Random Numbers

For HW 6, you need the `random` operator

It’s strange — it doesn’t return the same result every time for the same input:

```
> (random 3)
0
> (random 3)
2
> (random 3)
1
> (random 3)
2
```

Random Symbols

Suppose we need a **`random-symbol`** function

```
> (random-symbol 'huey 'dewey 'louie)
'dewey
> (random-symbol 'huey 'dewey 'louie)
'huey
> (random-symbol 'huey 'dewey 'louie)
'dewey
> (random-symbol 'huey 'dewey 'louie)
'louie
```

Random Symbols

Suppose we need a `random-symbol` function

```
> (random-symbol 'huey 'dewey 'louie)
'dewey
> (random-symbol 'huey 'dewey 'louie)
'huey
> (random-symbol 'huey 'dewey 'louie)
'dewey
> (random-symbol 'huey 'dewey 'louie)
'louie
```

Can we implement it with `random`?

Random Symbols

```
; random-symbol : sym sym sym -> sym
(define (random-symbol a b c)
  (cond
    [(= (random 3) 0) a]
    [(= (random 3) 1) b]
    [(= (random 3) 2) c]))
```

Random Symbols

```
; random-symbol : sym sym sym -> sym
(define (random-symbol a b c)
  (cond
    [(= (random 3) 0) a]
    [(= (random 3) 1) b]
    [(= (random 3) 2) c]))
```

This doesn’t work, because **`random`** produces a different result each time

Saving a Random Number

On the other hand...

```
(define n (random 3))
(list n n n)
```

Saving a Random Number

On the other hand...

```
(define n (random 3))
(list n n n)
```

produces `(list 0 0 0)`, `(list 1 1 1)`, or `(list 2 2 2)`

Constant definitions name constants, so `(random 3)` must be evaluted when defining **`n`**

Try it in the stepper

A Random Constant

Does this work?

```
(define n (random 3))

; random-symbol : sym sym sym -> sym
(define (random-symbol a b c)
  (cond
    [(= n 0) a]
    [(= n 1) b]
    [(= n 2) c]))
```

A Random Constant

Does this work?

```
(define n (random 3))

; random-symbol : sym sym sym -> sym
(define (random-symbol a b c)
  (cond
    [(= n 0) a]
    [(= n 1) b]
    [(= n 2) c]))
```

Not quite, because it always picks the same symbol

A Random Constant

Does this work?

```
(define n (random 3))

; random-symbol : sym sym sym -> sym
(define (random-symbol a b c)
  (cond
    [(= n 0) a]
    [(= n 1) b]
    [(= n 2) c]))
```

Not quite, because it always picks the same symbol

We want `(define n (random 3))` that is local to **`random-symbol`**'s body

Local Definitions

This works, in the ***Intermediate*** language

```
; random-symbol : sym sym sym -> sym
(define (random-symbol a b c)
  (local [(define n (random 3))]
    (cond
      [(= n 0) a]
      [(= n 1) b]
      [(= n 2) c])))
```

Local Definitions

This works, in the ***Intermediate*** language

```
; random-symbol : sym sym sym -> sym
(define (random-symbol a b c)
  (local [(define n (random 3))]
    (cond
      [(= n 0) a]
      [(= n 1) b]
      [(= n 2) c])))
```

- The **`local`** form has definitions and a body
- Local definitions are only visible in the body
- Local definitions are evaluated only when the **`local`** is evaluated
- The result of **`local`** is the result of its body

Evaluation with Local

```
(define (random-symbol a b c)
  (local [(define n (random 3))]
    (cond
      [(= n 0) a]
      [(= n 1) b]
      [(= n 2) c])))
(random-symbol 'huey 'dewey 'louie)
(random-symbol 'huey 'dewey 'louie)
```

Evaluation with Local

```
(define (random-symbol a b c)
  (local [(define n (random 3))]
    (cond
      [(= n 0) a]
      [(= n 1) b]
      [(= n 2) c])))
(random-symbol ’huey ’dewey ’louie)
(random-symbol ’huey ’dewey ’louie)
```

$\rightarrow$

```
(define (random-symbol ...) ...)
(local [(define n (random 3))]
  (cond
    [(= n 0) ’huey]
    [(= n 1) ’dewey]
    [(= n 2) ’louie]))
(random-symbol ’huey ’dewey ’louie)
```

Evaluation with Local

```
(define (random-symbol ...) ...)
(local [(define n (random 3))]
  (cond
    [(= n 0) 'huey]
    [(= n 1) 'dewey]
    [(= n 2) 'louie]))
(random-symbol 'huey 'dewey 'louie)
```

Evaluation with Local

```
(define (random-symbol ...) ...)
(local [(define n (random 3))]
  (cond
    [(= n 0) 'huey]
    [(= n 1) 'dewey]
    [(= n 2) 'louie]))
(random-symbol 'huey 'dewey 'louie)
```

$\rightarrow$

```
(define (random-symbol ...) ...)
(define n17 (random 3))
(cond
  [(= n17 0) 'huey]
  [(= n17 1) 'dewey]
  [(= n17 2) 'louie])
(random-symbol 'huey 'dewey 'louie)
```

