Environment model

; Can you figure out why this code works?
(define make-counter
  (lambda (n)
    (lambda ()
      (set! n (+ n 1))
      n)))
(define ca (make-counter 0))
(ca) ==> 1
(ca) ==> 2
(define cb (make-counter 0))
(cb) ==> 1
(ca) ==> 3  ; ca and cb are independent

The Environment Model is:

- A precise, completely mechanical description of:
  - name-rule looking up the value of a variable
  - define-rule creating a new definition of a var
  - set-rule changing the value of a variable
  - lambda-rule creating a procedure
  - application applying a procedure

- Enables analyzing arbitrary scheme code:
  - Example: make-counter

- Basis for implementing a scheme interpreter
  - for now: draw EM state with boxes and pointers
  - later on: implement with code

A shift in viewpoint

- As we introduce the environment model, we are going to shift our viewpoint on computation
- Variable:
  - OLD – name for value
  - NEW – place into which one can store things
- Procedure:
  - OLD – functional description
  - NEW – object with inherited context
- Expressions
  - Now only have meaning with respect to an environment

Frame: a table of bindings

- Binding: a pairing of a name and a value

Example:
- x is bound to 15 in frame A
- y is bound to (1 2) in frame A

Expression evaluation in the environment model

- All evaluation occurs in an environment
  - The current environment changes when the interpreter applies a procedure

- The top environment is called the global environment (GE)
  - Only the GE has no enclosing environment
  - To evaluate a combination
    - Evaluate the subexpressions in the current environment
    - Apply the value of the first to the values of the rest

Environment: a sequence of frames

- Environment E1 consists of frames A and B
- Environment E2 consists of frame B only
  - A frame may be shared by multiple environments

```
E2
  z: 10

E1
  x: 15
  y:
```

```
A
  x: 15
  y:
```

```
B
  x: 10
```

```
  this arrow is called the enclosing environment pointer
```

```
E1
  x: 15
  y:
```

```
```

```
E2
  z: 10

```

```
E1
  x: 15
  y:
```

```
B
  x: 10
```

```
  this arrow is called the enclosing environment pointer
```

```
E2
  z: 10

E1
  x: 15
  y:
```

```
A
  x: 15
  y:
```

```
  1
  2
```
**Name-rule**

- A name X evaluated in environment E gives the value of X in the first frame of E where X is bound.
  
- In E1, the binding of x in frame A shadows the binding of x in B.

**Define-rule**

- A define special form evaluated in environment E creates or replaces a binding in the first frame of E.

**Set!-rule**

- A set! of variable X evaluated in environment E changes the binding of X in the first frame of E where X is bound.

**Compare define versus set!**

- (define z 20) in E1
- (define z 25) in E1

**Your turn: evaluate the following in order**

- (+ z 1) in E1
- (set! z (+ z 1)) in E1
- (define z (+ z 1)) in E1
Double bubble: how to draw a procedure

A compound proc that squares its argument

Lambda-rule

• A lambda special form evaluated in environment E creates a procedure whose environment pointer is E

Example: inc-square

Example cont'd: (inc-square 4) | GE
Lessons from the inc-square example

- EM doesn’t show the complete state of the interpreter
- missing the stack of pending operations
- The GE contains all standard bindings (*, cons, etc)
  - omitted from EM drawings
- Useful to link environment pointer of each frame to the procedure that created it

Example: make-counter

- Counter: something which counts up from a number

    (define make-counter
        (lambda (n)
            (lambda () (set! n ( + n 1)) n)))

    (define ca (make-counter 0))
    (ca) ==> 1
    (ca) ==> 2
    (define cb (make-counter 0))
    (cb) ==> 1
    (cb) ==> 2 ; ca and cb are independent
(define cb (make-counter 0)) | GE

Capturing state in local frames & procedures

Lessons from the make-counter example
- Environment diagrams get complicated very quickly
  - Rules are meant for the computer to follow, not to help humans
- A lambda inside a procedure body captures the frame that was active when the lambda was evaluated
  - this effect can be used to store local state

Let and the environment

Let and the environment
Capturing state with let

(define ticker
  (let ((state 0))
    (lambda ()
      (set! state (+ state 1)))
    state)))

GE

p:
  b: (set! state (+ state 1))
  state

ticker: state: E1 E2 E3

E1: state: E2 E3

Summary of environments

- Evaluation of all expressions and subexpressions take place with respect to an environment.
- Environments serve as dictionaries to define all terms
- Specific rules for creating and manipulating environments
- Shift in environment takes place with procedure application