Environment Diagrams

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

**IIB2. “define adds a new binding to the current frame”**

\[
\text{STk} > \text{(define } A \ 3) \]

This is a frame (also known as an environment)

The current frame always starts as the Global frame. E.g. the \text{STk} prompt is the global environment

Current frame: Global

**IIB1. “\textit{lambda} creates a procedure”**

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

\[
\text{STk} > \text{(define } A \ 3) \]

\[
\text{STk} > \text{(define sq (lambda } (x) \ (* \ x \ x)))
\]

Current frame: Global

**Procedure Call**

\textbf{IIA Step1. “evaluate the arguments”}

\textbf{IIA Step2.}

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

\[
\text{STk} > \text{(sq } 4) \quad \text{STk} > \text{(sq } A)
\]

Current frame: Global
THE RULES (so far)

IIB2. “define adds a new binding to the current frame”
IIB1. “lambda creates a procedure”

Procedure Call
I IA 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)

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)

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

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!!!!
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)))

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

IIB2. “define adds a new binding to the current frame”

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

STk> (mystery 1 2)

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

Procedure Call

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

Current frame: Global

```
Global
rockin:
```

```
<table>
<thead>
<tr>
<th>E1</th>
<th>x: 3</th>
</tr>
</thead>
</table>

```

```
<table>
<thead>
<tr>
<th>E1</th>
<th>x: 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sq:</td>
</tr>
<tr>
<td></td>
<td>b: 9</td>
</tr>
</tbody>
</table>
```

```
Current frame: Global E1
```

IIA1. “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

```
Global
rockin:
```

```
<table>
<thead>
<tr>
<th>E1</th>
<th>x: 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sq:</td>
</tr>
<tr>
<td></td>
<td>b: 9</td>
</tr>
</tbody>
</table>
```

```
Current frame: Global E1
```

Current frame: Global

```
E1
x: 3
sq: |
```

```
髯

```
Current frame: Global E1 E2
```

```
<table>
<thead>
<tr>
<th>E2</th>
<th>x: 4</th>
</tr>
</thead>
</table>

```

```
<table>
<thead>
<tr>
<th>E1</th>
<th>x: 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sq:</td>
</tr>
<tr>
<td></td>
<td>b: 16</td>
</tr>
</tbody>
</table>
```

```
Current frame: Global E1 E2
```

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

```
Current frame: Global E1 E2
```

```
E2
x: 4
```

```
Current frame: Global E1 E2
```

```
<table>
<thead>
<tr>
<th>E1</th>
<th>x: 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sq:</td>
</tr>
<tr>
<td></td>
<td>b: 16</td>
</tr>
</tbody>
</table>

```

```
Current frame: Global E1 E2
```

```
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
IIB1. “\textit{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.
- (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

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

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

STk> (global-count) 1
STk> (global-count) 2
STk> (global-count) 3
**mystery-count**

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

STk> (mystery-count)
1
STk> (mystery-count)
2
STk> (mystery-count)
3

**broken-count**

```scheme
(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**

```scheme
(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)

**global-count**

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

Rewrite without syntactic sugar:

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

**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.
• (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

broken-count

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

Rewrite without syntactic sugar
Current frame: Global E1 E2

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

IIB. “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)))

STk>(define c (make-count))
IIB Step1. “evaluate the arguments”
IIB Step2.
• (a1) draw a frame
• (a1) bind the formal parameters
• (a2) extend environment the R bubble points to
• (a3) evaluate the body

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

IIB Step1. “evaluate the arguments”
IIB Step2.
• (a1) draw a frame
• (a1) bind the formal parameters
• (a2) extend environment the R bubble points to
• (a3) evaluate the body

STk> (define c (make-count))
IIB. “lambda creates a procedure”
• Left bubble points to the formal parameters & body
• Right bubble points to the current environment

STk> (define c (make-count))
IIB Step1. “evaluate the arguments”
IIB 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

Current frame: Global E1

Current frame: Global E1 E2
IIB1. “lambda creates a procedure”
- Left bubble points to the formal parameters & body
- Right bubble points to the current environment

(define c (make-count))

Environment Diagram Status Update
A) Got it
B) Getting it
C) Getting bits of it
D) Not getting it
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))))
```

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)))))
```

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)))
```

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

Rewrite without syntactic sugar:

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

```
(define 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)))
```

```
(define 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)))
```

IIB1. “lambda creates a procedure”

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

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

```
Global
rockin:

Params: x
Body: (define sq ...

Current frame: Global
```