Evaluation of **`local`** lifts and renames the definition

Evaluation with Local

```
(define (random-symbol ...) ...)
(define n17 (random 3))
(cond
  [(= n17 0) ’huey]
  [(= n17 1) ’dewey]
  [(= n17 2) ’louie])
(random-symbol ’huey ’dewey ’louie)
```

Evaluation with Local

```
(define (random-symbol ...) ...)
(define n17 (random 3))
(cond
  [(= n17 0) 'huey]
  [(= n17 1) 'dewey]
  [(= n17 2) 'louie])
(random-symbol 'huey 'dewey 'louie)
```

$\rightarrow$

```
(define (random-symbol ...) ...)
(define n17 1)
(cond
  [(= n17 0) 'huey]
  [(= n17 1) 'dewey]
  [(= n17 2) 'louie])
(random-symbol 'huey 'dewey 'louie)
```

Evaluation with Local

```
(define (random-symbol ...) ...)
(define n17 1)
(cond
  [(= n17 0) ’huey]
  [(= n17 1) ’dewey]
  [(= n17 2) ’louie])
(random-symbol ’huey ’dewey ’louie)
```

Evaluation with Local

```
(define (random-symbol ...) ...)
(define n17 1)
(cond
  [(= n17 0) 'huey]
  [(= n17 1) 'dewey]
  [(= n17 2) 'louie])
(random-symbol 'huey 'dewey 'louie)
```

$\rightarrow$

```
(define (random-symbol ...) ...)
(define n17 1)
(cond
  [(= 1 0) 'huey]
  [(= n17 1) 'dewey]
  [(= n17 2) 'louie])
(random-symbol 'huey 'dewey 'louie)
```

Evaluation of a constant name finds the value

Evaluation with Local

```
(define (random-symbol ...) ...)
(define n17 1)
(cond
  [(= 1 0) ’huey]
  [(= n17 1) ’dewey]
  [(= n17 2) ’louie])
(random-symbol ’huey ’dewey ’louie)
```

Evaluation with Local

```
(define (random-symbol ...) ...)
(define n17 1)
(cond
  [(= 1 0) 'huey]
  [(= n17 1) 'dewey]
  [(= n17 2) 'louie])
(random-symbol 'huey 'dewey 'louie)
```

$\rightarrow$

```
(define (random-symbol ...) ...)
(define n17 1)
(cond
  [false 'huey]
  [(= n17 1) 'dewey]
  [(= n17 2) 'louie])
(random-symbol 'huey 'dewey 'louie)
```

Evaluation with Local

```
(define (random-symbol ...) ...)
(define n17 1)
(cond
  [false 'huey]
  [(= n17 1) 'dewey]
  [(= n17 2) 'louie])
(random-symbol 'huey 'dewey 'louie)
```

Evaluation with Local

```
(define (random-symbol ...) ...)
(define n17 1)
(cond
  [false 'huey]
  [(= n17 1) 'dewey]
  [(= n17 2) 'louie])
(random-symbol 'huey 'dewey 'louie)
```

$\rightarrow$

```
(define (random-symbol ...) ...)
(define n17 1)
(cond
  [(= n17 1) 'dewey]
  [(= n17 2) 'louie])
(random-symbol 'huey 'dewey 'louie)
```

Evaluation with Local

```
(define (random-symbol ...) ...)
(define n17 1)
(cond
  [(= n17 1) 'dewey]
  [(= n17 2) 'louie])
(random-symbol 'huey 'dewey 'louie)
```

Evaluation with Local

```
(define (random-symbol ...) ...)
(define n17 1)
(cond
  [(= n17 1) 'dewey]
  [(= n17 2) 'louie])
(random-symbol 'huey 'dewey 'louie)
```

$\rightarrow$

```
(define (random-symbol ...) ...)
(define n17 1)
(cond
  [(= 1 1) 'dewey]
  [(= n17 2) 'louie])
(random-symbol 'huey 'dewey 'louie)
```

Evaluation with Local

```
(define (random-symbol ...) ...)
(define n17 1)
(cond
  [(= 1 1) ’dewey]
  [(= n17 2) ’louie])
(random-symbol ’huey ’dewey ’louie)
```

Evaluation with Local

```
(define (random-symbol ...) ...)
(define n17 1)
(cond
  [(= 1 1) 'dewey]
  [(= n17 2) 'louie])
(random-symbol 'huey 'dewey 'louie)
```

$\rightarrow$

```
(define (random-symbol ...) ...)
(define n17 1)
(cond
  [true 'dewey]
  [(= n17 2) 'louie])
(random-symbol 'huey 'dewey 'louie)
```

Evaluation with Local

```
(define (random-symbol ...) ...)
(define n17 1)
(cond
  [true ’dewey]
  [(= n17 2) ’louie])
(random-symbol ’huey ’dewey ’louie)
```

Evaluation with Local

```
(define (random-symbol ...) ...)
(define n17 1)
(cond
  [true 'dewey]
  [(= n17 2) 'louie])
(random-symbol 'huey 'dewey 'louie)
```

