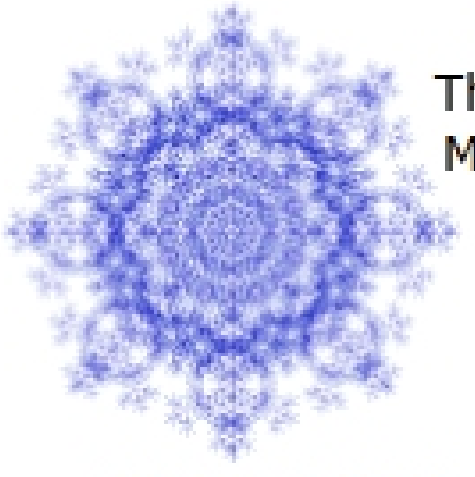


Lecture 21: Think Globally, Mutate Locally

Most Beautifulest Snowflake in the world
by Jordan Suter, Jon Faulkner

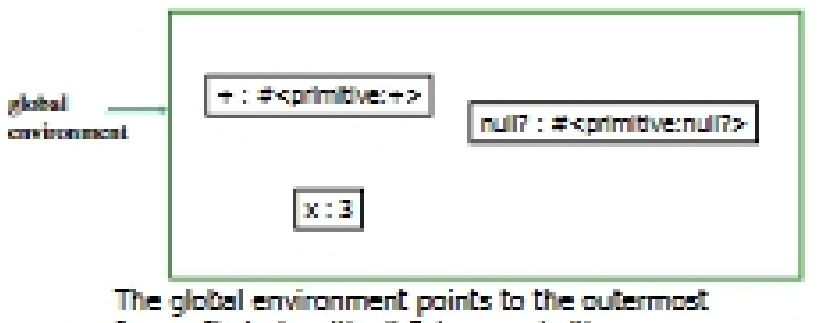
CS1.5: Computer Science
University of Virginia
Computer Science

David Evans
http://www.cs.virginia.edu/cs1.5

Review: Names, Places, Mutation

- A **name** is a **place** for storing a value.
- A **define** creates a new place
- A **cons** application creates two new places, the **car** and the **cdr**
- A **frame** is a collection of places
- An **environment** is a frame and a pointer to a parent environment
- **(set!** *name expr***)** changes the value in the place *name* to the value of *expr*

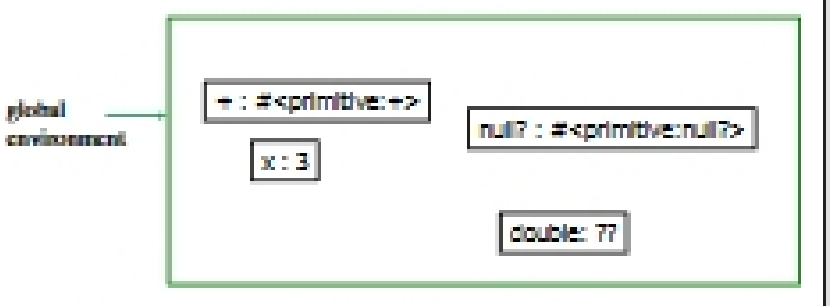
Environments



The global environment points to the outermost frame. It starts with all Scheme primitives.

```
> (define x 3)
```


Procedures

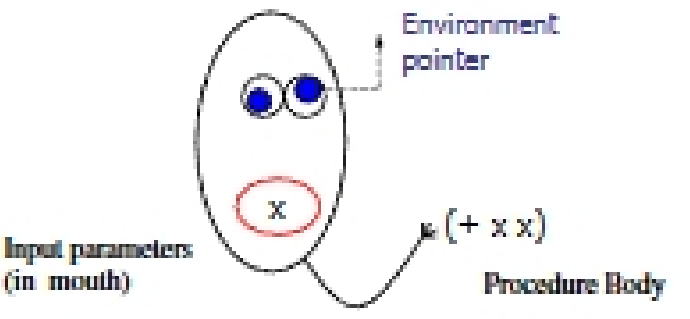



```
> (define double (lambda (x) (+ x x)))
```

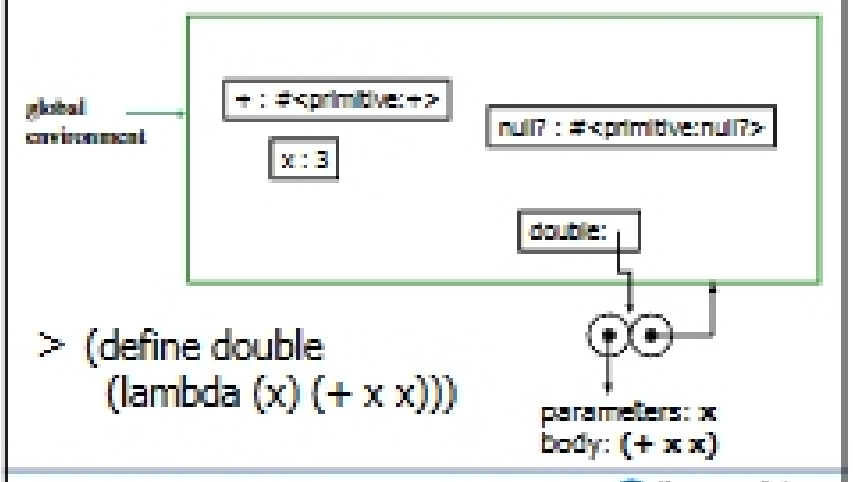

How to Draw a Procedure

- A procedure needs **both code and an environment**
  - We'll see why soon
- We draw procedures like this:

