Laboratory: Boolean Values and Predicate Procedures

Summary: In this laboratory, you will have the opportunity to explore a number of issues relating to predicates, Boolean values, and conditional operations.

Contents:

- Preparation
- Exercises
  - Exercise 1: Combining Boolean Values
  - Exercise 2: Simple Color Predicates
  - Exercise 3: Writing Your Own Predicates
  - Exercise 4: Comparing Colors
  - Exercise 5: Ranges
  - Exercise 6: Exploring and and or
  - Exercise 7: Converting to Black, Grey, or White
- For Those With Extra Time
  - Extra 1: Dominance
  - Extra 2: Transforming to Dominant Colors
- Notes
  - Notes on Problem 6

Preparation

You need not do any preparation for this lab, other than starting DrFu and making sure that you’ve done the reading.

Exercises

Exercise 1: Combining Boolean Values

Experience suggests that students understand and and or much better after a little general practice figuring out how they combine values. Fill in the following tables for each of the operations and and or. The third column of the table should be the value of (and arg1 arg2), where arg1 is the first argument and arg2 is the second argument. The forth column should be the value of (or arg1 arg2).
Exercise 2: Simple Color Predicates

As you may recall, in the reading on Boolean values and predicate procedures, we defined two simple predicates, rgb.light? and rgb.dark?. Here is their code again.

```scheme
;;; Procedure: rgb.light?
;;; Parameters: color, an RGB color
;;; Purpose: Determine if the color seems light.
;;; Produces: light?, a Boolean value
;;; Preconditions: [None]
;;; Postconditions: light? is true (#t) if color’s intensity is relatively high.
;;; light? is false (#f) otherwise.
(define rgb.light? (lambda (color)
                   (<= 192 (+ (* .30 (rgb.red color)) (* .59 (rgb.green color)) (* .11 (rgb.blue color))))))

;;; Procedure: rgb.dark?
;;; Parameters: color, an RGB color
;;; Purpose: Determine if the color seems dark.
;;; Produces: dark?, a Boolean value
;;; Preconditions: [None]
;;; Postconditions: dark? is true (#t) if color’s intensity is relatively low.
;;; dark? is false (#f) otherwise.
(define rgb.dark? (lambda (color)
                    (>= 64 (+ (* .30 (rgb.red color)) (* .59 (rgb.green color)) (* .11 (rgb.blue color))))))
```

a. Test those predicates on a few extreme values, such as black, white, and a grey, to make sure that they work as you might expect.

b. Determine experimentally whether there is a dark color with a blue component of 255.
c. Determine experimentally the largest green component a color can have and still be considered dark.

d. Determine experimentally the smallest green component a color can have and still be considered light.

e. Give instructions that someone else could follow in order to determine the darkest shade of grey that is still considered light. (In writing these instructions, assume that the precise algorithm rgb.lighter? uses is unknown. All you know about the procedure is that if it considers one shade of grey light, then it considers every lighter shade of grey light.)

f. Get instructions from someone else in class and attempt to follow those instructions. (Share your instructions with a neighbor, too.)

**Exercise 3: Writing Your Own Predicates**

a. Write a predicate, (not-very-blue? color), that holds only when the color’s blue component is less than 64.

b. Write a predicate, (red-dominates? color), that holds only if the red component is greater than the sum of the green and the blue components.

c. Write a predicate, (greyish? color), that holds only if no two components of color differ by more than 8.

**Exercise 4: Comparing Colors**

As you’ve noted, the < procedure can be used to determine if one number is smaller than another. Can we do similar comparisons for colors? Certainly. There are, however, a number of different criteria one could use to compare colors.

a. Write a two-parameter predicate, (rgb.greener? color1 color2), that holds only if the green component of color1 is larger than the green component of color2.

b. Write a two-parameter predicate, (rgb.lighter? color1 color2), that holds only if color1 is lighter than color2. Note that in doing this comparison, you should first figure out how light a color is (either by averaging the three components or by using the more complex lightness computation).

**Exercise 5: Ranges**

a. Write a procedure, (valid-component? comp), that determines if the value named by comp is between 0 and 255, inclusive.

b. Test that procedure for different values of comp.
Exercise 6: Exploring and and or

a. Determine the value and returns when called with no parameters.

b. Explain why you think the designers of Scheme had and return that value.

c. Determine the value and returns when called with integers as parameters.

d. Explain why you think the designers of Scheme had and return that value.

e. Determine the value or returns when called with no parameters.

f. Explain why you think the designers of Scheme had or return that value.

g. Determine the value or returns when called with only integers as parameters.

h. Explain why you think the designers of Scheme had or return that value.

If you are puzzled by some of the answers, you may want to look at the \[\text{notes on this problem}\] available at the end of the lab.

Exercise 7: Converting to Black, Grey, or White

The reading included a procedure that used and and or to convert colors to black, grey, or red.

;;; Procedure:
;;;   rgb.bgw
;;; Parameters:
;;;   color, an RGB color
;;; Purpose:
;;;   Convert an RGB color to black, grey, or white, depending on
;;;   the intensity of the color.
;;; Produces:
;;;   bgw, an RGB color
;;; Preconditions:
;;;   rgb.light? and rgb.dark? are defined.
;;; Postconditions:
;;;   If (rgb.light? color) and not (rgb.dark? color), then bgw is white.
;;;   If (rgb.dark? color) and not (rgb.light? color), then bgw is black.
;;;   If neither (rgb.light? color) nor (rgb.dark? color), then bgw is
;;;   grey.
;;;   Otherwise, bgw is one of black, white, or grey.
(define black (rgb.new 0 0 0))
(define white (rgb.new 255 255 255))
(define grey (rgb.new 128 128 128))
(define rgb.bgw
  (lambda (color)
    (or (and (rgb.light? color) white)
        (and (rgb.dark? color) black)
          grey))))
a. Test this procedure by applying it to some colors you know are light, some colors you know are dark, and some colors you know are neither light nor dark. For example,

\[ \texttt{(rgb->string (rgb.bgw (rgb.new 10 10 10)))} \]

b. Open a new image and apply rgb.bgw as a filter to the image. (Use image.map to apply it.)

c. Change the code so that light colors are converted to yellow, dark colors to blue, and other colors to green.

d. Apply this transformation to your image.

**For Those With Extra Time**

**Extra 1: Dominance**

Write three predicates, each of which checks if one component is greater than the others.

- `(mostly-red? color)` holds if the red component of `color` is the largest of the three components.
- `(mostly-green? color)` holds if the green component of `color` is the largest of the three components.
- `(mostly-blue? color)` holds if the blue component of `color` is the largest of the three components.

**Extra 2: Transforming to Dominant Colors**

a. Write a procedure, `(rgb.dominant color)`, similar to rgb.bgw, that, given a color, converts it to red if the color is mostly red, to blue if the color is mostly blue, and to green if the color is mostly green. If the color is none of those, `rgb.dominant` should convert the color to a middle grey.

b. Test this procedure on an image of your choice.

**Notes**

**Notes on Problem 6**

`(and)` has value true (#t) because “and has a value of true if none of the parameters have value false”. Since this call has no parameters, none are false.

Alternately, you can think of #t as the “and-itive identity”. That is, `(and #t x)` is x.

`(or)` has value false (#f) because “or has value false if none of the parameters is non-false”. Since this call has no parameters, none are non-false.
Alternately, you can think of \( \# f \) as the “or-itive identity”. That is, \(( \text{or} \; \# f \; x) \) is \( x \).