$\rightarrow$

```
(define (random-symbol ...) ...)
(define n17 1)
'dewey
(random-symbol 'huey 'dewey 'louie)
```

Evaluation with Local

```
(define (random-symbol ...) ...)
(define n17 1)
'dewey
(random-symbol 'huey 'dewey 'louie)
```

Evaluation with Local

```
(define (random-symbol ...) ...)
(define n17 1)
'dewey
(random-symbol 'huey 'dewey 'louie)
```

$\rightarrow$

```
(define (random-symbol ...) ...)
(define n17 1)
'dewey
(local [(define n (random 3))]
  (cond
    [(= n 0) 'huey]
    [(= n 1) 'dewey]
    [(= n 2) 'louie]))
```

Evaluation with Local

```
(define (random-symbol ...) ...)
(define n17 1)
'dewey
(local [(define n (random 3))]
  (cond
    [(= n 0) 'huey]
    [(= n 1) 'dewey]
    [(= n 2) 'louie]))
```

Evaluation with Local

```
(define (random-symbol ...) ...)
(define n17 1)
'dewey
(local [(define n (random 3))]
  (cond
    [(= n 0) 'huey]
    [(= n 1) 'dewey]
    [(= n 2) 'louie]))
```

$\rightarrow$

```
(define (random-symbol ...) ...)
(define n17 1)
'dewey
(define n45 (random 3))
(cond
  [(= n45 0) 'huey]
  [(= n45 1) 'dewey]
  [(= n45 2) 'louie])
```

Evaluation of **`local`** picks a new name each time

Evaluation with Local

```
(define (random-symbol ...) ...)
(define n17 1)
'dewey
(define n45 (random 3))
(cond
  [(= n45 0) 'huey]
  [(= n45 1) 'dewey]
  [(= n45 2) 'louie])
```

Evaluation with Local

```
(define (random-symbol ...) ...)
(define n17 1)
'dewey
(define n45 (random 3))
(cond
  [(= n45 0) 'huey]
  [(= n45 1) 'dewey]
  [(= n45 2) 'louie])
```

$\rightarrow$

```
(define (random-symbol ...) ...)
(define n17 1)
'dewey
(define n45 0)
(cond
  [(= n45 0) 'huey]
  [(= n45 1) 'dewey]
  [(= n45 2) 'louie])
```

Another Example

```
; kind-of-blue? : image -> bool
(define (kind-of-blue? i)
  (and
   (> (total-blue (image->color-list i))
      (total-red (image->color-list i)))
   (> (total-blue (image->color-list i))
      (total-green (image->color-list i)))))
```

Easier to read, converts image only once:

```
(define (kind-of-blue? i)
  (local [(define colors
            (image->color-list i))]
    (and (> (total-blue colors)
            (total-red colors))
         (> (total-blue colors)
            (total-green colors)))))
```

Another Example

```
(define (eat-apples l)
  (cond
    [(empty? l) empty]
    [(cons? l)
     (cond
       [(symbol=? (first l) 'apple)
        (eat-apples (rest l))]
       [else
        (cons (first l) (eat-apples (rest l)))])]))
```

Better:

```
(define (eat-apples l)
  (cond
    [(empty? l) empty]
    [(cons? l)
     (local [(define ate-rest (eat-apples (rest l)))]
       (cond
         [(symbol=? (first l) 'apple) ate-rest]
         [else (cons (first l) ate-rest)]))]))
```

Another Use for Local

`local` can define functions as well as constants

This is useful for making a function private

```
(define (random-symbol a b c)
  (local [(define (real-random-symbol a b c)
            (local [(define n (random 3))]
              (cond
                [(= n 0) a]
                [(= n 1) b]
                [(= n 2) c])))]
    (cond
      [(and (symbol? a) (symbol? b) (symbol? c))
       (real-random-symbol a b c)]
      [else (error ’random-symbol "not a symbol")])))
```

Another Use for Local

`local` can define functions as well as constants

This is useful for making a function private

```
(define (random-symbol a b c)
  (local [(define (real-random-symbol a b c)
            (local [(define n (random 3))]
              (cond
                [(= n 0) a]
                [(= n 1) b]
                [(= n 2) c])))]
    (cond
      [(and (symbol? a) (symbol? b) (symbol? c))
       (real-random-symbol a b c)]
      [else (error ’random-symbol "not a symbol")])))
```

Use Check Syntax and mouse over variables

Lexical Scope

```
(define (random-symbol a b c)
  (local [(define (real-random-symbol a b c)
            (local [(define n (random 3))]
              (cond
                [(= n 0) a]
                [(= n 1) b]
                [(= n 2) c])))]
    (cond
      [(and (symbol? a) (symbol? b) (symbol? c))
       (real-random-symbol a b c)]
      [else (error 'random-symbol "not a symbol")])))
```

Italic ***a*** could be changed to **z** without affecting non-italic **a**, no matter how the code runs

In other words, bindings are static; this is ***lexical scope***