

Laboratory: Iterating Over Lists

Summary: In this laboratory, you will explore techniques for iterating over lists using the `map` and `foreach!` procedures.

Contents:

- Preparation
- Exercises
 - Exercise 1: Drawing the Figure
 - Exercise 2: Drawing Copies
 - Exercise 3: Drawing Copies, Revisited
 - Exercise 4: Safely Drawing Copies
 - Exercise 5: Complementing Figures
 - Exercise 6: Replicating Figures
- For Those With Extra Time
 - Extra 1: Getting Spots
 - Extra 2: Getting Lots of Spots
 - Extra 3: Mirroring Lists
 - Extra 4: Replicating Figures, Revisited
- Some Useful Definitions

Reference:

Preparation

- a. Create a new 200x200 image and call it `canvas`.
- b. Load an image and call it `portrait`.
- c. Create a list of six to ten spots and call it `figure`. If you're not feeling particularly creative, you can use the following.

```
(define figure
  (list (spot.new 2 0 color.blue)
        (spot.new 0 1 color.blue)
        (spot.new 1 1 color.blue)
        (spot.new 2 1 color.blue)
        (spot.new 3 1 color.blue)
        (spot.new 4 1 color.blue)
        (spot.new 2 2 color.blue)
        (spot.new 1 3 color.blue)
        (spot.new 3 3 color.blue)
        (spot.new 1 4 color.blue)
        (spot.new 3 4 color.blue)))
```

d. Add the definitions of `spot.new`, `spot.col`, `spot.row`, `spot.color`, `spot.draw`, `spot.decadraw`, `spot.htrans`, and `spot.vtrans` to your definitions pane. If you did not define any of those procedures, look at the collection of useful definitions that appears at the end of this lab.

e. Add the following definitions to your definitions pane.

```
(define spot-list.draw
  (lambda (spots image)
    (foreach! (r-s spot.draw image) spots)))
(define spot-list.decadraw
  (lambda (spots image)
    (foreach! (r-s spot.decadraw image) spots)))
```

Exercises

Exercise 1: Drawing the Figure

a. Using `spot-list.draw`, render figure on canvas.

b. Using `spot-list.decadraw`, render figure on canvas.

Exercise 2: Drawing Copies

Recall that we can horizontally translate each spot in a list of spots using `(map (lambda (spot) (spot.htrans spot offset)) spots)` and that you can vertically translate each spot using `(map (lambda (spot) (spot.vtrans spot offset)) spots)`.

a. Confirm that `(map (lambda (spot) (spot.htrans spot 7)) figure)` translates each spot seven spaces to the right. (Execute the instruction and read the results.)

b. Confirm that `(map (lambda (spot) (spot.vtrans spot 11)) figure)` translates each spot eleven spaces down.

c. Draw each of the translated figures using `spot-list.draw`.

Exercise 3: Drawing Copies, Revisited

Recall that when we write lambda expressions that simply fill in one of the parameters to a procedure, as in the cases above, we can substitute a call to `l-s` (to fill in the left parameter) or `r-s` (to fill in the right parameter).

a. Confirm that `(map (r-s spot.htrans 10) figure)` translates each spot ten spaces to the right. (Execute the instruction and read the results.)

b. Confirm that `(map (r-s spot.vtrans 5) figure)` translates each spot down five spaces.

- c. Draw each of the translated figures using `spot-list.draw`.
- d. Draw a copy of `figure` that is translated down 15 and right 20.

Exercise 4: Safely Drawing Copies

- a. What do you expect `(map (r-s spot.htrans -2) figure)` to do?
- b. Check your answer experimentally.
- c. What do you expect the following code to do?

```
(spot-list.draw (map (r-s spot.htrans -2) figure) canvas)
```

- d. Check your answer experimentally.
- e. As you should have discovered, trying to draw the left-translated figure will result in an error, since some of the pixels are outside the boundary of the image. Fix this problem by rewriting `spot.draw` so that if the spot is to be drawn outside of the range of the image, nothing happens. (That is, a spot is not drawn and an error message is not produced.)
- f. Determine whether your revised procedure now lets us draw the portion of the left-translated figure that is still on screen.

Exercise 5: Complementing Figures

- a. Write a procedure, `(spot-list.complement spots)`, that computes a new list of spots by complementing the color of each pixel in `spots`.
- b. Draw a complemented version of `figure`.
- c. Draw a complemented and translated version of `figure`.

