Scoping and Binding of Variables

- A variable can appear as a *declaration* or a *reference*.

Declaration:

```
int f(int x, int y) {

   int a;

(lambda (x y)

  (let((a ...
```

Scoping and Binding of Variables

- Reference:

```
(lambda (x y)
  (let (
        (a 4)
       )
    (+ a (* x y))))
```

Scoping and Binding of Variables

- A variable reference is said to be *bound* by a declaration:

```
(lambda (x y)
  (let (
        (a 4)
       )
    (+ a (* x y))))
```

Scoping and Binding of Variables

- Binding takes place within a *scope*.
- Scope of x, y:

```
(lambda (x y)
        (let (
              (a 4)
             )
         (+ a (* x y))))
```

Scoping and Binding of Variables

- Scope of a:

```
(lambda (x y)
  (let (
        (a 4)
       )
    (+ a (* x y))))
```

Scoping and Binding of Variables

- Scope in Scheme (and most languages) is *static*.
- Means we can determine scope by looking at code.
- But can get tricky with nesting!

```
(let ((a 3))
     (let ((a (+ a a)))
       a))
```

Free and Bound Variables

- Variable *x*, expression *E*

  x occurs free in *E* iff there is some use of *x* in *E* that is not bound by any declaration of *x* in *E*

  x occurs bound in *E* iff there is some use of *x* in *E* that is bound by any declaration of *x* in *E*

E = `(lambda (a) (* a b))`

Free and Bound Variables in English!

John hates him. // must be free

John hates himself. // must be bound

John said that Bill hates him. // free or bound

* *Himself hates John.* // reference before declaration