Clicker Questions
September 25
I want to write a function `sum2dVectors` that adds the first elements of a bunch of pairs and then adds the second elements, so

```
(sum2dVectors '(3 4) '(1 2) '(2 3)) returns '(6 9).
```

The start is easy:

```
(define sum2dVectors (lambda (pairs)
    (cond
      [(null? (cdr pairs)) (car pairs)]
      [else (let ([a (car (car pairs))]
                  [b (cadr (car pairs))]
                  [v ; THIS SHOULD BE THE RESULT OF
                   sum2dVectors RECURSING ON ALL BUT ITS
                   FIRST ARGUMENT
                   (list (+ a (car v)) (+ b (cadr v))))])))
```
In the definition

```
(define sum2dVectors (lambda pairs ...
```

how does `sum2dVectors` recurse on all but its first argument

A. `(sum2dVectors (cdr (list pairs)))`
B. `(sum2dVectors (cdr pairs))`
C. `(apply sum2dVectors (cdr pairs))`
D. `(apply (sum2dVectors (cdr pairs)))`
Answer C: (apply sum2dVectors (cdr pairs))
How would you write `sum2dVectors` with `foldr`??

A. I wouldn't.

B. `(define sum2dVectors (lambda pairs
   (foldr (lambda (x y) (list (+ (car x) (car y)) (+ (cadr x) (cadr y))))
     (list 0 0) pairs)))`

C. `(define sum2dVectors (lambda pairs
   (foldr (lambda (x y) (+ x y)) (list 0 0) pairs)))`

D. `(define sum2dVectors (lambda pairs
     (foldr (lambda (x y) (apply + x y)) (list 0 0) pairs)))`
Answer B:
A. (define sum2dVectors (lambda pairs
    (foldr (lambda (x y) (list (+ (car x) (car y)) (+ (cadr x) (cadr y)))))
    (list 0 0) pairs)))
How would you write `sum2dVectors` with map and apply??

A. I wouldn't.

B. `(define sum2dVectors (lambda pairs
    (list (apply + (map car pairs)) (apply + (map cadr pairs))))`

C. `(define sum2dVectors (lambda pairs
    (apply + (map list pairs))`

D. `(define sum2dVectors (lambda pairs
    (apply list (map + pairs))`
Answer B:

\[
\text{(define sum2dVectors (lambda pairs)}
\text{ (list (apply + (map car pairs)) (apply + (map cadr pairs)))}
\text{)}
\]
How would you write \((\text{count a lat})\), which counts the number of times atom \(a\) appears in \(\text{lat}\), with \text{fold}?

A. \((\text{define count (lambda (a lat)\\n\qquad (foldl (lambda (x y) (if (eq? x a) (+ y 1) y)) 0 lat)))})\)

B. \((\text{define count (lambda (a lat)\\n\qquad (foldr (lambda (x y) (if (eq? x a) (+ y 1) x)) 0 lat)))})\)

C. \((\text{define count (lambda (a lat)\\n\qquad (foldr (lambda (x y) (if (eq? x a) x y)) 0 lat)))})\)

D. \((\text{define count (lambda (a lat)\\n\qquad (foldr (lambda (x y) (if (eq? x y) (+ y 1) y)) a lat)))})\)
A. Answer A:

(define count (lambda (a lat)
  (foldl (lambda (x y) (if (eq? x a) (+ y 1) y)) 0 lat)))
Which of the following could you use to represent the fraction 3/4?

a) (list 3 4) ==> '(3  4)
b) (cons 3 4) ==> '(3.4)
c) (lambda (t)
   (cond
     [(eq? t 'numerator) 3]
     [(eq? t 'denominator) 4]))
d) (/ 3 4)
e) 0.75

A. None of these                   C. All of these but (c)
B. All of these                   D. All of these but (e)
Answer D: All of these except 0.75. There is a fundamental difference between fractions and floating-point numbers. Any of the rest could be used to represent fractions.