How to Draw a Procedure (for artists only)



Procedures




```
> (define double
    (lambda (x) (+ x x)))
```


Application

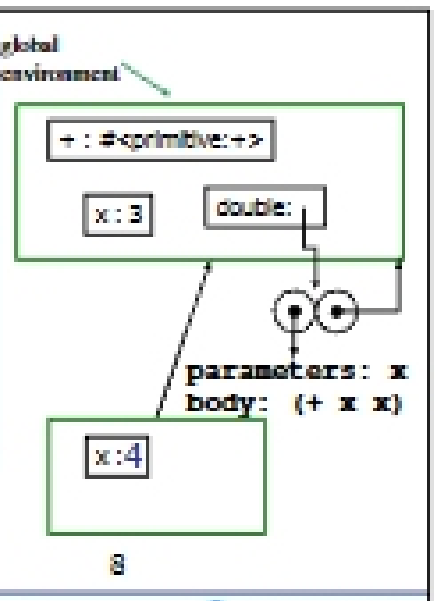
- Old rule: (Substitution model)

Apply Rule 2: Compounds. If the procedure is a *compound procedure*, **evaluate** the body of the procedure with each formal parameter replaced by the corresponding actual argument expression value.

New Rule: Application

1. Construct a new frame, enclosed in the environment of this procedure
2. Create places in that frame with the names of each parameter
3. Put the values of the parameters in those places
4. Evaluate the body in the new environment

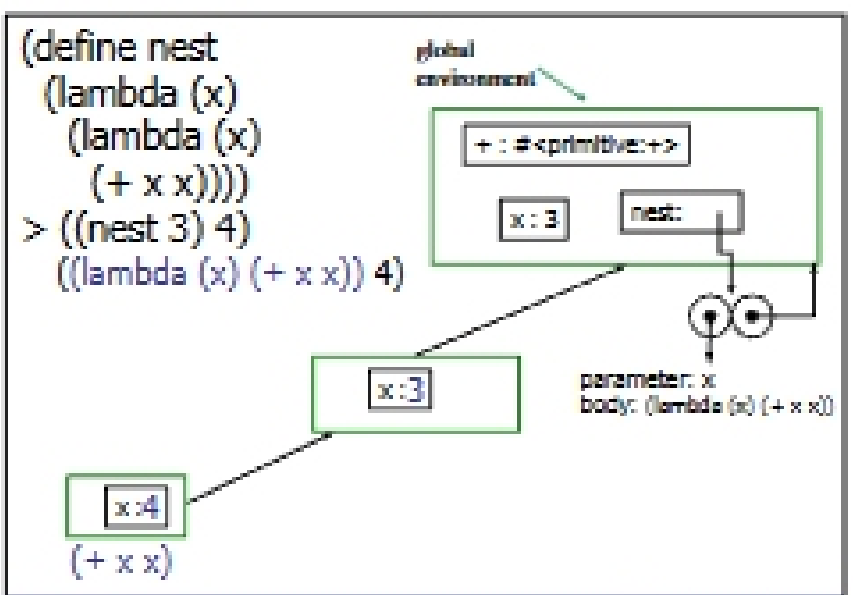
1. Construct a new frame, enclosed in the environment of this procedure
2. Make places in that frame with the names of each parameter
3. Put the values of the parameters in those places
4. Evaluate the body in the new environment




```
> (double 4)
8
```

8


```
(define nest
  (lambda (x)
    (lambda (x)
      (+ x x))))
> ((nest 3) 4)
  ((lambda (x) (+ x x)) 4)
```



Evaluation Rule 2 (Names)

If the expression is a *name*, it evaluates to the value associated with that name.

To find the value associated with a name, look for the name in the frame pointed to by the evaluation environment. If it contains a place with that name, use the value in that place. If it doesn't, evaluate the name using the frame's parent environment as the new evaluation environment. If the frame has no parent, error (name is not a place).

evaluate-name

```
(define (evaluate-name name env)
  (if (null? env) (error "Undefined name: ...")
      (if (frame-contains name (get-frame env))
          (lookup name (get-frame env))
          (evaluate-name name
            (parent-environment
              (get-frame env))))))
```

Hmm...maybe we can define a Scheme interpreter in Scheme!

Charge

- PS5 due Wednesday