c))

; (list '((1 a) (2 b)) '(3 c))

; (append '((1 a) (2 b)) '(3 c))

; LIST FUNCTIONS

; process a list:
(define (sum-list lst)
        (if (null? lst)
                0
                (+ (car lst) (sum-list (cdr lst)))))

; produce a list:
(define (countdown num)
        (if (= num 0)
                '()
                (cons num (countdown (- num 1)))))
```