CS61A Lecture 16 & 17

2011-07-18

Colleen Lewis

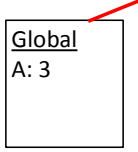
Environment Diagrams

- Tedious algorithm
  - But it helps us keep track of how scheme ACTUALLY evaluates expressions
- Always re-write syntactic sugar
- Always re-write `let`
- Follow the rules

IIB2. "`define` adds a new binding to the current frame"

```
STk>(define A 3)
```

This is a frame (also known as an environment)



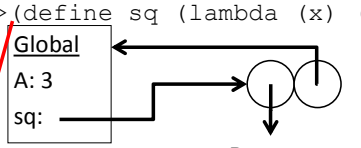
The current frame always starts as the Global frame. E.g. the `STk` prompt is the global environment

Current frame: Global

IIB1. "`lambda` creates a procedure"

- Left bubble points to the formal parameters & body
- Right bubble points to the current environment

```
STk>(define A 3)
STk>(define sq (lambda (x) (* x x)))
```



IIB2. "`define` adds a new binding to the current frame"

Current frame: Global

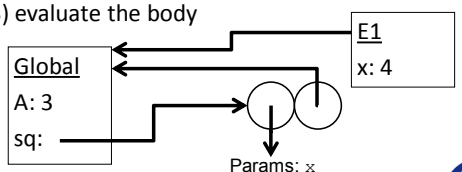
Procedure Call

IIA Step1. "evaluate the arguments"

IIA Step2.

```
STk>(sq 4)
```

- (a1) draw a frame
- (a1) bind the formal parameters
- (a2) extend environment the R bubble points to
- (a3) evaluate the body



Current frame: ~~Global~~ E1

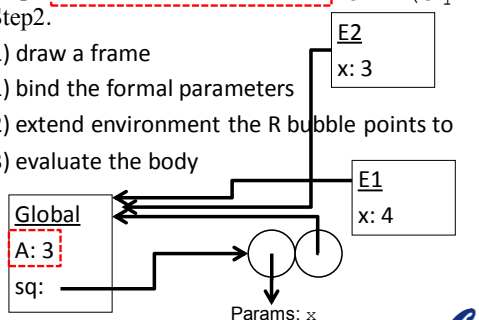
Procedure Call

IIA Step1. "evaluate the arguments"

IIA Step2.

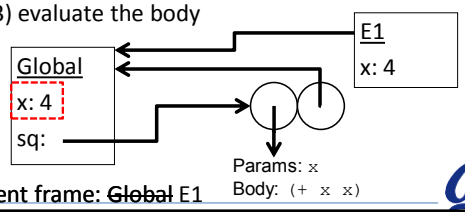
```
STk>(sq 4)
STk>(sq A)
```

- (a1) draw a frame
- (a1) bind the formal parameters
- (a2) extend environment the R bubble points to
- (a3) evaluate the body



Current frame: ~~Global~~ E2

Procedure Call
IIA Step1. "evaluate the arguments"
IIA Step2. `STk>(sq x)`
- (a1) draw a frame
- (a1) bind the formal parameters
- (a2) extend environment the R bubble points to
- (a3) evaluate the body



Current frame: ~~Global~~ E1

THE RULES (so far)

- IIB2. "`define` adds a new binding to the current frame"
- IIB1. "`lambda` creates a procedure"

Procedure Call
IIA Step1. "evaluate the arguments"
IIA Step2.
- (a1) draw a frame
- (a1) bind the formal parameters
- (a2) extend environment the R bubble points to
- (a3) evaluate the body

```
STk> (define (fact n)
       (if (< n 2)
           n
           (* n (fact (- n 1)))))
STk> (fact 3)
```

Draw the environment diagram for this…

FIRST: rewrite without Syntactic Sugar!!!!

How many environments did you have?

