MIT EECS 6.001
Environment Diagrams

Why are we doing this?

To understand scope of variables

This is how Scheme is actually implemented

It's boring, but important: It's a technical investment now, but you'll get a lot out of it soon.

Looking Ahead:

- We'll be writing an evaluator in Scheme. You'll see that the code we write closely relates to the rules of the environment model.
- In Object Oriented Programming, we need a model to represent hierarchies, shadowing, and inheritance.

What are environment diagrams made of?

Frames: We draw a frame as a box. In the box go bindings. Every frame (except the frame representing the global environment) needs to have a link to exactly one other frame.

Bindings: A binding is an association between an identifier (or symbol) to a value.

Procedure Objects: These are special objects (symbolized by the double bubble) created by evaluating a lambda expression as described below.

Notes:

We start with a global environment that has all the built-in stuff defined. It's sometimes handy to draw the built-in's we use.

As we're evaluating stuff, we always have a current environment (pointer to one frame that in turn points upwards to other frames, stopping at the global environment).

You can use colors to track different procedures.

Limitations of the environment model: It gets messy quite easily and it does not keep track of expressions to evaluate.

Remember: it’s all very easy and completely mechanical!
Don’t burn your neurons.

What are the rules again?

Combination

To evaluate a combination with respect to an environment, first evaluate the subexpressions with respect to the environment and then apply the value of the operator subexpression to the values of the operand subexpressions.

Looking up an identifier

Look for a value in the current frame.

If there is no value for that identifier in the current frame, follow the link from the current frame to the one that it is linked from.

Continue in this manner until we find a binding for the identifier we're looking up or until we run out of frames in our chain of links (i.e. we come to the global environment). If there's no binding in the global environment, the identifier we're looking up is an unbound variable.

Define

Define adds a binding to the frame in which it is evaluated.

Set!

Recursively lookup the identifier starting in the current environment. If not found, error.

Rebind the identifier in the frame it was found to the value of the expression.

Lambda

Evaluating a lambda expression will result in a two-part procedure object (two circles next to each other -- the double bubble).

The pointer of the left circle points down to a list of the parameters and the body of the procedure.

The pointer of the right circle points up to the environment frame in which the lambda expression was evaluated. Note that nothing else happens until we apply the procedure.

Applying a procedure

1 Draw a new environment frame.

2 Link the new environment frame to the environment frame pointed to ***by the right bubble of the procedure object.***

3 Bind the parameter variables (1^{st} bubble) to the arguments of the procedure (similar to define).

4 Evaluate the body expression with respect to the current environment.

Let

Just de-sugar!

In practice, drop a frame, link it to the current environment, bind the variables and evaluate the expression in this new environment. .