→ Which bindings we can access → Scope?

explicit bindings:

```
val x = 1
var x = 2
  ⋮
x = 3
```

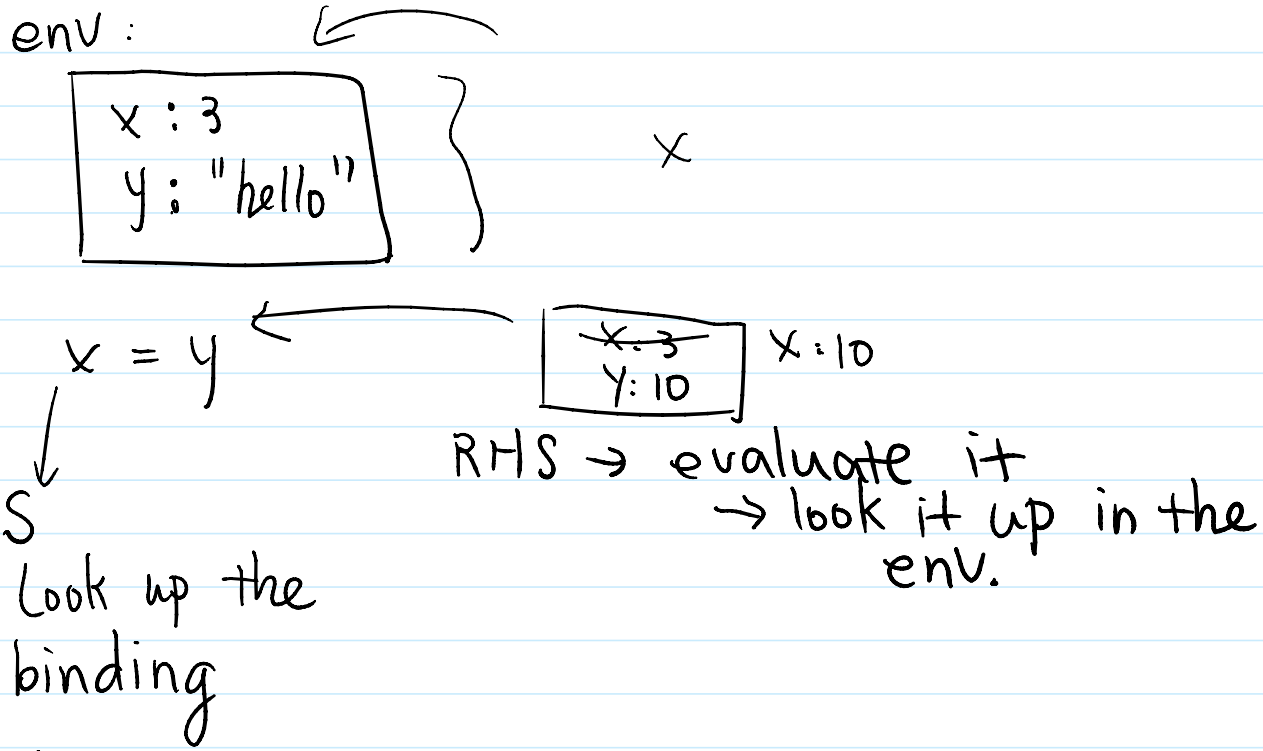
implicit/indirect:

```
def foo (a:Int) = {···}

foo (3)
```

* assumption: a : 3

Sets of bindings → environment



Lots of choices:

```
val y = 2
def foo(x:Int) = {
  x + y
}
foo(3)
```

```
foo(3)
val y = 2
def foo(x:Int) = {
  x + y
}
```

In Scala, this can work!

```
val y = z
val z = 2
```

In scala, it will complain.

Scope → area in which a set of bindings is active.