A) 1 (just global)
B) 2 (E1 extends global)
C) 3 (E1 & E2 extend global)
D) 3 (E1 extends global & E2 extends E1)

Draw the environment diagram for this…

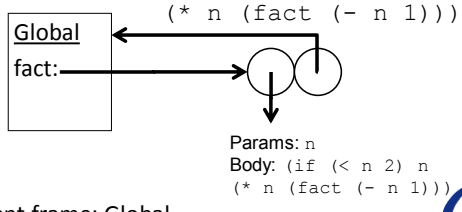
```
STk> (define (fact n)
       (if (< n 2)
           n
           (* n (fact (- n 1)))))
STk> (fact 3)
```

FIRST: rewrite without Syntactic Sugar!!!!

IIB1. "`lambda` creates a procedure"

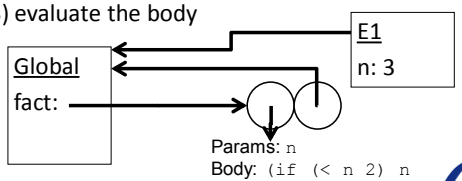
- Left bubble points to the formal parameters & body
- Right bubble points to the current environment

```
STk>(define fact
       (lambda (n) (if (< n 2) n
                       (* n (fact (- n 1))))))
```



Current frame: Global

Procedure Call
IIA Step1. "evaluate the arguments"
IIA Step2. `STk>(fact 3)`
- (a1) draw a frame
- (a1) bind the formal parameters
- (a2) extend environment the R bubble points to
- (a3) evaluate the body

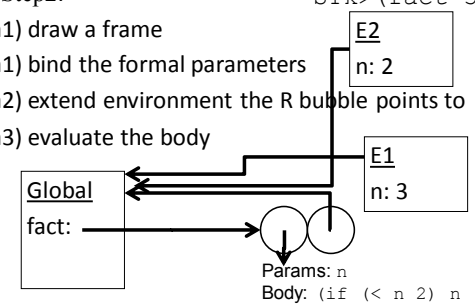


Current frame: ~~Global~~ E1

Procedure Call
IIA Step1. "evaluate the arguments"
IIA Step2.

```
STk>(fact 3)
```

- (a1) draw a frame
- (a1) bind the formal parameters
- (a2) extend environment the R bubble points to
- (a3) evaluate the body



Current frame: ~~Global E1~~ E2

Things that we don't keep track of

- Return values
- Pending calculations
  - e.g. `(fact 3)` was waiting for `(fact 2)` to finish.
- Current environment
  - officially it is not tracked
  - but we recommend you do!

Where can we access `a` and `b`?

```
STk> (define (mystery x y)
               (define a 3)
               (define b 4))
STk> a
```

Is this an error?

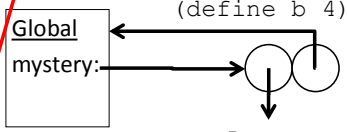
A) Yes B) No C) I don't know

```
STk> (define mystery (lambda (x y)
               (define a 3)
               (define b 4)))
```

IIB1. "`lambda` creates a procedure"

- Left bubble points to the formal parameters & body
- Right bubble points to the current environment

```
STk>(define mystery (lambda (x y)
                      (define a 3)
                      (define b 4)))
```



Params: x y
Body: (define a 3)...

IIB2. "`define` adds a new binding to the current frame"

Current frame: Global

Where can we access `a` and `b`?

```
STk> (define (mystery x y)
               (define a 3)
               (define b 4))
STk> (mystery 1 2)
STk> a
```

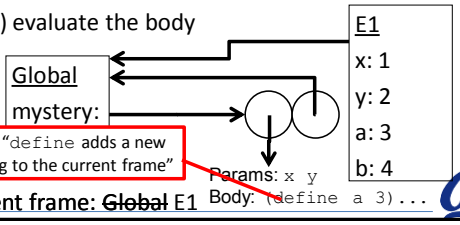
Is this an error?

