UMass Boston Computer Science

CS450 High Level Languages (section 2)

More High-Level Features

Wednesday, September 27, 2023

Logistics

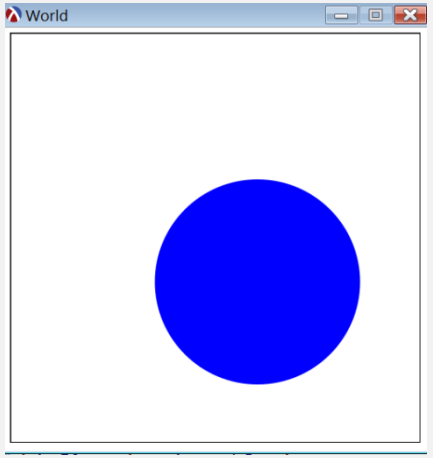
- HW 2 out
  - due: Sun 10/1 11:59 pm EST

- See piazza note about HW2 updates

Last Time

```
;; A Coordinate is a Real
;; Represents x or y position on big-bang canvas
```

```
;; A WorldState is a
(struct world [x y])
;; where:
;; x: Coordinate – represents x coordinate of ball center
;; y: Coordinate – represents y coordinate of ball center
```



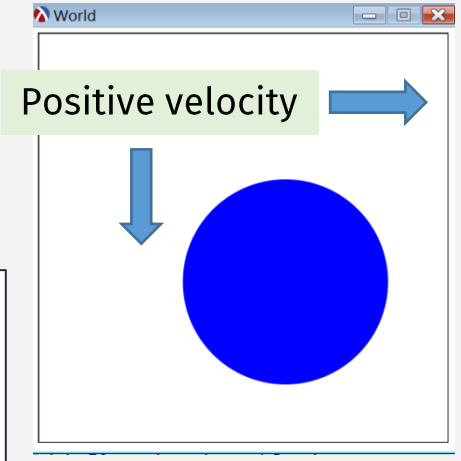

```
;; mouse-handler : WorldState Coordinate Coordinate MouseEvent -> WorldState
;;  Sets the WorldState to be the current mouse location
(define (mouse-handler w x y evt)
  (world x y))
```

Let's allow ball to move on its own

```
(check-equal? (mouse-handler (world 1 1) 0 0 "button-down")
              (world 0 0))
```

```
;; A Velocity is an Int in [0,10)
;; represents pixels/tick
;; positive = down or right
;; negative = up or left
```

```
;; A WorldState is a
(struct world [x y xvel yvel])
;; where:
;; x: Coordinate – represents x coordinate of ball center
;; y: Coordinate – represents y coordinate of ball center
;; xvel: Velocity - in x direction
;; yvel: Velocity – in y direction
```




```
;; WorldState TEMPLATE
;; world-fn : WorldState -> ???

(define (world-fn w)
  .... (world-x w) .... (world-y w) ....
  .... (world-xvel w) .... (world-yvel w) ....)
```

(extract pieces of compound data … to be combined with arithmetic)

```
;; A WorldState is a
(struct world [x y xvel yvel])
;; where:
;; x: Coordinate – represents x coordinate of ball center
;; y: Coordinate – represents y coordinate of ball center
;; xvel: Velocity - in x direction
;; yvel: Velocity – in y direction
```

```
;; next-world : WorldState -> WorldState
;; Computes the next ball pos
(define (next-world w)
  (world
    (+ (world-x w) (world-xvel w))
    (+ (world-y w) (world-yvel w))
    (world-xvel w)
    (world-yvel w)))
```

```
(check-equal?
  (next-world (world 2 2 1 1))
              (world 3 3 1 1)))
```

Add velocity to pos

Repeated code
(not that bad, but let's see some ways to remove it)

let

```
;; A WorldState is a
(struct world [x y xvel yvel])
;; where:
;; x: Coordinate – represents x coordinate of ball center
;; y: Coordinate – represents y coordinate of ball center
;; xvel: Velocity - in x direction
;; yvel: Velocity – in y direction
```

```
;; next-world : WorldState -> WorldState
;; Computes the next ball pos
(define (next-world w)
  (let ([x (world-x w)]
        [y (world-y w)]
        [xvel (world-xvel w)]
        [yvel (world-yvel w)])
    (world (+ x xvel) (+ y yvel) xvel yvel)))
```

Extract all compound data pieces first, before doing “arithmetic”

```
(let ([id val-expr] ...) body ...+)
```