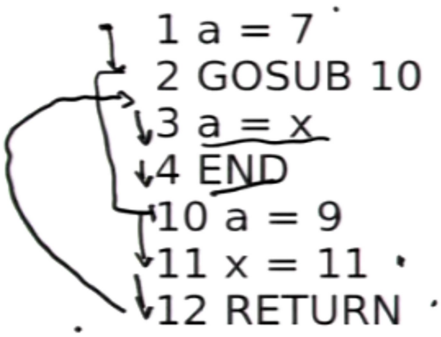
2 main 'styles' of scoping

→ Static Scoping
  → modern languges : Java, Scala
→ Dynamic Scoping
  → older languges : early LISP

Static Scope

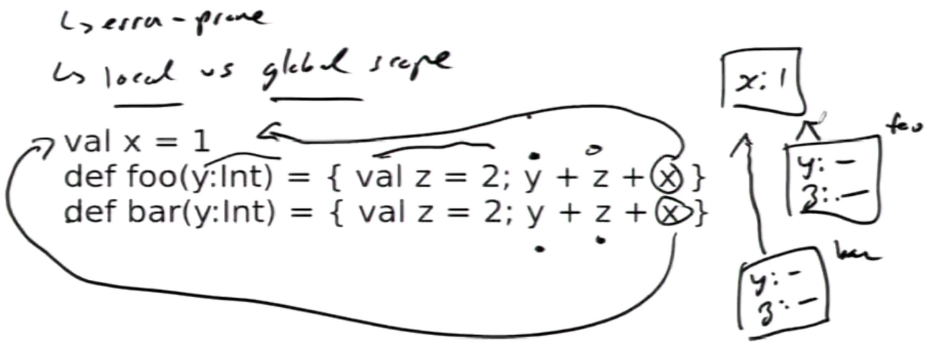
→ lexicod scoping
→ find which vars you can access by looking it in the code. (no dynamic/runtime info needed)

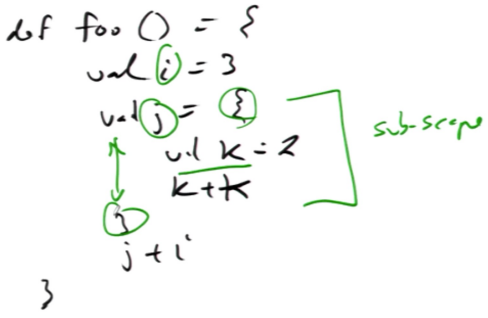
Simplest model → early Basic
  → one scope
    → no procs / funcs

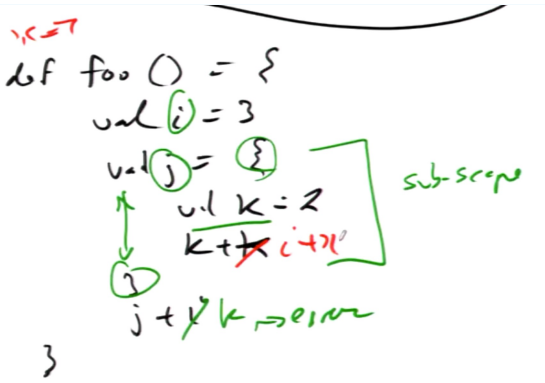


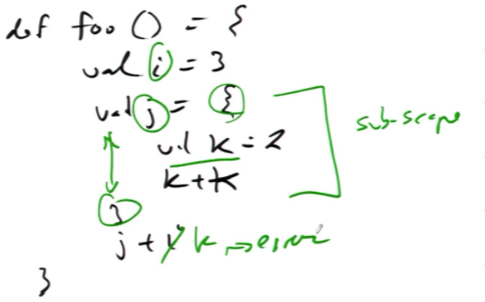
a: 7 9
x: 11

→ error-prone
→ local vs. global scope









First, my scope → parent's scope → ...

