CSC 151, Review Session with SamR, Week 6

Things to talk about

- Let’s talk about let
- And about for-each
- And cond
- Recursion (maybe a little)

What’s on the quiz?

- NOT recursion
- Conditionals, and, or, etc.
- Lists with cons, car, cdr, null, and null?
- Plus everything you learned in advance of that
- No turtle graphics

Let’s talk about Let

- You can use define within procedures, but we don’t allow you to because it can lead to future confusion.
- Alternative, with clean syntax and semantics

Form

(let ([NAME EXP]
        [NAME EXP])
  EXP
  EXP
  EXP)

Meaning

- Evaluate all of the top expressions (the ones that follow NAME)
- Associate the names with the results of those expressions (aka "update the 'name table'")
- Evaluate the remaining expressions, using those names
- Forget about the new names

Simple example

(let ([x 2]) (* x x))
• Step 1: Evaluate 2. Done.
• Step 2: Remember that "whenever we see x, use 2"
• Step 3: Look at (* x x) => (* 2 2)
• Step 4: Finish evaluating: 4
• Step 5: Forget what x means

Complication

(+ (let ([x 2]) (* x x)) x)

• Steps 1 ... 5, as before
• Now we have (+ 4 x) What value does x have? None! Crash

Sample interaction

> (let ([x 2]) (* x x))
  4
> (+ (let ([x 2]) (* x x)) x)
  . . reference to an identifier before its definition: x
> (begin 2 3)
  3
> (+ (begin (define x 2) (* x x)) x)
  . define: not allowed in an expression context in: (define x 2)

Another example

DEFINITIONS
(define x 3)
(let ([x 2]) (* x x))

OUTPUT
4

And another

DEFINITIONS
(define x 3)
(+ (let ([x 2]) (* x x)) x)

OUTPUT
7

More complicated

DEFINITIONS
(let ([x 2]
  [y (* x x)])
  (+ x y))

OUTPUT
boom!
More complicated

DEFINITIONS

(define x 3)
(let ([x 2]
      [y (* x x)])
  (+ x y))

INTERACTIONS
11

And more

DEFINITIONS
(define x 3)
(let* ([x 2]
       [y (* x x)])
  (+ x y))

INTERACTIONS
6

Let’s talk about for-each

(repeat N PROCEDURE PARAMETER1 PARAMETER2 ...)
(map PROCEDURE LIST1 LIST2 ...)
(for-each PROCEDURE LIST1 LIST2 ...)

- Difference between map, repeat, and for-each
- Commonality: All three call a procedure repeatedly
- Difference: Parameters
  - map and for-each - take the parameters from a list
  - repeat takes the same parameter each
- Difference: Specify the number of times to call the procedure
  - repeat: N
  - for-each and map: Depends on the length of the list
- Difference: Return value
  - repeat: none
  - for-each: none
  - map: list (of the same length)
- Difference: Order of evaluation
  - for-each: Left to right
  - map: May be any order

- Some exercises to help us think about using them:
Given a list of grades, add 10 to each grade, giving us a new list of grades

- Argument for repeat: Adding 10 to each but it’s to different numbers, so maybe not
- Argument for map: Making a new list (define grades (list 80 90 89 85 123)) (define newgrades (map (l-s + 10) grades))

Make a turtle move forward 1 and make its color darker fifteen times

Code written on the fly

```
(define yertle (turtle-new salamasond))
(repeat 15
  (lambda (turtle)
    (turtle-forward! turtle 1)
    (turtle-set-color! turtle (rgb-darker (turtle-get-color turtle)))
    yertle))
```

```
(define yertle (turtle-new salamasond))
(for-each
  (lambda (color)
    (turtle-forward! yertle 1)
    (turtle-set-color! yertle color))
  (list color1 color2 ... color15))
```

Code after testing in DrRacket (and then a failed cut-and-paste)

```
#lang racket
(require gigls/unsafe)

(define turtle-get-color
  (lambda (turtle)
    (turtle ':color))

(define salamasond (image-show (image-new 200 200)))
(define yertle (turtle-new salamasond))

(turtle-teleport! yertle 100 100)
(for-each
  (lambda (color)
    (turtle-forward! yertle 3)
    (turtle-set-color! yertle color))
  (list "blue" "pink" "red" "orange" "maroon"
    "green" "blue" "indigo" "violet"
    "black" "yellow" "purple" "blue"
    "pink" "grey" "gray" "black"))

(turtle-teleport! yertle 100 130)
(turtle-set-color! yertle (rgb-new 255 128 128))
(repeat 15
  (lambda (turtle)
    (turtle-forward! turtle 5)
    (turtle-set-color! turtle (rgb-darker (turtle-get-color turtle)))
    yertle)
```