A) Yes B) No C) I don't know

Procedure Call
IIA Step1. "evaluate the arguments"
IIA Step2.

```
STk>(mystery 1 2)
```

- (a1) draw a frame
- (a1) bind the formal parameters
- (a2) extend environment the R bubble points to
- (a3) evaluate the body



IIB2. "`define` adds a new binding to the current frame"

Current frame: ~~Global~~ E1

Draw the environment diagram

```
STk>(define (rockin x)
       (define sq (lambda (x)(* x x)))
       (define b (+ x x))
       b)
STk>(rockin 3)
```

The first step is to:

A) Rewrite without syntactic sugar

B) Draw bubbles

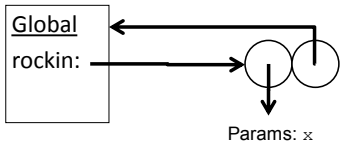
C) Draw a frame

The procedure defined as `sq` points to:

A) Global B) E1 C) E2

IIB1. "`lambda` creates a procedure"

- Left bubble points to the formal parameters & body
- Right bubble points to the current environment



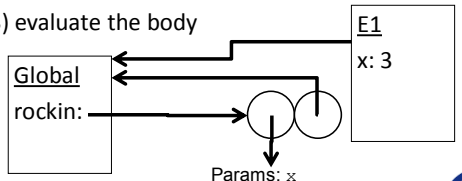
Current frame: Global

Procedure Call

IIA Step1. "evaluate the arguments"

IIA Step2. `STk>(rockin 3)`

- (a1) draw a frame
- (a1) bind the formal parameters
- (a2) extend environment the R bubble points to
- (a3) evaluate the body

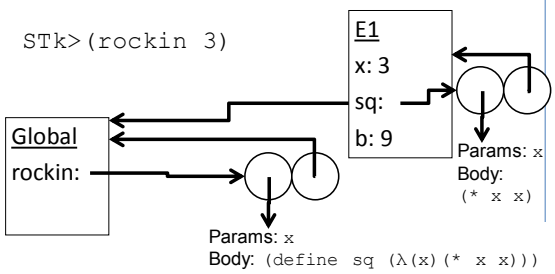


Current frame: ~~Global~~ E1

IIB1. "`lambda` creates a procedure"

- Left bubble points to the formal parameters & body
- Right bubble points to the current environment

`STk>(rockin 3)`



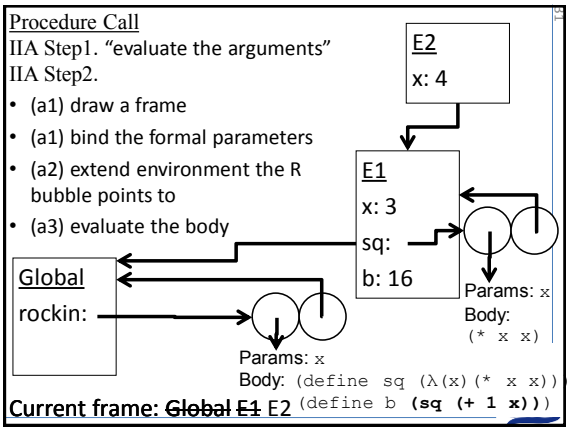
Current frame: ~~Global~~ E1 `(define b (+ x x))`

Procedure Call

IIA Step1. "evaluate the arguments"

IIA Step2.

- (a1) draw a frame
- (a1) bind the formal parameters
- (a2) extend environment the R bubble points to
- (a3) evaluate the body



Current frame: ~~Global E1~~ E2 `(define b (sq (+ 1 x)))`

Rewrite without syntactic sugar

```
(define (plus)
    (let ((a 3) (b 4))
         (+ a b)))
```

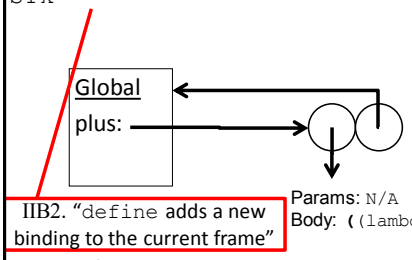
How many `lambda`s?

