```
#lang racket
; Programming Languages, Lecture 2

; Functions and recursion
(define (add1 x)
  (+ 1 x))

(define (abs x)
  (if (< x 0)
      (- x)
      x))

; only correct for y >= 0
(define (pow x y)
  (if (= y 0)
      1
      (* x (pow x (- y 1)))))

(define (cube x)
  (pow x 3))

(define sixtyfour (cube 4))

(define fortytwo (+ (pow 2 4) (pow 4 2) (cube 2) 2))

; pairs
(define (swap pair)
  (cons (cdr pair) (car pair)))

(define (sum-two-pairs pair1 pair2)
  (+ (car pair1) (cdr pair1) (car pair2) (cdr pair2)))

(define (div-mod n1 n2)
  (cons (quotient n1 n2) (remainder n1 n2)))
; returning more than one value is a pain in Java

(define (sort-pair pair)
  (if (> (car pair) (cdr pair))
      pair
      (swap pair)))

; lists

(define (sum-list lst)
  (if (null? lst)
      0
      (+ (car lst) (sum-list (cdr lst)))))


(define (countdown num)
  (if (= num 0)
      '()
      (cons num (countdown (- num 1)))))
```

```
; lists of pairs

; the next four functions assume that
; the argument lst is a list of pairs of numbers
(define (sum-pair-list lst)
  (if (null? lst)
      0
      (+ (car (car lst)) (cdr (car lst)) (sum-pair-list (cdr lst)))))

(define (firsts lst)
  (if (null? lst)
      '()
      (cons (car (car lst)) (firsts (cdr lst)))))

(define (seconds lst)
  (if (null? lst)
      '()
      (cons (cdr (car lst)) (seconds (cdr lst)))))

(define (sum-pair-list2 lst)
  (+ (sum-list (firsts lst)) (sum-list (seconds lst))))
```