Defines new local variables

in scope only in the body

Local variables **shadow** previously defined vars

Internal **defines** (equiv to **let**)

```
;; A WorldState is a
(struct world [x y xvel yvel])
;; where:
;; x: Coordinate – represents x coordinate of ball center
;; y: Coordinate – represents y coordinate of ball center
;; xvel: Velocity - in x direction
;; yvel: Velocity – in y direction
```

```
;; next-world : WorldState -> WorldState
;; Computes the next ball pos
(define (next-world w)
  (define x (world-x w))
  (define y (world-y w))
  (define xvel (world-xvel w))
  (define yvel (world-yvel w))
  (world (+ x xvel) (+ y yvel) xvel yvel)))
```

Extract all compound data pieces first, before doing “arithmetic”

(is there an easier way to do this?)

Pattern Matching!

```
;; A WorldState is a
(struct world [x y xvel yvel])
;; where:
;; x: Coordinate – represents x coordinate of ball center
;; y: Coordinate – represents y coordinate of ball center
;; xvel: Velocity - in x direction
;; yvel: Velocity – in y direction
```

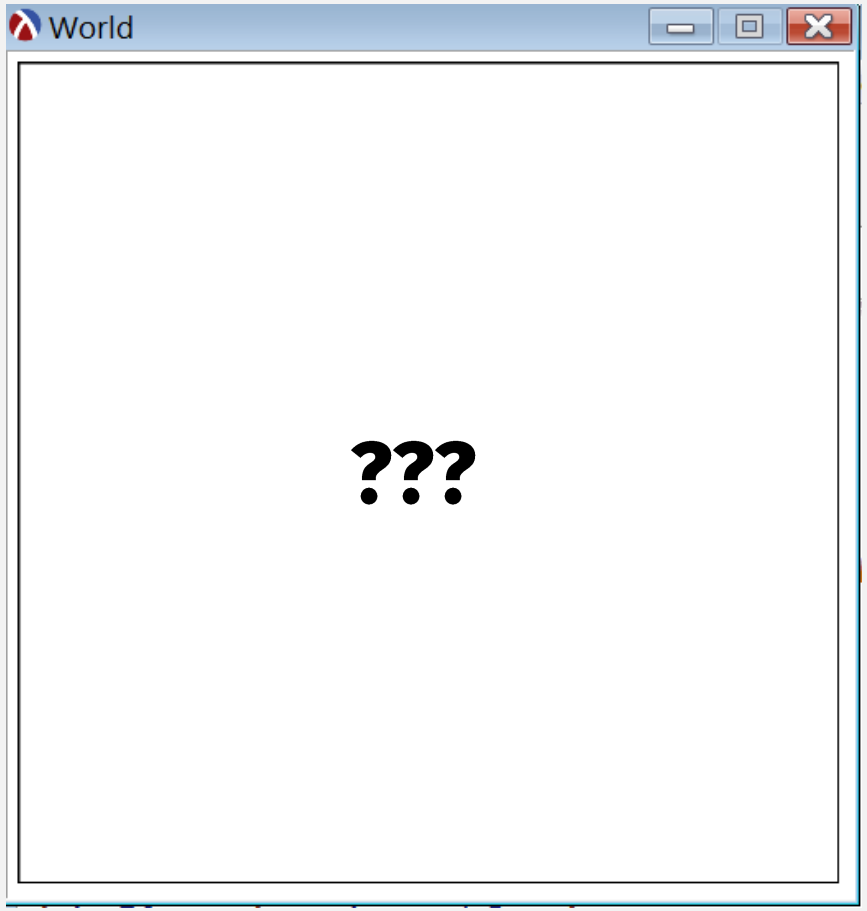
```
;; next-world : WorldState -> WorldState
;; Computes the next ball pos
(define (next-world w)
  (match-define (world x y xvel yvel) w)



  (world (+ x xvel) (+ y yvel) xvel yvel)))
```

Extract all compound data pieces, at the same time!