A) 0 B) 1 C) 2 D) 3 E)4

IIB1. "`lambda` creates a procedure"

- Left bubble points to the formal parameters & body
- Right bubble points to the current environment

STk



IIB2. "`define` adds a new binding to the current frame"

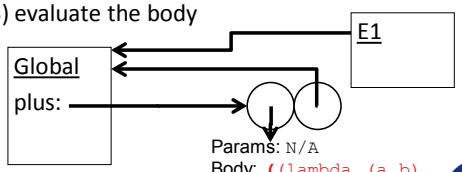
Current frame: Global

Procedure Call

IIA Step1. "evaluate the arguments"

IIA Step2. STk> (plus)

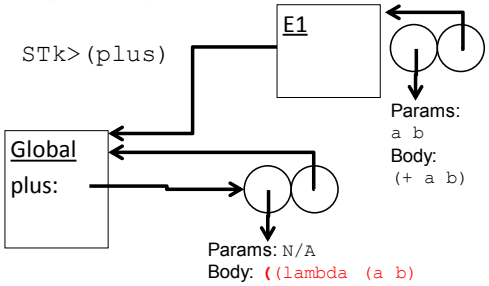
- (a1) draw a frame
- (a1) bind the formal parameters
- (a2) extend environment the R bubble points to
- (a3) evaluate the body



Current frame: ~~Global~~ E1

IIB1. "`lambda` creates a procedure"

- Left bubble points to the formal parameters & body
- Right bubble points to the current environment



Current frame: ~~Global~~ E1

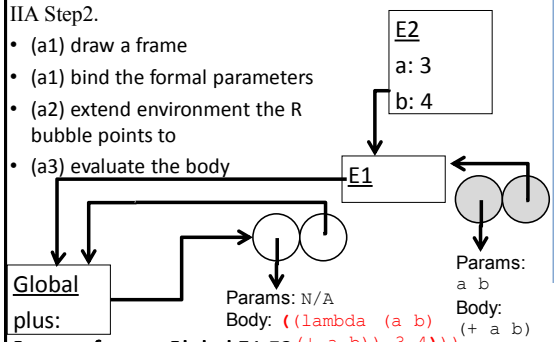
Procedure Call

STk> (plus)

IIA Step1. "evaluate the arguments"

IIA Step2.

- (a1) draw a frame
- (a1) bind the formal parameters
- (a2) extend environment the R bubble points to
- (a3) evaluate the body



Current frame: ~~Global E1~~ E2

Environment Diagram Status Update

A) Got it
B) Getting it
C) Getting bits of it
D) Not getting it

`global-count`

```
(define  counter 0)
(define (global-count)
    (set! counter (+ counter 1)))

STk> (global-count)
1
STk> (global-count)
2
STk> (global-count)
3
```

mystery-count

```
(define (mystery-count)
    (let ((counter 0))
        (set! counter (+ counter 1))
        counter))
STk> (mystery-count)
1
STk> (mystery-count)
2
STk> (mystery-count)
3
```

Does this work?
A) Yes
B) No
C) Sometimes

broken-count

```
(define (broken-count)
    (let ((counter 0))
        (set! counter (+ counter 1))
        counter))
STk> (broken-count)
1
STk> (broken-count)
1
STk> (broken-count)
1
```

make-count

```
(define (make-count)
  (let ((result 0))
    (lambda ()
       (set! results (+ result 1))
        result)))
STk>(define dracula (make-count))
STk>(dracula)
1
STk>(dracula)
2
```

```
STk>(dracula)
3
STk>(dracula)
4
STk> (define ryan (make-count)
ryan
STk> (ryan)
1
STk> (ryan)
2
STk> (dracula)
5
```

global-count

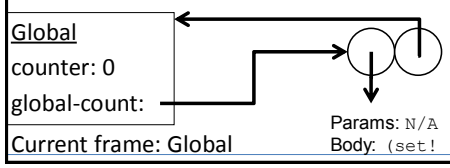
```
(define  counter 0)
(define (global-count)
    (set! counter (+ counter 1)))
```

Rewrite without syntactic sugar:

