Clicker Questions
November 1
The first step in Lab 7 is to handle lambda expressions. What will (lambda (x y) body) **parse** to?

A. A closure (a new structure that is a triple with the parameter list, the body and the current environment).

B. A new tree structure that holds the parameter list and the body.

C. A Scheme lambda expression

D. A let expression, because lambdas and lets are equivalent and we have already implemented lets.
Answer B: A new tree structure that hold the parameter list and the body.

Note that we don't have any particular environment when we are parsing, so we couldn't build a closure even if we wanted to.
So let's say that (lambda (x y) body) parses to
(lambda-exp (x y) (parse body) )
Call this tree. What is (eval-exp tree env)?

A. We can't evaluate it until it is called.

B. (eval-exp (body tree) env)

C. (eval-exp (body tree) (extended-env (params tree) (env tree))

D. (closure (params tree) (body tree) env) where closure is a new structure.
Answer D: (closure (params tree) (body tree) env) where closure is a new structure.
So go step by step through the way the following is evaluated:

(let ([foo (lambda (x y) (+ x y))]))
  (foo (+ 1 2) 4))
What does this evaluate to?

(begin
  (set! V 3)
  (+ V 5))

A. 3
B. 5
C. 8
D. An error
Answer D: An error. You can't set a variable that doesn't exist.
What does this evaluate to?
(let ([V 25])
  (begin
    (set! V 3)
    (+ V 5)))
A. 3
B. 30
C. 8
D. An error
It is pretty easy to guess that we will parse (set! symbol expression) into a new structure (assign-exp symbol (parse expression)). If tree is this structure, how will we do (eval-exp tree env)?

A. (set! (lookup-env (symbol tree) (eval-exp (value tree) env))) where symbol and value are getters for the two fields of assign-exp.
B. (extended-env (list (symbol tree)) (list (eval-exp (value tree) env) env) (In other words, extend the environment with a new binding)
C. Recursively rebuild the environment so that the binding of the symbol is changed to (eval-exp (value tree) env)
D. We can\'t, at least we can\'t using anything we have seen so far.
Answer D: We can't do this using what we have seen so far. The most tempting of those alternatives is B, where we extend the environment with a new binding for the symbol, but that won't handle this, which should evaluate to 53:

```
(let ([V 5])
  (let ([f (lambda (x) (+ x V))])
    (begin
      (set! V 50)
      (f 3))))
```