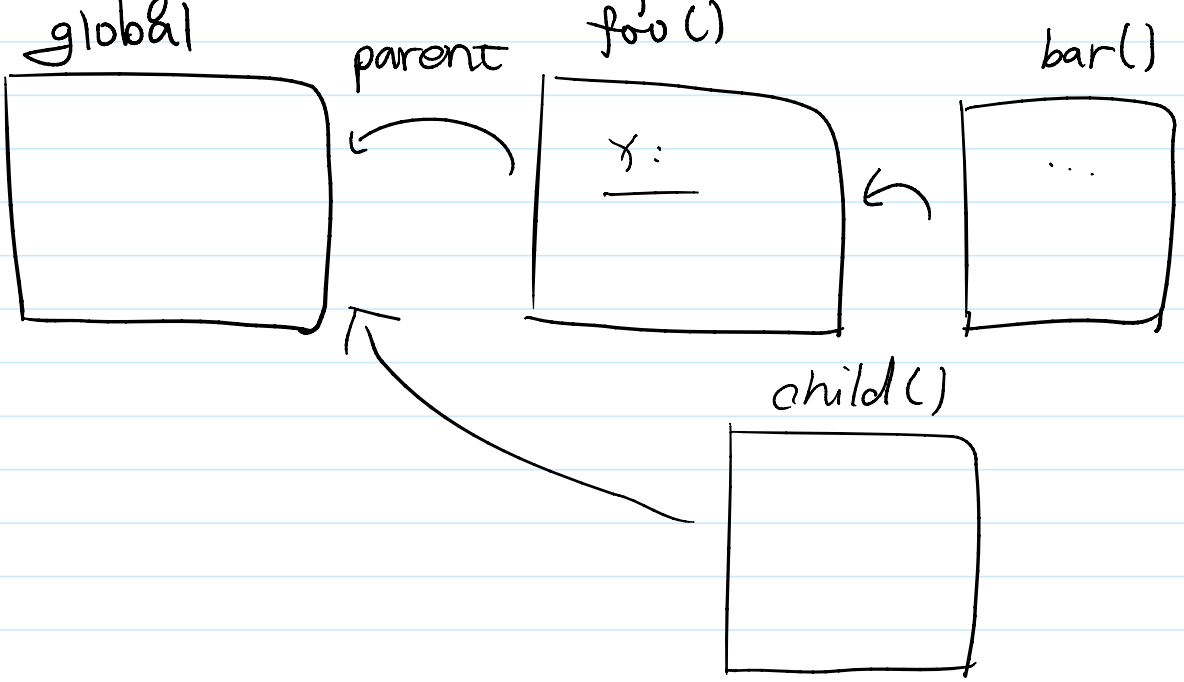
Algol-style
→ A name declared in a scope is known to its scope & all inner scopes down to the point at which an identically named variable is declared.

```
foo () {
  val i = 3 {
    val i = 4
      :
```

↑ shadowing

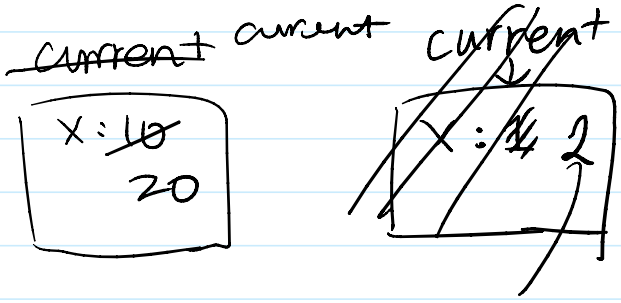
shadowing

How does it work?
→ nesting (or chain) of environments




```
{
  var x = 10
    {
      var x = 1
        x = x + x
    }
    x = x + x
    x
}
```

current ~~current~~ current



aside: JavaScript

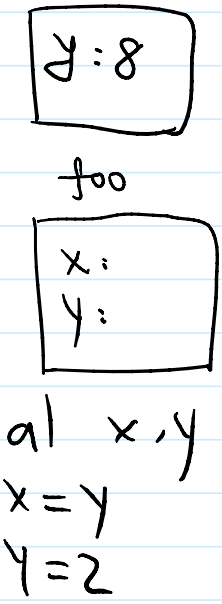
```
function foo () {        // return 2.
    var x = 1            // In scala,
    { var x = 2 }        // return 1.
return x
```

```
    { var x = 2 }  // return 1.
  return x
}
```

Declaration Order

→ Modula-3 style (A language)
→ declaration order doesn't matter

```
val y = 8
def foo() = {
    val x = y
    val y = 2
    x
}
foo()
```



Why unordered declarations?