Let’s see what our animation looks like …



Make it bounce?

```
;; A WorldState is a
(struct world [x y xvel yvel])
;; where:
;; x: Coordinate – represents x coordinate of ball center
;; y: Coordinate – represents y coordinate of ball center
;; xvel: Velocity - in x direction
;; yvel: Velocity – in y direction
```

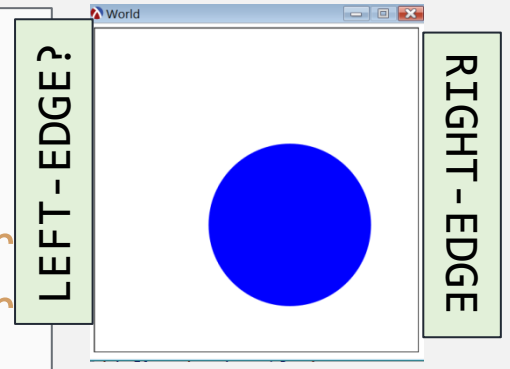
```
;; next-world : WorldState -> WorldState
;; Computes the next ball pos
(define (next-world w)
  (match-define (world x y xvel yvel) w)



  (world (+ x xvel) (+ y yvel) xvel yvel)))
```

Make it bounce?

```
;; A WorldState is a
(struct world [x y xvel yvel])
;; where:
;; x: Coordinate – represents x coordinate of ball center
;; y: Coordinate – represents y coordinate of ball center
;; xvel: Velocity - in x direction
;; yvel: Velocity – in y direction
```

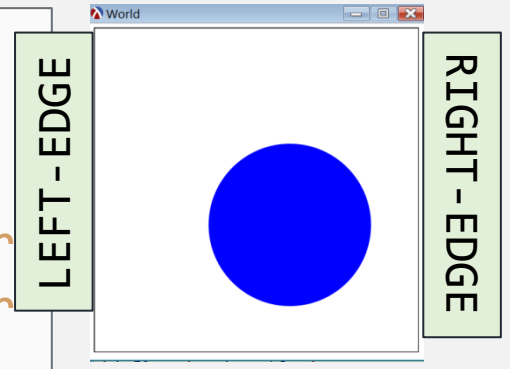



```
;; next-world : WorldState -> WorldState
;; Computes the next ball pos
(define (next-world w)
  (match-define (world x y xvel yvel) w)
  (define new-xvel
    (if (>= x RIGHT-EDGE) (- xvel) xvel))

  (world (+ x xvel) (+ y yvel) new-xvel yvel)))
```

Make it bounce?

```
;; A WorldState is a
(struct world [x y xvel yvel])
;; where:
;; x: Coordinate – represents x coordinate of ball center
;; y: Coordinate – represents y coordinate of ball center
;; xvel: Velocity - in x direction
;; yvel: Velocity – in y direction
```




```
;; next-world : WorldState -> WorldState
;; Computes the next ball pos
(define (next-world w)
  (match-define (world x y xvel yvel) w)
  (define new-xvel
    (if (or (>= x RIGHT-EDGE)
            (<= x LEFT-EDGE)) (- xvel) xvel)
  (world (+ x xvel) (+ y yvel) new-xvel yvel)))
```

Make it bounce?

```
;; A WorldState is a
(struct world [x y xvel yvel])
;; where:
;; x: Coordinate – represents x coordinate of ball center
;; y: Coordinate – represents y coordinate of ball center
;; xvel: Velocity - in x direction
;; yvel: Velocity – in y direction
```

```
;; next-world : WorldState -> WorldState
;; Computes the next ball pos
(define (next-world w)
  (match-define (world x y xvel yvel) w)
  (define new-xvel
    (if (or (>= x RIGHT-EDGE)
            (<= x LEFT-EDGE)) (- xvel) xvel)
  (world (+ x new-xvel) (+ y yvel) new-xvel yvel)))
```

Should this be **xvel** or **new-xvel**???

Make it bounce?

```
;; A WorldState is a
(struct world [x y xvel yvel])
;; where:
;; x: Coordinate – represents x coordinate of ball center
;; y: Coordinate – represents y coordinate of ball
;; xvel: Velocity - in x direction
;; yvel: Velocity – in y direction
```

If you're **no longer following the template**, then the **Data Definitions need updating!**

```
;; next-world : WorldState -> WorldState
;; Computes the next ball
(define (next-world w)
  (match-define (world x
  (define new-xvel
    (if (or (>= x RIGHT-EDGE)
            (<= x LEFT-EDGE
  (define new-yvel???
    (if (or (>= y BOTTOM-EDGE)
```

Keep hacking and hope that it works???

DON'T PROGRAM LIKE THIS!!!

This is <u>undisciplined</u> programming and is **much** <u>slower and error-prone</u> than thinking first!

