Aquarium

Our zoo was so successful, let’s start an aquarium

For a fish, we only care about its weight, so for two fish:

; An aquarium is
; (make-aq num num)
(define-struct aq (first second))
Aquarium Template

; An aquarium is
; (make-aq num num)

Generic template:
; func-for-aq : aquarium -> ...
; (define (func-for-aq a)
;    ... (aq-first a) ... (aq-second a) ...)

; aq-weight : aquarium -> num
(define (aq-weight a)
    (+ (aq-first a) (aq-second a)))

(check-expect (aq-weight (make-aq 7 8)) 15)

And so on, for many other simple aquarium functions...
Tragedy Strikes the Aquarium

Poor blue fish... now we have only one

Worse, we have to re-write all our functions...

; An aquarium is
; (make-aq num)
(define-struct aq (first))
Aquarium Template, Revised

; An aquarium is
; (make-aq num)

; func-for-aq : aquarium -> ...
; (define (func-for-aq a)
; ... (aq-first a) ...)

; aq-weight : aquarium -> num
(define (aq-weight a)
  (aq-first a))

(check-expect (aq-weight (make-aq 7)) 7)

And so on, for all of the aquarium functions...
The Aquarium Expands

Hooray, we have two new fish!

Unfortunately, we have to re-re-write all our functions...

; An aquarium is
; (make-aq num num num num)
(define-struct aq (first second third))
A Flexible Aquarium Representation

Our data choice isn’t working

- An aquarium isn’t just 1 fish, 2 fish, or 100 fish—it’s a collection containing an arbitrary number of fish
- No data definition with just 1, 2, or 100 numbers will work

To represent an aquarium, we need a list of numbers

We don’t need anything new in the language, just a new idea
Structs as Boxes

Pictorially,

• `define-struct` lets us define a new kind of box

• The box can have as many compartments as we want, but we have to pick how many, once and for all

```
(define-struct snake (name weight food))
```

⇒

```
```

```
(define-struct ant (weight loc))
```

⇒

```
```

```
```
Boxes Stretch

The boxes stretch to fit any one thing in each slot:

"Slinky" 12 "rats"

Even other boxes:

0.002 2 3

Still, the number of slots is fixed
Packing Boxes

Suppose that

• You have four things to pack as one
• You only have 2-slot boxes
• Every slot must contain exactly one thing

How can you create a single package?
Packing Boxes

This isn’t good enough

because it’s still two boxes...

But this works!
Packing Boxes

And here’s 8 fish:

```
  |   |   |
  |   |   |
  |   |   |
  |   |   |
```

And here’s 16 fish!

```
  |   |   |   |   |
  |   |   |   |   |
  |   |   |   |   |
  |   |   |   |   |
  |   |   |   |   |
  |   |   |   |   |
  |   |   |   |   |
  |   |   |   |   |
```

But what if we just add 1 fish, instead of doubling the fish?

But what if we have 0 fish?
General Strategy for Packing Boxes

Here’s a general strategy:

- For 0 fish, use empty
- If you have a package and a new fish, put them together

To combine many fish, start with empty and add fish one at a time
General Strategy for a List of Numbers

To represent the aquarium as a list of numbers, use the same idea:

• For 0 fish, use \texttt{empty}

• If you have a list and a number, put them together with \texttt{make-bigger-list}

\begin{verbatim}
     empty

     (make-bigger-list 10 empty)

     (make-bigger-list 5 (make-bigger-list 10 empty))

     (make-bigger-list 7 (make-bigger-list 5 (make-bigger-list 10 empty)))
\end{verbatim}