→ mainly recursive

```
def foo() = {          def bar = {
    bar()                  foo()
}                      }
```

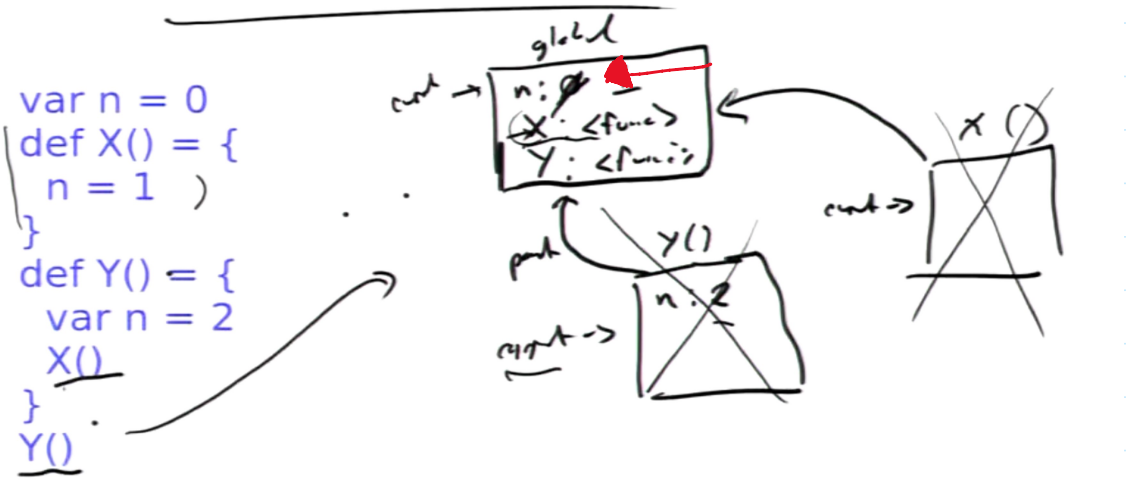
→ No solution unless treated unordered.

Would it make sense?

```
val x = y    (wouldn't work!)
val y = x
```

→ $x = y \cup \{a\}$
$y = x \cup \{b\}$

→ That is solution
$x = \{a, b\} \mid \{a, b, c\}$
$y = \{a, b\} \mid \{a, b, c\}$



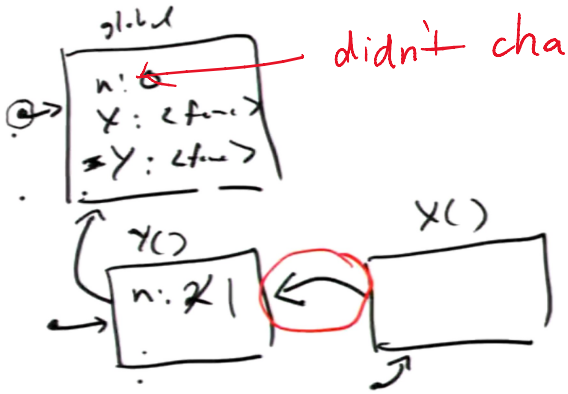
- current env → static following the func calls,
- parent env → env in which the func was defined.

→ look up symbols
→ start at current → follow the parent ptrs.

Dynamic Scoping

→ parent ptrs are based on the current env.

```
var n = 0
def X() = {
  n = 1
}
def Y() = {
  var n = 2
  X()
}
Y()
```



→ This seems good ?
→ errors

```
def foo() = {x}
def bar() {
    var x = ...
    foo() :
}
// In scala, it is broken.
// In dynamic, it is okay.
```