Program Design Recipe

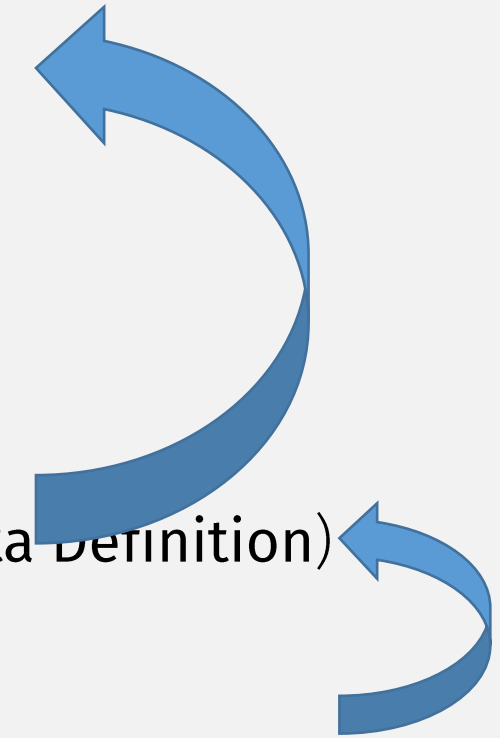
… is **iterative!**

1. **Data Design**
2. **Function Design**

Function Design Recipe

… is iterative!

1. **Name**
2. **Signature** – types of the **function input(s)** and **output**
3. **Description** – explain (in **English prose**) the **function behavior**
4. **Examples** – show (using `rackunit`) the **function behavior**
5. **Template** – sketch out the **function structure** (using input's **Data Definition**)
6. **Code** – implement the **rest of the function** (arithmetic)
7. **Tests** – check (using `rackunit`) the **function behavior**



Make it bounce?

```
;; A WorldState is a
(struct world [x y xvel yvel])
;; where:
;; x: Coordinate – represents x coordinate of ball center
;; y: Coordinate – represents y coordinate of ball
;; xvel: Velocity - in x direction
;; yvel: Velocity – in y direction
```

If you're **no longer following the template**, then the **Data Definitions need updating!**

```
;; next-world : WorldState -> WorldState
;; Computes the next ball pos
(define (next-world w)
  (match-define (world x y xvel yvel) w)
  (define new-xvel
    (if (or (>= x RIGHT-EDGE)
            (<= x LEFT-EDGE)) (- xvel) xvel)
  (define new-yvel???
    (if (or (>= y BOTTOM-EDGE)
```

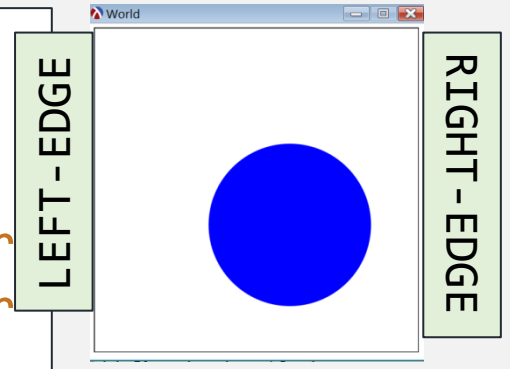
DON'T PROGRAM LIKE THIS!!!

Ma

```
;; A Coordinate is a Real
;; Represents x or y position on big-bang canvas
```

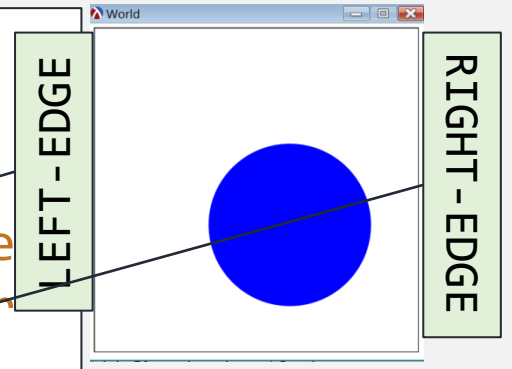
Seems like we want some **intervals**

```
;; A WorldState is a
(struct world [x y xvel yvel])
;; where:
;; x: Coordinate - represents x coordinate of ball center
;; y: Coordinate - represents y coordinate of ball center
;; xvel: Velocity - in x direction
;; yvel: Velocity - in y direction
```



