# Non-Empty Lists

CS 5010 Program Design Paradigms "Bootcamp"

Lesson 4.4

# Empty lists

- Most computations on lists make sense on empty lists
  - **`(sum empty) = 0`**
  - **`(product empty) = 1`**
  - **`(double-all empty) = empty`**
  - etc, etc.

# Non-empty lists

- But some computations don't make sense for empty lists
  - min, max
  - average

# Non-Empty Lists

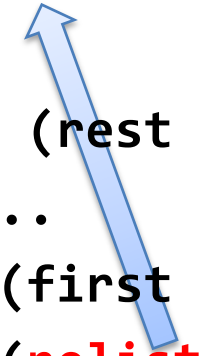
- For these problems, the list template doesn't make sense, either.
- For these problems, we use a different data definition and a different template that is tuned for dealing with lists that are always non-empty.

# Data Definition for Non-Empty List

```
;; A NonEmptyListOfSardines is one of
;; -- (cons Sardine empty)
;; -- (cons Sardine
;;               NonEmptyListOfSardines)
```

# Template for Non-Empty List

```
;; nelist-fn : NonEmptyListOfSardines -> ??
(define (nelist-fn ne-lst)
  (cond
    [(empty? (rest ne-lst)) (... (first ne-lst))]
    [else (...
           (first ne-lst)
           (nelist-fn (rest ne-lst)))]))
```



**`(rest ne-lst)`** is a **NonEmptyListOfSardines** so call **`nelist-fn`** on it

# Template Questions for Non-Empty Lists

```
;; nelist-fn : NonEmptyListOfSardines -> ??
(define (list-fn ne-lst)
  (cond
    [(empty? (rest ne-lst)) (... (first ne-lst))]
    [else (...
           (first ne-lst)
           (list-fn (rest ne-lst)))]))
```

If we knew the answer for the rest of the list, and we knew the first of the list, how could we combine them to get the answer for the whole list?

What is the answer for a list of length 1?

# Non-Empty Lists: The General Pattern

```
A NonEmptyListOfX is one of
-- (cons X empty)
   interp: a list with a single X
-- (cons X NonEmptyListOfX)
   interp: (cons x lst) represents a
   sequence whose first element is x
   and whose other elements are
   represented by lst.
```

# Template Questions for Non-Empty Lists

```
;; nelist-fn : NonEmptyListOfX -> ??
(define (list-fn ne-lst)
  (cond
    [(empty? (rest ne-lst)) (.... (first ne-lst))]
    [else (...
           (first ne-lst)
           (list-fn (rest ne-lst)))]))
```

If we knew the answer for the rest of the list, and we knew the first of the list, how could we combine them to get the answer for the whole list?

What is the answer for a list of length 1?

# Example: max

```
;; list-max : NonEmptyListOfInteger -> Integer
;; GIVEN: a non-empty list of integers,
;; RETURNS: the largest element of the list
(define (list-max ne-lst)
  (cond
    [(empty? (rest ne-lst)) (first ne-lst)]
    [else (max
           (first ne-lst)
           (list-max (rest ne-lst)))]))
```

# Example: average

```
lon-avg : LON -> Number
Given a non-empty LON, returns its average
(lon-avg (cons 11 empty)) = 11
(lon-avg (cons 33 (cons 11 empty))) = 22
(lon-avg (cons 33 (cons 11 (cons 11 empty)))) = 55/3
```

# Example: average

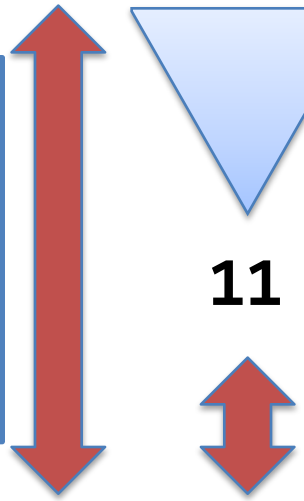
```
;; lon-avg : NELON -> Number
;; Given a non-empty LON, returns its average
;; strategy: structural decomposition
(define (lon-avg ne-lst)
  (cond
    [(empty? (rest ne-lst)) (first ne-lst)]
    [else (....
           (first ne-lst)
           (lon-avg (rest ne-lst)))]))
```

If we knew the answer for the rest of the list, and we knew the first of the list, how could we combine them to get the answer for the whole list?

# Oops...

- **`(lon-avg (list 33 11 11))`** = 55/3

Here are two lists. They have the same first element (33), and the average of their rests is the same (11). But they have different averages. So there's no way to combine 33 and 11 that will give the right answer for both examples. So simply using the template can't possibly work.



➔ **`(... 33 11)`** = 55/3

**11**

- **`(lon-avg (list 33 11))`** = 22

➔ **`(... 33 11)`** = 22

- Can't have both!

# Try something simpler!

```
lon-avg : NELON -> Number
Given a non-empty LON, returns its average
Strategy: combine simpler functions
(define (lon-avg lst)
  (/ (lon-sum lst) (lon-length lst)))
```

Here we had a problem that could not be solved by blindly following the template. But we could still solve it by dividing it into simpler pieces and combining the answers for the pieces.

# Another way of defining non-empty lists

```
A NonEmptyListOfX is a
  (cons X ListOfX)
```

# When to use this one?

- Use this one when the first element of the list needs to be treated specially.
- This one is most often useful with a help function that takes an X and a ListOfX's.
- We'll see this again in Module 7 when we talk about accumulators and generalizing with invarariants.

# Remember, don't use non-empty lists unless you really need to

- The vast majority of problems make sense for the empty list.
- Make your data definitions in the form ListOfX if that make sense (even if the list in the problem never happens to be empty).
- If you're using a NonEmptyListOfX template, and you have duplicated code, that's a sign that it should be a plain old ListOfX.

# Summary

- You should now be able to explain the difference between a list of items and a non-empty list of items
- You should be able to write down the template for a non-empty list and use it.

# Next Steps

- If you have questions about this lesson, ask them on the Discussion Board
- Do Problem Set 04