[10-10-18] ⟹ Basically just review of prev. lectures

See the code for implementing substitution!

Recall: **Environments** are collections of multiple bindings
2 things you can do with an Env:
- look up what vars have what values (lookup keyword)
- extend the Env. (usually by "cons"ing onto it...)

Recall: **let** (and where I think...) clauses basically allow you to **declare local variables**

Recall: **Scope** pretty much works the way you'd expect it to...
"Really, every language you know uses static scoping... you already know what static scoping is."

"Substitutions and Environments are very similar notions."
Both are ways of handling variable declarations

Recall: **UndeFined Variable Error** is exactly what you'd expect, so don't worry about it, you already know this.

Example:

(shorthand for this)

```
x = 2
y = x + 1
```
⟺
```
Declare "x" (Number 2)
  Declare "y" (add (Variable "x") (Number 1))
```

```
x = 2
y = x + 1
z = y + 2
x * y * z
```
→ Declare "z" (add (Variable "y") (Number 2))
→ Multiply (. . .

```
x = 2
y = x + 1
z = y + 2
x * y * z
```
⇨
```
y = 2 + 1
z = y + 2
x * y * z
```
⇨
```
y = 3
z = y + 2
x * y * z
```
⇨ and so on...

keyword:
FromJust → If you give me Just v, I'll return v.
If you give me Nothing, I'll return an error

I think Environments are like code blocks in that it's a way of imposing a scope on some code that creates bindings by declaring variables (or something like that...)

"Shadowing" not that complicated...

When there are two bindings for the same variable, what do you do?

```
var x = 9
var x = x * x
x + x
```
⇨
```
x ↦ 9
x ↦ 81
162
```

Basically, the most recent binding "shadows" (more like "overshadows") the first binding, making it go away