Adding Intervals

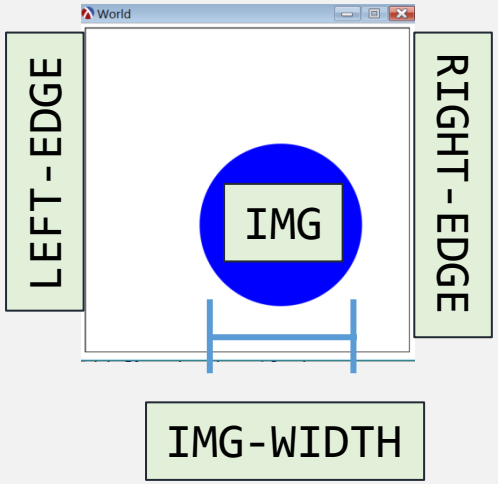
```
;; A WorldState is a
(struct world [x y xvel yvel])
;; where:
;; x: XCoordinate – represents x coordinate of ball cente
;; y: Coordinate – represents y coordinate of ball center
;; xvel: Velocity - in x direction
;; yvel: Velocity – in y direction
```




```
;; An XCoordinate is a real number in one of these intervals:
;; (LEFT-EDGE, RIGHT-EDGE) : image fully within scene
;; (-infinity, LEFT-EDGE]  : (at least) part of image out of scene, to the left
;; [RIGHT-EDGE, +infinity) : (at least) part of imageout of scene, to the right
;; Interp: The coordinate is the x coordinate of image center;
;; the intervals represent whether the image is fully within
```

WAIT! Is this correct?

Adding Intervals

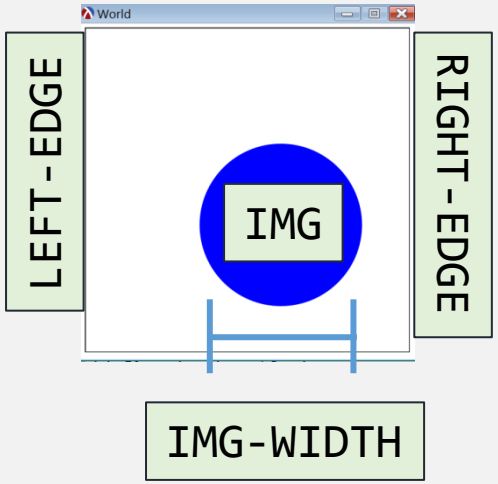



```
;; An XCoordinate is a real number in one of these intervals:

;; ( LEFT-EDGE + IMG-WIDTH/2, RIGHT-EDGE - IMG-WIDTH/2) : image fully within scene
;; (-inf, LEFT-EDGE + IMG-WIDTH/2]    : (part of) image out of scene, to the left
;; [RIGHT-EDGE - IMG-WIDTH/2, +inf)   : (part of) image out of scene, to the right
```

```
;; TEMPLATE???
```

Adding Intervals

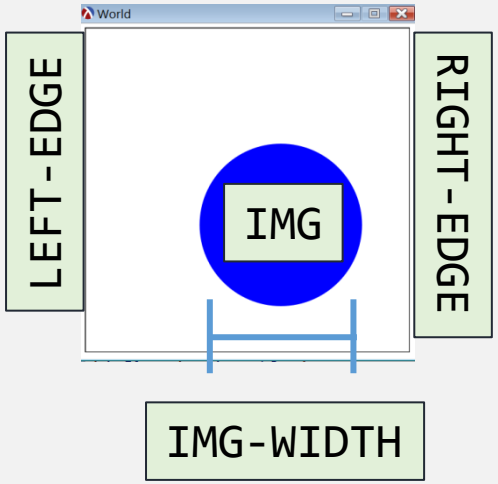


Now the **shape of the function** matches the **shape of the data definition**!

```
;; An XCoordinate is a real number in one of these intervals:
;; ( LEFT-EDGE + IMG-WIDTH/2, RIGHT-EDGE - IMG-WIDTH/2) : image fully within scene
;; (-inf, LEFT-EDGE + IMG-WIDTH/2]    : (part of) image out of scene, to the left
;; [RIGHT-EDGE - IMG-WIDTH/2, +inf)   : (part of) image out of scene, to the right

;; TEMPLATE
(define (x-fn x)
  (cond [(< (/ IMG-WIDTH 2) x (- RIGHT-EDGE (/ IMG-WIDTH 2))) ....]
        [(<= x (/ IMG-WIDTH 2)) ....]
        [(>= x (- RIGHT-EDGE (/ IMG-WIDTH 2))) ....]))
```

