CSC 151.02 2013F, Extra Class, Week 4: Whatever You Want to Talk About

- Admin
- Questions and Answers
  - Your questions, my answers
  - My questions, your answers or my answers

Admin

- I will end class at about 2:00 p.m. because my 2:15 class requires some complicated preparation.

Questions and Answers

What should we do to prepare for this Friday's quiz?

- Lists (Wednesday), including `map`
  - Also `iota`, `increment`, `make-list`, etc.
- The color transforms - `rgb-redder`, `rgb-darker`, `rgb-complement`, and the ilk.
- The two basic ways to apply color transforms to images

What are ways to make lists of drawings?

- `(list d1 d2 d3)`
- `(make-list 10 d3)`
  - Warning! All of them are in the same place
- Using `map` and an existing list of drawings

An example of using map and anonyous procs

```
> (map (lambda (x) (* x 30)) (iota 5))
'(0 30 60 90 120)
```

And another

```
> (define example (map drawing-hshift
                       (make-list 20 d1)
                       (map (lambda (x) (* 30 x)) (iota 20))))
```

Syntax-wise, where do we put `map`?

`map` is a procedure that expects a procedure and a list as input, and creates a list as output.

We would therefore use it anywhere that we want a list.

More concretely, before each procedure that you want to apply to a list. So, if we want to hshift then vshift, we write

```
(map drawing-vshift
     (map drawing-hshift
          *list-of-drawings*
          *list-of-horizontal-offsets*)
     *list-of-vertical-offsets*)
```

Do we only use `map` with drawings?

No, we use it with any list.

```
> (map times10 (iota 10))
(0 10 20 30 40 50 60 70 80 90)

> (map image-show
       (map image-new
            (map times10 (map increment (iota 10)))
            (map times20 (map increment (iota 10)))))
```

As we saw, that last command can be dangerous.

Code from Today

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

; Render a drawing
(define render
  (lambda (drawing)
    (image-show (drawing->image drawing 300 200))))

; Render a list of drawings
(define render-list
  (lambda (drawings)
    (render (drawing-compose drawings))))

(define d1
  (drawing-scale (drawing-recolor drawing-unit-circle "red") 50))

(define d2
  (drawing-scale (drawing-recolor drawing-unit-square "grey") 40))

(define d3
  (drawing-vshift
   (drawing-scale
    (drawing-recolor
     drawing-unit-square
     "orange")
    10)
   40))
```

```
(define stuff
  (map drawing-hshift
       (make-list 10 d3)
       (iota 10)))

(define times10
  (lambda (x)
    (* 10 x)))

(define times20
  (lambda (x)
    (* 20 x)))

(define stuff10
  (map drawing-hshift
       (make-list 10 d3)
       (map times10 (iota 10))))


(define stuff20
  (map drawing-hshift
       (make-list 10 d1)
       (map times20 (iota 10))))

(define rainbow
  (map drawing-recolor
       stuff20
       (list "green" "red" "blue" "orange" "pink" "black"
             "purple" "white" "yellow" "grey")))


(define resized-rainbow
  (map drawing-scale
       rainbow
       (reverse (map increment (iota 10)))))

(define concentric
  (map drawing-recolor
       (map drawing-scale
            (make-list 10 d1)
            (reverse (map increment (iota 10))))
       (list "red" "orange" "yellow" "green" "blue" "purple"
             "black" "grey" "white" "black")))

(define concentric-like
  (map drawing-hshift
       concentric
       (map times10 (iota 10))))

(define mod4
  (lambda (x)
    (mod x 4)))

(define more-stuff
  (map drawing-vshift
       stuff20
       (map times10
```

```
            (map mod4 (iota 10)))))

; Things you don't know yet, but that you wanted to learn about
(define alternates
  (apply append (make-list 5 (list d1 d2))))

(define shifty-alternates
  (map drawing-vshift
       alternates
       (map times20 (iota 10))))
```

Samuel A. Rebelsky, rebelsky@grinnell.edu