Clicker Questions for December 3
Your last lab deals with streams. What is the difference between
(define bobs (cons 'bob bobs))
and
(define bob$ (cons$ 'bob bob$))

A. bobs is a valid list and bob$ is a valid stream
B. bobs is an infinite recursion and bob$ is a valid stream
C. bobs is a valid list and bob$ isn't worth $1
D. No difference -- both are infinite recursions
(define bobs (cons 'bob bobs))
and
(define bob$ (cons$ 'bob bob$))

Answer  B: bobs is an infinite recursion and bob$ is a perfectly valid and delightful stream.
I want to define the stream Evens$ of even integers: 0 2 4 6 etc. What does this calculation tell you:

\[
\begin{array}{cccccccc}
0 & 2 & 4 & 6 & 8 & 10 & 12 & 14 & 16 \\
+ 2 & & & & & & & & \\
\hline
2 & 4 & 6 & 8 & 10 & 12 & 14 & 16 & 18
\end{array}
\]

A. (define Evens$ (cons$ 0 (+$ 2 Evens$)))
B. (define Evens$ (cons$ 0 (cdr$ Evens$)))
C. (define Evens$ (cons$ 0 (map$ (lambda (x) (+ 2 x)) Evens$)))
D. It tells me that streams are very weird.
Answer C:
(define Evens$ (cons$ 0 (map$ (lambda (x) (+ 2 x)) Evens$)))
This one is a hint for one of the lab exercises. What is an easy way to make the stream of alternating 1 and -1: Alts$ = 1 -1 1 -1 1 -1 ....?

A. If you square every element you get the stream of 1s:
   One$ = 1 1 1 ...
B. If you add Alts$ to (cdr$ Alts$) you get the stream of 0s:
   Zero$ = 0 0 0 0 ... = (cons$ 0 Zero$)
C. If you multiply Alts$ by -1 and cons$ 1 onto the front you get Alts$ back.
D. (define Alts$ (cons$ 1 (cons$ -1 Alts$)))
\[ \text{Answer D: } \text{(define Alts\$ (cons\$ 1 (cons\$ -1 Alts\$)))} \]
Some of you didn't much like continuation-passing style, but it is coming back to haunt you. The rules are
A. Every recursive function gets an extra parameter k, which is the continuation for its call.
B. The continuation k is applied to every answer.
C. All recursive calls must be tail recursive.
What is the cps version of the function that finds the length of a list?

(see the next slide)
A. (define length-k (lambda (L k)
    (if (null? L) (k 0)
        (length-k (cdr L) (lambda (t) (k (+ 1 t)))))))
B. (define length-k (lambda (L k)
    (if (null? L) 0
        (length-k (cdr L) (lambda (t) (+ 1 t))))))
C. (define length-k (lambda (L k)
    (if (null? L) (k 0)
        (+ 1 (length-k (cdr L) k)))))
D. (define length-k (lambda (L k)
    (if (null? L) (k 0)
        (k (length-k (cdr L) (lambda (t) (+ 1 t)))))))
Answer A:
(define length-k (lambda (L k)
    (if (null? L) (k 0)
        (length-k (cdr L) (lambda (t) (k (+ 1 t)))))))
Last question: What is the CPS version of the function (CountAtoms L) that returns the number of atoms in the general list L?
(define CountAtoms-k (lambda (L k)
    (cond
        [(null? L) (k 0)]
        [(atom? (car L))
            (CountAtoms-k (cdr L) (lambda (t) (k (+ 1 t))))]
        [else (CountAtoms-k (car L)
            (lambda (s) (CountAtoms-k (cdr L)
                (lambda (t) (k (+ s t))))))])))