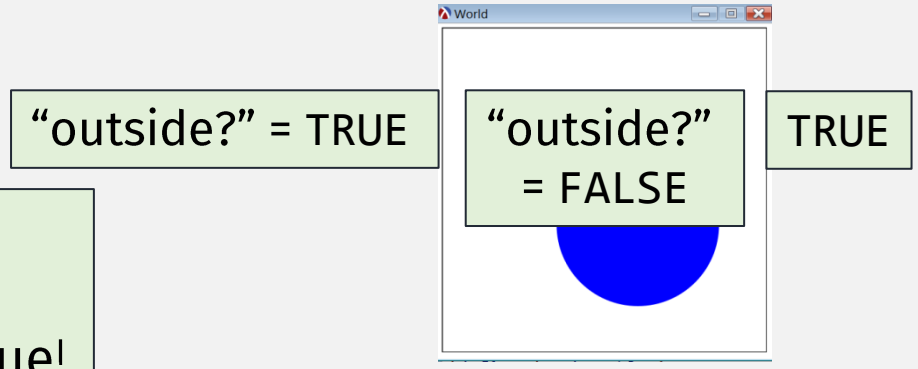
Adding Intervals




```
;; An XCoordinate is a real number in one of these intervals:
;; ( LEFT-EDGE + IMG-WIDTH/2, RIGHT-EDGE – IMG-WIDTH/2) : image fully within scene
;; (-inf, LEFT-EDGE + IMG-WIDTH/2]    : (part of) image out of scene, to the left
;; [RIGHT-EDGE – IMG-WIDTH/2, +inf)   : (part of) image out of scene, to the right
```

```
;; outside-L/R-edges? : XCoordinate -> Bool
(define (outside-L/R-edges? x)
  (cond [(< (/ IMG-WIDTH 2) x (- RIGHT-EDGE (/ IMG-WIDTH 2))) ....]
        [(<= x (/ IMG-WIDTH 2)) ....]
        [(>= x (- RIGHT-EDGE (/ IMG-WIDTH 2))) ....]))
```

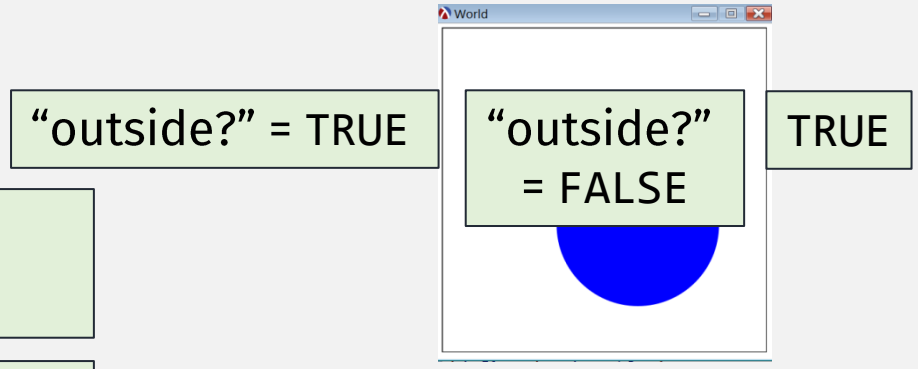
Adding Intervals



A **cond that evaluates to a boolean** is slightly awkward …
Because the **tests already compute the correct value**!

```
;; outside-L/R-edges? : XCoordinate -> Bool
(define (outside-L/R-edges? x)
  (cond [(< (/ IMG-WIDTH 2) x (- RIGHT-EDGE (/ IMG-WIDTH 2))) #false]
        [(<= x (/ IMG-WIDTH 2)) #true]
        [(>= x (- RIGHT-EDGE (/ IMG-WIDTH 2))) #true]))
```

Adding Intervals



A **cond that evaluates to a boolean** is slightly awkward …

Instead, use **`or`** and **just keep true cases**!

```
;; outside-L/R-edges? : XCoordinate -> Bool
(define (outside-L/R-edges? x)
  (or (<= x (/ IMG-WIDTH 2))
      (>= x (- RIGHT-EDGE (/ IMG-WIDTH 2)))))
```

Make it bounce?

```
;; A WorldState is a
(struct world [x y xvel yvel])
;; where:
;; x: Coordinate – represents x coordinate of ball center
;; y: Coordinate – represents y coordinate of ball center
;; xvel: Velocity - in x direction
;; yvel: Velocity – in y direction
```

```
;; next-world : WorldState -> WorldState
;; Computes the next ball pos
(define (next-world w)
  (match-define (world x y xvel yvel) w)
  (define new-xvel
    (if (or (>= x RIGHT-EDGE)
            (<= x LEFT-EDGE)) (- xvel) xvel)
  (define new-yvel???
    (if (or (>= y BOTTOM-EDGE)
```

DON'T PROGRAM LIKE THIS!!!