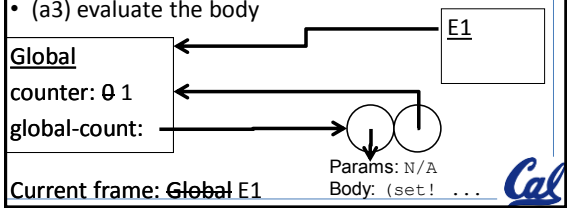
IIB1. "lambda creates a procedure"

- Left bubble points to the formal parameters & body
- Right bubble points to the current environment

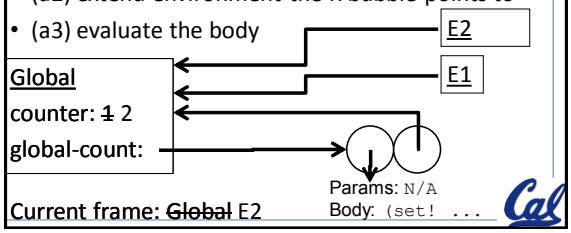
```
STk> (define counter 0)
```



Procedure Call
IIA Step1. "evaluate the arguments"
IIA Step2. STk>(global-count)
- (a1) draw a frame
- (a1) bind the formal parameters
- (a2) extend environment the R bubble points to
- (a3) evaluate the body



Procedure Call
IIA Step1. "evaluate the arguments"
IIA Step2. STk>(global-count)
- (a1) draw a frame STk>(global-count)
- (a1) bind the formal parameters
- (a2) extend environment the R bubble points to
- (a3) evaluate the body



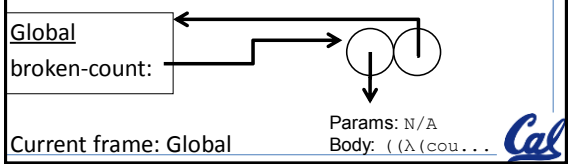
broken-count

```
(define (broken-count)
    (let ((counter 0))
        (set! counter (+ counter 1))
        counter))
```

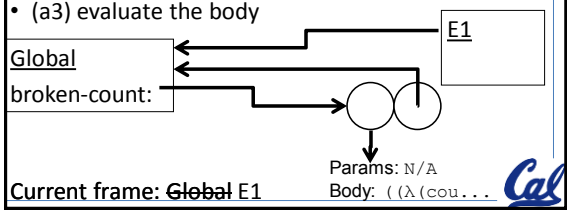
Rewrite without syntactic sugar

IIB1. "lambda creates a procedure"

- Left bubble points to the formal parameters & body
- Right bubble points to the current environment

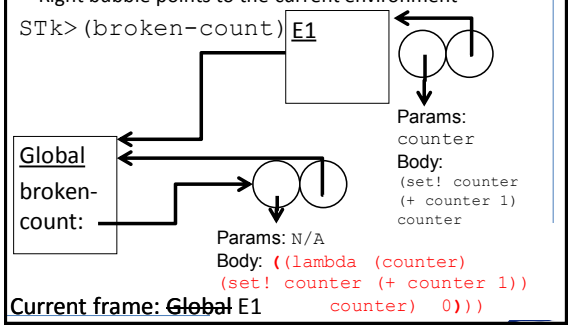


Procedure Call
IIA Step1. "evaluate the arguments"
IIA Step2. STk>(broken-count)
- (a1) draw a frame
- (a1) bind the formal parameters
- (a2) extend environment the R bubble points to
- (a3) evaluate the body

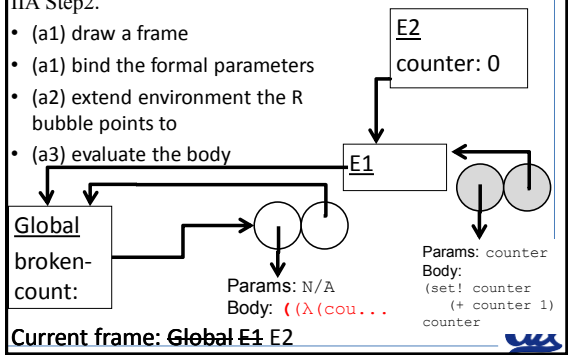


IIB1. "lambda creates a procedure"

