Clicker Qs for November 14
What will this expression evaluate to in standard Scheme?

(let ([a 5])
  (let ([f (lambda (x) (* b (+ x a)))]))
  (let ([a 10] [b 2])
    (f a))))

A. It is an error.
B. 40 (i.e., (* 2 (+ 10 10)))
C. 30 (i.e., (* 2 (+ 10 5)))
D. 20 (i.e., (* 2 (+ 5 5)))
(let ([a 5])
  (let ([f (lambda (x) (* b (+ x a)))]))
  (let ([a 10] [b 2])
    (f a))))

Answer A: This is an error. The closure environment for f only has one binding: [a 5].
So what value does this have in standard Scheme?

(let ([a 5] [b 3])
  (let ([f (lambda (x) (* b (+ x a)))]))
  (let ([a 10] [b 2])
    (f a))))

A. It is still an error
B. 30  (* 3 (+ 5 5))
C. 45  (* 3 (+ 10 5))
D. 60  (* 3 (+ 10 10))
(let ([a 5] [b 3])
  (let ([f (lambda (x) (* b (+ x a)))]))
  (let ([a 10] [b 2])
    (f a))))

Answer C: 45 (* 3 (+ 10 5))
Remember that with dynamic scoping free variables in a function body get their values from the most recent environment. What value would this expression have if Scheme used dynamic scoping?

(let ([a 5][b 3])
  (let ([f (lambda (x) (* b (+ x a)))]))
    (let ([a 10][b 2])
      (f a))))

A. It would still be an error
B. 40  (* 2 (+ 10 10))
C. 45  (* 3 (+ 10 5))
D. 60  (* 3 (+ 10 10))
(let ([a 5][b 3])
  (let ([f (lambda (x) (* b (+ x a)))]))
  (let ([a 10][b 2])
    (f a)))

Answer B: 40  (* 2 (+ 10 10))
How could we change MiniScheme to use dynamic binding?

A. Evaluate the arguments to a call in their current environment, then pass these values to the function.
B. Evaluate the arguments to a call in the closure environment after they are passed to the function.
C. Extend the closure environment with bindings of the parameters to the values of the arguments, and use this extended environment to evaluate the function body.
D. Extend the current environment with bindings of the parameters to the values of the arguments, and use this extended environment to evaluate the function body.
Answer D: Extend the current environment with bindings of the parameters to the values of the arguments, and use this extended environment to evaluate the function body.

Note that answer C:

Extend the closure environment with bindings of the parameters to the values of the arguments, and use this extended environment to evaluate the function body" is what we currently do in MiniScheme. This creates static (lexical) binding.
This expression evaluates to 0 in Scheme because Scheme uses call-by-value. What would it evaluate to if Scheme used call-by-reference?

\[
\begin{align*}
&\text{(let ([a 0])} \\
&\quad \text{(let ([f (lambda (x) (begin (set! x (+ x 1)) 23)])} \\
&\quad \quad \text{(begin} \\
&\quad \quad \quad \text{(f a) \\
&\quad \quad \quad \quad a)))))
\end{align*}
\]

A. It is an error
B. 0
C. 1
D. 23
(let ([a 0])
  (let ([f (lambda (x) (begin (set! x (+ x 1)) 23))])
    (begin
      (f a)
      a))))

Answer C: 1
Last question: Remember that with call-by-name the text of the argument is substituted for the parameter of the function. This expression evaluates to 1 in standard Scheme. What would it evaluate to if Scheme used call-by-name?

(let ([x 0])
  (let ([f (lambda (y)
              (begin
                (set! x y)
                y))])
    (f (+ x 1))))

A. 0
B. 1
C. 2
D. call-by-name is stupid ....
Answer C: 2