Clicker Questions for December 10
Remember the expression \((\text{call/cc } (\lambda (k) \text{ body}))\).

The lambda expression is called with the continuation of the call/cc expression as the value for \(k\). This continuation is a function of one argument. If \(k\) is never called in the body this returns a value like any other lambda expression. If \(k\) is called within the body control returns to the top level, with the value of \(k\) applied to its argument.
For example, consider \((+ 1 (\text{call/cc (lambda (k) (k 5))))))\n
The continuation of the call/cc expression is \((\text{lambda (x) (+ 1 x)})\). When we invoke \((k 5)\) we return to the top level with value 6.
What about these two lines?

(define foo 0)
(+ 1 (call/cc (lambda (k) (begin (set! foo k) 23)))))

A. The second line evaluates to 23.
B. The second line evaluates to 24
C. The second line evaluates to the "+ 1" continuation.
D. The second line evaluates to 0.
(define foo 0)
(+ 1 (call/cc (lambda (k) (begin (set! foo k) 23)))))

Answer: The second line returns 24.
(define foo 0)
(+ 1 (call/cc (lambda (k) (begin (set! foo k) 23)))))

OK, so the second line evaluates to 24. It also does a set! to foo. After these two lines are executed, what is foo?

A. foo is still 0.
B. foo is 24.
C. foo is the "+ 1" function. So (* 3 (foo 4)) is 15.
D. foo is the "+ 1" continuation. So (* 3 (foo 4)) is 5.
(define foo 0)
(+ 1 (call/cc (lambda (k) (begin (set! foo k) 23))))

Answer D: foo is the "+ 1" continuation. So (* 3 (foo 4)) is 5.
(define C 0)
(define grab (lambda (vec)
    (call/cc (lambda (k)
        (cond
            [(null? vec) 1]
            [(= 0 (car vec)) (begin (set! C k) 0)]
            [else (* (car vec) (grab (cdr vec)))]))))

What is (grab '(3 4 5))? 
A. 1 
B. 3*4*5 = 60 
C. 3 
D. 0
(define C 0)
(define grab (lambda (vec)
  (call/cc (lambda (k)
    (cond
      [(null? vec) 1]
      [(= 0 (car vec)) (begin (set! C k) 0)]
      [else (* (car vec) (grab (cdr vec)))]]))))

What is (grab '(3 4 5))? 

Answer B: 3*4*5 = 60
What is (grab '(3 4 0 5))?

A. $3 \times 4 \times 0 = 0$
B. $3 \times 4 = 12$
C. $3 \times 4 \times 0 \times 5 = 0$
D. Some weird continuation
(define C 0)
(define grab (lambda (vec)
    (call/cc (lambda (k)
        (cond
          [(null? vec) 1]
          [(= 0 (car vec)) (begin (set! C k) 0)]
          [else (* (car vec) (grab (cdr vec)))]))))

What is (grab '(3 4 0 5))?

Answer A: 3*4*0 = 0
Finally, what does (grab '(3 4 0 5)) do to C?
A. It sets C to 0.
B. It sets C to 3*4 = 12.
C. It sets C to the "multiply times 3 then by 4 continuation."
D. It turns C into the spawn of the devil .....

(define C 0)
(define grab (lambda (vec)
    (call/cc (lambda (k)
        (cond
            [(null? vec) 1]
            [ (= 0 (car vec)) (begin (set! C k) 0)]
            [else (* (car vec) (grab (cdr vec))))])))))
(define C 0)
(define grab (lambda (vec)
  (call/cc (lambda (k)
    (cond
      [(null? vec) 1]
      [ (= 0 (car vec)) (begin (set! C k) 0)]
      [else (* (car vec) (grab (cdr vec)))])])))

What does (grab '(3 4 0 5)) do to C?

Answer C: It sets C to the "multiply times 3 then by 4 continuation."
After we execute these two defines, we know A is going to get the value of the `call/cc` expression and then B will get the same value as A. But what is that value?

A. B is the identity function. (B k) is k, so (B 23) is 23.
B. B tries to redefine B, so (B 23) is an error.
C. B tries to redefine A, so (B 23) is an error.
D. B does redefine A, so (B 23) sets A to 23.
(define A (call/cc (lambda (k) k)))
(define B A)

What does B do?

Answer: B redefines A, so (B 23) sets A to 23.
(define bar 0)
(define foo (+ 1 (call/cc (lambda (k) (begin (set! bar k) 0)))))

These are similar to other questions we have looked at. First, what is the value of foo?

A. 0
B. 1
C. A continuation that sets bar to 1
D. A continuation that sets foo to 1
(define bar 0)
(define foo (+ 1 (call/cc (lambda (k) (begin (set! bar k) 0))))))

What is the value of foo?

Answer B: 1  The body of the (lambda (k)...)) and the call/cc both return 0.
(define bar 0)
(define foo (+ 1 (call/cc (lambda (k) (begin (set! bar k) 0))))))

So these definitions make foo into 1. But what do they do to bar?

A. bar is set to 0.
B. bar is set to 1.
C. bar is set to the "+ 1" continuation, so (bar 23) returns 24.
D. bar is set to the "take a value, add 1 to it, and set foo to be that value" continuation, so (bar 23) sets foo to 24.