- Left bubble points to the formal parameters & body
- Right bubble points to the current environment



Procedure Call STk>(broken-count)
IIA Step1. "evaluate the arguments"
IIA Step2.
- (a1) draw a frame
- (a1) bind the formal parameters
- (a2) extend environment the R bubble points to
- (a3) evaluate the body



Current frame: ~~Global~~ ~~E1~~ E2

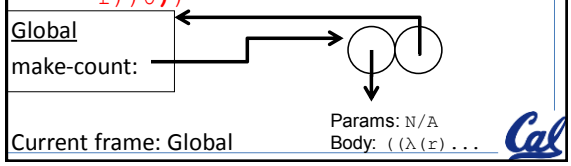
Rewrite `make-count`

```
(define (make-count)
  (let ((result 0))
    (lambda ()
      (set! results (+ result 1))
       result)))
```

IIB1. "`lambda` creates a procedure"

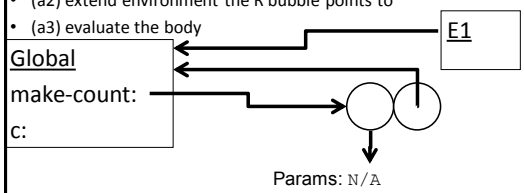
- Left bubble points to the formal parameters & body
- Right bubble points to the current environment

```
STk>(define make-count (lambda ()
  ((lambda (r) (lambda ()
      (set! r (+ r 1))
    r))0))
```



Current frame: Global

Procedure Call STk>(define c **(make-count)**)
IIA Step1. "evaluate the arguments"
IIA Step2.
- (a1) draw a frame
- (a1) bind the formal parameters
- (a2) extend environment the R bubble points to
- (a3) evaluate the body

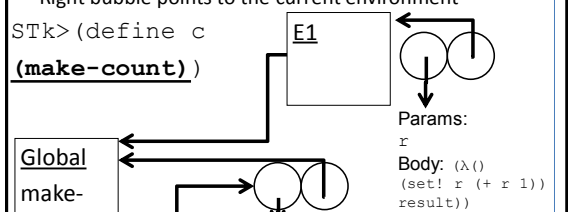


Current frame: ~~Global~~ E1

IIB1. "`lambda` creates a procedure"

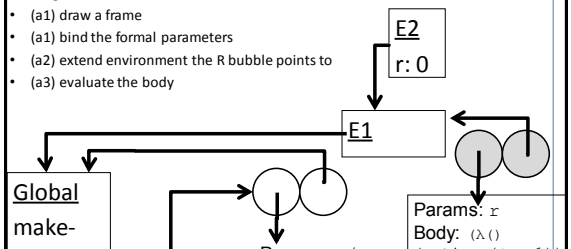
- Left bubble points to the formal parameters & body
- Right bubble points to the current environment

STk>(define c **(make-count)**)