Exercise 6: Replicating Figures

Write a procedure, `(spot-list.draw-hcopies figure hoffsets image)`, that, given a list of spots, a list of integers, and an image, draws multiple copies of the figure, each copy offset horizontally by the specified amount.

For example, to draw the figure offset by 0, 8, 15, 23, and 38, we might write

```
(spot-list.draw-hcopies figure (list 0 8 15 23 38) canvas)
```

Hint: You should use `foreach!` to do the repeated drawing.

For Those With Extra Time

Extra 1: Getting Spots

a. Write a procedure, (`image.get-spot image pos`), that gets the spot that corresponds to the given position in the image. That is, if the color at position (col, row) is c , then `image.get-spot` will return $(col\ row\ c)$.

Hint: You will probably need to use `pos.col`, `pos.row`, `image.get-pixel`, and `spot.new`.

b. While `image.get-spot` is useful as is, it will be difficult to map because we need two parameters. Hence, here's a variant that gets a spot from the image named `picture`.

```
(define get-spot-from-picture
  (lambda (pos)
    (image.get-spot picture pos)))
```

Verify that this procedure works correctly.

c. Here's an alternate definition of `get-spot-from-picture` that uses `l-s` rather than `lambda`.

```
(define get-spot-from-picture (l-s image.get-spot picture))
```

Verify that this procedure works correctly.

Extra 2: Getting Lots of Spots

a. Pick a group of nearby interesting positions in `picture` and, by using `map`, grab all of the spots at those positions. Name the result `stamp`. That is, write something like the following.

```
(define stamp
  (map get-spot-from-picture (list (position.new 0 0) (position.new 0 1))))
```

b. Render a copy of the stamp elsewhere in the image.

c. Render another copy of the stamp elsewhere in the image.

Extra 3: Mirroring Lists

a. Write a procedure, (`spot-list.hmirror spots col`), that “mirrors” the figure given by `spots` around the vertical line through `col`. For example, if a spot is at 3,2, and we horizontally mirror that spot around 5, we get a spot at 7,2 with the same color.

b. Write a similar procedure, (`spot-list.vmirror spots row`), that mirrors the figure given by `spots` around the horizontal line through `row`.

Extra 4: Replicating Figures, Revisited

Write a procedure, (`spot-list.draw-replicated figure offsets image`), that, given a list of spots, a list of offsets, and an image, draws multiple copies of the figure, each copy offset by the specified amount. Each offset is a length-two list of the form $(col\ row)$.

For example, to draw the figure at its original position, offset by (10,10), (10,20), (20,30), (30,50), and (50,80), we might write.

```
(spot-list.draw-replicated
  figure
  (list (list 0 0) (list 10 10) (list 10 20) (list 20 30) (list 30 50) (list 50 80))
  canvs)
```

Hint: You'll need to use `foreach!` to do the repeated drawing.

Some Useful Definitions

This lab depends on a variety of procedures that you should have defined in the lab on representing images as lists of spots. However, some of you may have not had time to complete all of those problems and others of you may not be sure of your answers. Here are the definitions we wrote.

```
(define spot.new
  (lambda (col row color)
    (list col row color)))

(define spot.col
  (lambda (spot)
    (car spot)))
(define spot.row
  (lambda (spot)
    (cadr spot)))
(define spot.color
  (lambda (spot)
    (caddr spot)))

(define spot.draw
  (lambda (spot image)
    (image.set-pixel! image (spot.col spot) (spot.row spot) (spot.color spot))))

(define spot.decadraw
  (lambda (spot image)
    (region.compute-pixels!
     image
     (* 10 (spot.col spot)) (* 10 (spot.row spot))
     (+ 9 (* 10 (spot.col spot)) (+ 9 (* 10 (spot.row spot))))
     (lambda (pos) (spot.color spot)))))

(define spot.htrans
  (lambda (spot offset)
    (spot.new (+ offset (spot.col spot)) (spot.row spot) (spot.color spot))))
```

```
(define spot.vtrans  
  (lambda (spot offset)  
    (spot.new (spot.col spot) (+ offset (spot.row spot)) (spot.color spot))))
```

Copyright © 2007 Janet Davis, Matthew Kluber, and Samuel A. Rebelsky. (Selected materials copyright by John David Stone and Henry Walker and used by permission.) This material is based upon work partially supported by the National Science Foundation under Grant No. CCLI-0633090. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/2.5/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.