List of Numbers

; A list-of-num is either
;   - empty
;   - (make-bigger-list num list-of-num)
(define-struct bigger-list (first rest))
List of Numbers

; A list-of-num is either
;   - empty
;   - (make-bigger-list num list-of-num)
(define-struct bigger-list (first rest))

Generic template:
; func-for-lon : list-of-num -> ...
(define (func-for-lon l)
  ...)

List of Numbers

; A list-of-num is either
;   - empty
;   - (make-bigger-list num list-of-num)
(define-struct bigger-list (first rest))

Generic template:
; func-for-lon : list-of-num -> ...
(define (func-for-lon l)
  (cond
    [(empty? l) ...]
    [(bigger-list? l) ...])))
List of Numbers

; A list-of-num is either
; - empty
; - (make-bigger-list num list-of-num)
(define-struct bigger-list (first rest))

Generic template:
; func-for-lon : list-of-num -> ...
(define (func-for-lon l)
  (cond
   [(empty? l) ...]
   [(bigger-list? l)
    ... (bigger-list-first l)
    ... (bigger-list-rest l)
    ...])])
List of Numbers

; A list-of-num is either
; - empty
; - (make-bigger-list num list-of-num)
(define-struct bigger-list (first rest))

Generic template:

; func-for-lon : list-of-num -> ...
(define (func-for-lon l)
  (cond
    [(empty? l) ...]
    [(bigger-list? l)
      ... (bigger-list-first l)
      ... (bigger-list-rest l)
      ...]))
List of Numbers

; A list-of-num is either
; - empty
; - (make-bigger-list num list-of-num)
(define-struct bigger-list (first rest))

Generic template:

; func-for-lon : list-of-num -> ...
(define (func-for-lon l)
  (cond
   [(empty? l) ...]
   [(bigger-list? l)
     ... (bigger-list-first l)
     ... (func-for-lon (bigger-list-rest l))
     ...]))
Aquarium Weight

; aq-weight : list-of-num -> num
; Sums the fish weights in l
(define (aq-weight l)
  ... )
Aquarium Weight

; aq-weight : list-of-num -> num
; Sums the fish weights in l
(define (aq-weight l)
  ...)

(check-expect (aq-weight empty) 0)
Aquarium Weight

; aq-weight : list-of-num -> num
; Sums the fish weights in l
(define (aq-weight l)
  ...
)

(check-expect (aq-weight empty) 0)
(check-expect (aq-weight (make-bigger-list 2 empty)) 2)
Aquarium Weight

; aq-weight : list-of-num -> num
; Sums the fish weights in l
(define (aq-weight l)
  ...)

(check-expect (aq-weight empty) 0)
(check-expect (aq-weight (make-bigger-list 2 empty)) 2)
(check-expect (aq-weight (make-bigger-list 5 (make-bigger-list 2 empty))) 7)
Aquarium Weight

; aq-weight : list-of-num -> num
; Sums the fish weights in l
(define (aq-weight l)
  (cond
   [(empty? l) ...]
   [(bigger-list? l)
     ... (bigger-list-first l)
     ... (aq-weight (bigger-list-rest l))
     ...]])

(check-expect (aq-weight empty) 0)

(check-expect (aq-weight (make-bigger-list 2 empty)) 2)

(check-expect (aq-weight (make-bigger-list 5 (make-bigger-list 2 empty))) 7)
Aquarium Weight

; aq-weight : list-of-num -> num
; Sums the fish weights in l
(define (aq-weight l)
  (cond
   [(empty? l) 0]
   [(bigger-list? l)
     (+ (bigger-list-first l)
        (aq-weight (bigger-list-rest l))))])

(check-expect (aq-weight empty) 0)
(check-expect (aq-weight (make-bigger-list 2 empty)) 2)
(check-expect (aq-weight (make-bigger-list 5 (make-bigger-list 2 empty))) 7)
Aquarium Weight

; aq-weight : list-of-num -> num
; Sums the fish weights in l
(define (aq-weight l)
  (cond
   [(empty? l) 0]
   [(bigger-list? l)
    (+ (bigger-list-first l)
      (aq-weight (bigger-list-rest l)))]))

Try examples in the stepper

(check-expect (aq-weight empty) 0)

(check-expect (aq-weight (make-bigger-list 2 empty)) 2)

(check-expect (aq-weight (make-bigger-list 5 (make-bigger-list 2 empty))) 7)
Design Recipe for Lists

Design recipe changes for today:

None

Granted, the self-reference was slightly novel...

; A list-of-num is either
;  - empty
;  - (make-bigger-list num list-of-num)
Recursion

A self-reference in a data definition leads to a **recursive** function—one that calls itself

```scheme
(define (aq-weight l)
  (cond
    [(empty? l) 0]
    [(bigger-list? l)
      (+ (bigger-list-first l)
         (aq-weight (bigger-list-rest l)))]))
```