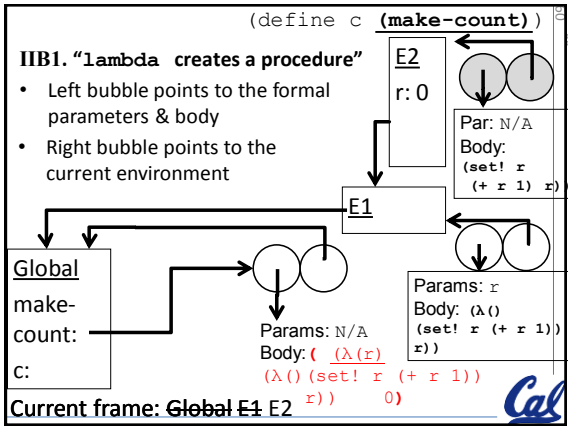


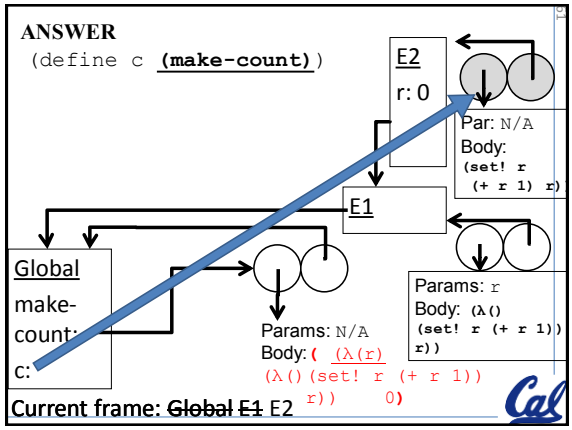
Current frame: ~~Global~~ E1

Procedure Call (define c **(make-count)**)
IIA Step1. "evaluate the arguments"
IIA Step2.
- (a1) draw a frame
- (a1) bind the formal parameters
- (a2) extend environment the R bubble points to
- (a3) evaluate the body



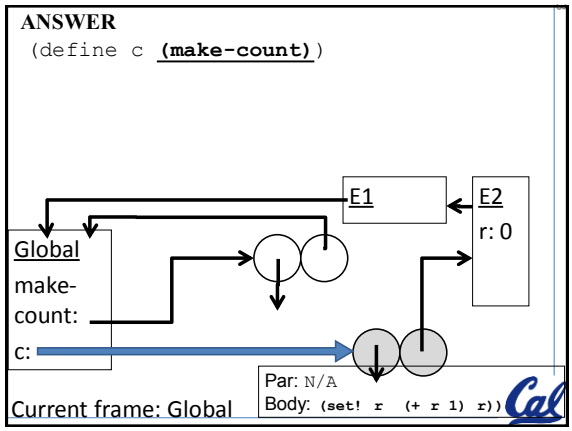
Current frame: ~~Global~~ ~~E1~~ E2





make-count

```
(define make-count (lambda ()
  ((lambda (r)
     (lambda ()
       (set! r (+ r 1))
       r))
   0)))
(define c (make-count))
```



Environment Diagram Status Update

A) Got it
B) Getting it
C) Getting bits of it
D) Not getting it

Solutions

Draw the environment diagram for this…

```
STk> (define (fact n)
       (if (< n 2)
           n
           (* n (fact (- n 1)))))
STk> (fact 3)
```

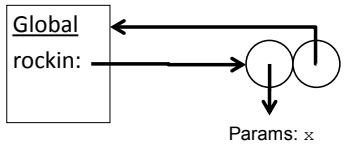
FIRST: rewrite without Syntactic Sugar!!!!

```
STk>(define fact
      (lambda (n)
        (if (< n 2) n
            (* n (fact (- n 1))))))
```

IIB1. "lambda creates a procedure"

- Left bubble points to the formal parameters & body
- Right bubble points to the current environment

```
STk>(define rockin (lambda (x)
      (define sq (lambda (x)(* x x)))
      (define b (+ x x))
      b))
```



Params: x
Body: (define sq ...

Current frame: Global

Rewrite without syntactic sugar

```
(define (plus)
    (let ((a 3) (b 4))
         (+ a b)))
```

How many lambdas?

A) 0 B) 1 C) 2 D) 3 E)4

```
(define plus (lambda ()
  ((lambda (a b)
     (+ a b))
   3 4)))
```

global-count

```
(define  counter 0)
(define (global-count)
    (set! counter (+ counter 1)))
```

Rewrite without syntactic sugar:

```
(define global-count
    (lambda()
      (set! counter (+ counter 1))))
```

broken-count

```
(define (broken-count)
    (let ((counter 0))
        (set! counter (+ counter 1))
        counter))
```

Rewrite without syntactic sugar

```
(define broken-count (lambda ()
    ((lambda (counter)
        (set! counter (+ counter 1))
        counter)
     0)))
```

Rewrite make-count

```
(define (make-count)
  (let ((result 0))
    (lambda ()
       (set! results (+ result 1))
        result)))
(define make-count (lambda ()
  ((lambda (result)
    (lambda ()
      (set! results (+ results 1))
      result))
   0))
```