Computing new velocity

```
;; next-xvel : Xcoordinate Velocity -> Velocity
;; Computes a (possibly) new velocity, based on x position
```

```
(define (next-xvel x xvel)
  (if (outside-L/R-edges? x)
      (- xvel)
      xvel)))
```

flips

No flip

```
(check-equal? (next-xvel LEFT-EDGE -10) 10)
```

flips

```
(check-equal? (next-xvel RIGHT-EDGE 10) -10)
```

flips

```
(check-equal? (next-xvel (sub1 RIGHT-EDGE) 10) 10)
```

No flip

Make it bounce?

```
;; A WorldState is a
(struct world [x y xvel yvel])
;; where:
;; x: Coordinate – represents x coordinate of ball center
;; y: Coordinate – represents y coordinate of ball center
;; xvel: Velocity - in x direction
;; yvel: Velocity – in y direction
```

```
;; next-world : WorldState -> WorldState
;; Computes the next ball pos
(define (next-world w)
  (match-define (world x y xvel yvel) w)
  (define new-xvel
    (if (or (>= x RIGHT-EDGE)
            (<= x LEFT-EDGE)) (- xvel) xvel)
  (define new-yvel???
    (if (or (>= y BOTTOM-EDGE)
```

DON'T PROGRAM LIKE THIS!!!

Make it bounce?

```
;; A WorldState is a
(struct world [x y xvel yvel])
;; where:
;; x: Coordinate – represents x coordinate of ball center
;; y: Coordinate – represents y coordinate of ball center
;; xvel: Velocity - in x direction
;; yvel: Velocity – in y direction
```

```
;; next-world : WorldState -> WorldState
;; Computes the next ball pos
```

```
(define (next-world w)
  (match-define (world x y xvel yvel) w)
  (define new-xvel (next-xvel x xvel))
  (define new-yvel (next-yvel y yvel))
  (world (+ x new-xvel) (+ y new-yvel) new-xvel new-yvel)))
```

Does it work?

Previously

Kinds of Data Definitions

- **Basic data**
  - E.g., **numbers, strings,** etc
- **Intervals**
  - Data that is from a **range of values,** e.g., $[0, 100)$
- **Enumerations**
  - Data that is **one of a list of possible values**, e.g., `"green"`, `"red"`, `"yellow"`
- **Itemizations**
  - Data value that can be from a **list of possible other data definitions**
  - E.g., **either** a **string** or **number** (Generalizes enumerations)

Previously

Kinds of Data Definitions

- Basic data
  - E.g., **numbers, strings,** etc
- Intervals
  - Data that is from a **range of values,** e.g., $[0, 100)$
- Enumerations
  - Data that is **one of a list of possible values**, e.g., `"green"`, `"red"`, `"yellow"`
- Itemizations
  - Data value that can be from a **list of possible other data definitions**
  - E.g., **either** a **string** or **number** (Generalizes enumerations)
- **Compound Data**
  - Data that is a **combination of values from other data definitions**

Last time

Multi-ball Animation

Design a **`big-bang`** animation that:

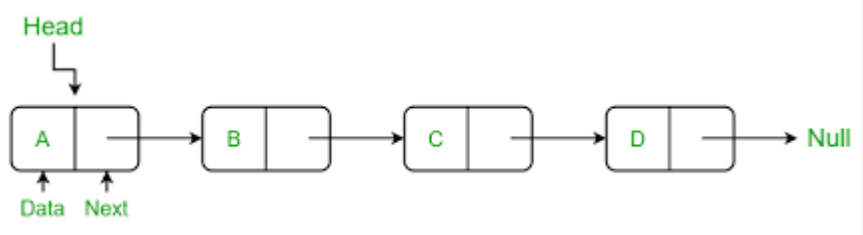
- Start: **a single ball, moving with random x and y velocity**
- On a click: **add a ball at random location, with random velocity**
- If any ball "hits" an edge:
  - if it's a **vertical** edge, **the x velocity should flip direction**
  - If it's a **horizontal** edge, **the y velocity should flip direction**

```
;; A WorldState is … an unknown number of balls!
```

Arbitrary Size Data - Lists

In C

```
struct node
{ int data;
  struct node *next; } *head;
```



This is a **self-referential** (i.e., **recursive**!) definition!

