Review from Tuesday

Scheme, and probably all of the languages you know, uses *lexical* or *static scoping*. The value of a free variable in a lambda expression comes form the bindings that were in place when the function was defined or created.

With *dynamic scoping* free variables get their values from the environment in place when the function is called.

Consider this expression:

```
(let ([x 10])
     (let([f (lambda (y) (* x y))])
          (let ([x 100])
               (f 3))))
```

What does

```
(let ([x 10])
     (let([f (lambda (y) (* x y))])
            (let ([x 100])
                   (f 3))))
```

evaluate to under dynamic scoping?

A. 300

B. 30

C. 1000

D. An error

What does

```
(let ([x 10])
     (let([f (lambda (y) (* x y))])
          (let ([x 100])
               (f 3))))
```

evaluate to under dynamic scoping?

Answer A: 300

What does

```
(let ([x 10])
     (let([f (lambda (y) (* x y))])
            (let ([x 100])
                   (f 3))))
```

evaluate to under static scoping?

A. 300

B. 30

C. 1000

D. An error

What does

```
(let ([x 10])
     (let([f (lambda (y) (* x y))])
            (let ([x 100])
                   (f 3))))
```

evaluate to under static scoping?

Answer B:30

What about?

```
(let ([x 10])
       (let([f (lambda (x)
                           (lambda (y) (* x y)))])
              (let ([x 100])
                     (let ([g (f 5)])
                           (g 3)))))
```

That evaluates to 15 under either scoping mechanism.