Lab 7 Lambda

Purpose: The purpose of this lab is to provide you with some more practice with higher-order functions, and to demonstrate how they can lead very quickly to very fun code.
Textbook References: Chapter 17: Nameless Functions

Goals: Practice using functions as data. Practice designing functions that produce functions. Practice using lambda to create anonymous functions. Practice with world programs.

We will design a game in which you can create many balls which move across the screen in various ways. The player can add new balls by clicking on the screen.

Much of the work here is with higher-order functions. When it comes to higher-order operations, signatures are there to keep you sane and give you a hint as to what's going on.

Sample Problem Provide a data definition for a Ball. A Ball should have a radius, a mode (solid or outline), a color, and a function that takes a Natural Number and produces a Posn indicating where the ball is at that time.

```
; A Ball is a (make-ball Nat Mode Color [Nat -> Posn])
(define-struct ball (r mode color placement))
; - where r is the ball's radius
; - mode is the ball's mode
; - color is the ball's color
; - and placement is a function that, given the current time,
;   outputs a new coordinate for the ball to be drawn at

; A Mode is one of:
; - 'solid
; - 'outline
```

Sample Problem Provide some examples of Balls.

```
(define HEIGHT 500)
(define WIDTH 500)
(define BALL-1 (make-ball 5 'solid 'red (λ (t) (make-
posn 20 (modulo t HEIGHT)))))
(define BALL-2 (make-ball 7 'outline 'blue (λ (t) (make-
posn (modulo t WIDTH) 100))))
```

Sample Problem Write a template for functions that process a Ball.

```
; ball-temp : Ball -> ???
(define (ball-temp b)
  (... (ball-r b) ... (mode-temp (ball-mode b)) ...
       (ball-color b) ... (ball-placement b) ...))

; mode-temp : Mode -> ???
(define (mode-temp m)
  (... (cond [(symbol=? m 'solid) ...]
             [(symbol=? m 'outline) ...]) ...))
```

Sample Problem Provide a data definition for a World. A World needs to keep track of the time (so it knows where the balls should go) and all of its Balls.

```
; A World is a (make-world Nat [List-of Ball])
(define-struct world (t balls))
; - where t is the amount of time that has passed
; - and balls is the balls of the world
```

Sample Problem Provide some examples of Worlds.

```
(define WORLD-1 (make-world 0 '()))
(define WORLD-2 (make-world 10 (list BALL-1 BALL-2)))
```

Sample Problem Write a template for functions that process a World.

```
; world-temp : World -> ???
(define (world-temp w)
  (... (world-t w) ... (ball-list-temp (world-balls w)) ...))

; ball-list-temp : [List-of Ball] -> ???
(define (ball-list-temp alob)
  (... (cond [(empty? alob) ...]
             [(cons? alob)
              ... (ball-temp (first alob)) ...
              ... (ball-list-temp (rest alob)) ...]) ...))
```

Sample Problem Write the main function.

```
; main : [List-of Ball] -> World
; Run this game with this list of initial balls
(define (main init-list)
  (big-bang (make-world 0 init-list)
            [on-tick tick]
            [to-draw draw]
            [on-mouse place-ball]))
```

Exercise 1 Design the function *tick*, which will take a World, and return one with the time incremented by one.

Exercise 2 Design the function *draw-ball* that given a Ball, a Posn, and an Image, draws the ball at that point on that image.

Exercise 3 Design the function *make-drawer* that given a time will create a function that takes a Ball and an Image and will draw it.

Hint: Use draw-ball.

Exercise 4 Design the function *draw* that draws the World. Look at the helpers you've just written: especially the signature for *make-drawer*. What pre-defined list abstraction does it remind you of? How can you use that to help you? May it be a light to you in dark places, when all other lights go out.

We want to be able to add new balls when the user clicks the screen. Here are some data definitions that will help us:

```
; A BallGenerator is a [Nat Nat Nat -> [Nat -> Posn]]
; Given the time, x-coordinate, and y-coordinate of when and where a
; ball is created, create a function that, given the current time of
; the world, will output a Posn

; Example:
; move-horizontally : BallGenerator
(define (move-horizontally t0 x0 y0)
```

```
  (λ (t) (make-posn (modulo (+ x0 (- t t0)) WIDTH) y0)))
(check-expect ((move-horizontally 3 5 8) 10) ; 7 seconds have passed
              (make-posn 12 8))
```

Exercise 5 Design the BallGenerator *move-vertically*.

Exercise 6 Create a constant `GENERATORS`, which is a [List-of BallGenerator] that will keep track of all the BallGenerators you've made thusfar.

Exercise 7 Design the function *place-ball*, that given a World, Nat, Nat, and MouseEvent will output a World with a Ball added to it if the person clicked. Feel free to use any radius. color, or mode you like. As far as the ball's placement: look at the data definition of a Ball, the signature of this function, and the data definition of a BallGenerator. Feel free to use any BallGenerator you like, as long as it's used correctly! Your main function should now run. Try it out!

Exercise 8 Now comes the fun stuff. Design a function *select-random*, that given any non-empty list will select a random element from the list.

Exercise 9 Update *place-ball* to select a random BallGenerator from the `GENERATORS` you previously defined.

Exercise 10 Update *place-ball* to make a radius of random size. Make sure you don't make radiuses of size 0, though, those aren't very fun! For more random fun, use a random Mode, and a random Color (look at the documentation for*make-color*).

Exercise 11 Go crazy. Create as many BallGenerators as you want to and add them to your `GENERATORS`. Have a Ball appear in a totally random place, make it zig-zag, the World is your oyster and WIDTH and HEIGHT are the limit!