Racket List Data Definition Example

```
;; A ListofInts is one of
;; - null
;; - (cons Int ListofInts)
```

Empty (base) case

Non-empty case

"node"

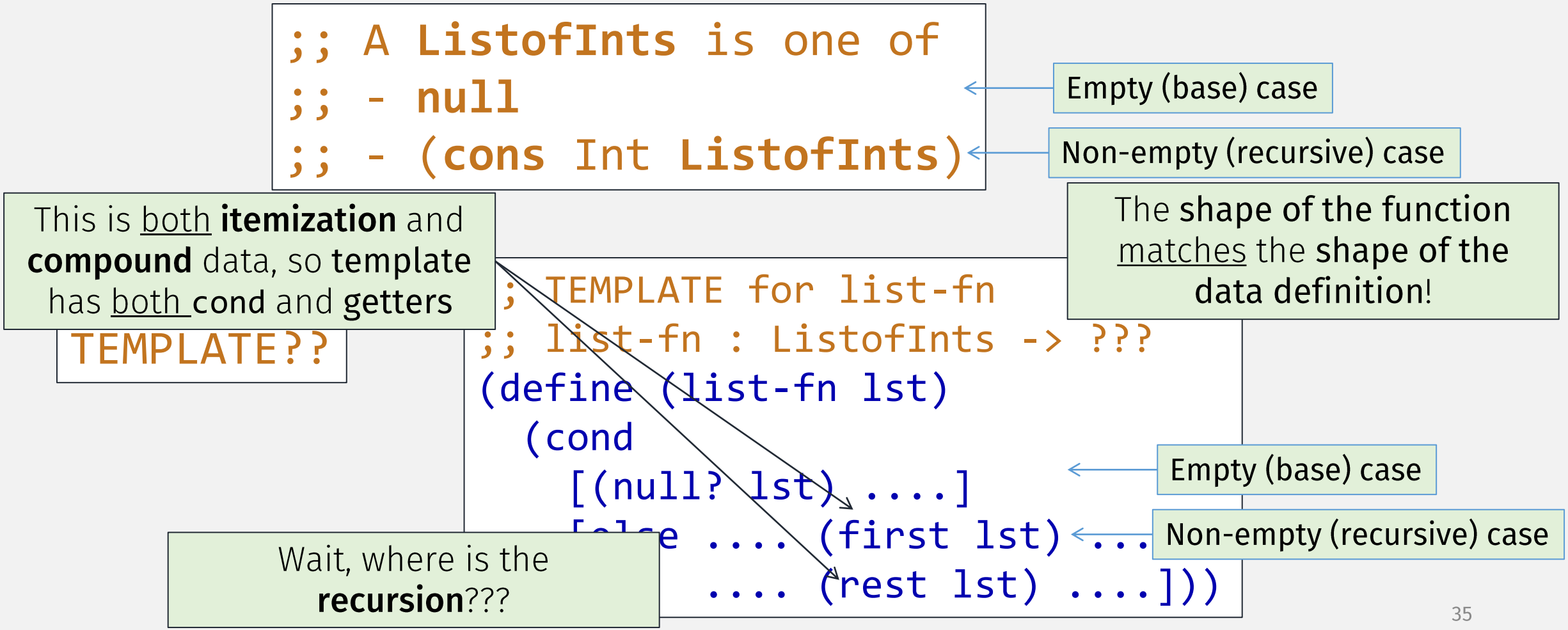
Recursive!

TEMPLATE??

(how can we **use a list of ints to define a list of ints**?!?)

Recursion is only valid if there is <u>both</u>
- A **base case**
- A **recursive case**

Racket List Data Definition Example



Racket List Data Definition Example

```
;; A ListofInts is one of
;; - null
;; - (cons Int ListofInts)
```

TEMPLATE??

... is also recursive!

```
;; TEMPLATE for list-fn
;; list-fn : ListofInts -> ???
(define (list-fn lst)
  (cond
    [(null? lst) ....]
    [else .... (first lst) ....
     .... (list-fn ... (rest lst) ...) ....]))
```

The shape of the function matches the shape of the data definition!

This means that recursion in the data definition ... means recursion in the (template) function that processes that data!

Multi-ball Animation

Design a **`big-bang`** animation that:

- Start: **a single ball, moving with random x and y velocity**
- On a click: **add a ball at random location, with random velocity**
- If any ball "hits" an edge:
  - if it's a **vertical** edge, **the x velocity should flip direction**
  - If it's a **horizontal** edge, **the y velocity should flip direction**

```
;; A WorldState is … an unknown number of balls!
```

```
;; A WorldState is … a list of balls!
```

Check-In Quiz 9/27

on gradescope

(due 1 minute before midnight)