

Non-Empty Lists

CS 5010 Program Design Paradigms
“Bootcamp”
Lesson 4.4



© Mitchell Wand, 2012-2014

This work is licensed under a [Creative Commons Attribution-NonCommercial 4.0 International License](https://creativecommons.org/licenses/by-nc/4.0/).

Lesson Introduction

- In this lesson, we'll learn about non-empty lists, another example of recursive data.

Empty lists

- Most computations on lists make sense on empty lists
 - $(\text{sum empty}) = 0$
 - $(\text{product empty}) = 1$
 - $(\text{double-all empty}) = \text{empty}$
 - 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 constructor and observer templates for lists don't make sense, either.
- For these problems, we can use a different data definition that is suited for dealing with lists that are always non-empty.
- Let's imagine we've defined a data type called **Sardine**, and we want to work with non-empty lists of **Sardines**.

Constructor Templates for Non-Empty List of Sardines

```
;; Data Definition for NonEmptySardineList:  
  
;; CONSTRUCTORS  
;; (cons s empty) where s is a Sardine  
;; (cons s ss)  
;;     where s is a Sardine  
;;     and ss is a NonEmptySardineList
```

Observer Template for Non-Empty List

```
;; nesl-fn : NonEmptySardineList -> ??  
(define (nesl-fn ne-1st)  
  (cond  
    [(empty? (rest ne-1st)) (... (first ne-1st))]  
    [else (...  
            (first ne-1st)  
            (nesl-fn (rest ne-1st))])))
```

nesl-fn =
NonEmptySardineList-
Function 😊

(rest ne-1st) is a
NonEmptySardineList
so call nesl-fn on it

Template Questions for Non-Empty Lists

```
;; nesl-fn : NonEmptySardineList -> ??  
(define (nesl-fn ne-1st)  
  (cond  
    [(empty? (rest ne-1st)) (... (first ne-1st))]  
    [else (...  
            (first ne-1st)  
            (nesl-fn (rest ne-1st))]))])
```

The diagram consists of two light blue callout boxes with white text. The first box is located at the bottom left and has a blue arrow pointing to the recursive call '(nesl-fn (rest ne-1st))' in the code. The second box is located at the bottom right and has a blue arrow pointing to the base case '(... (first ne-1st))' in the code.

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

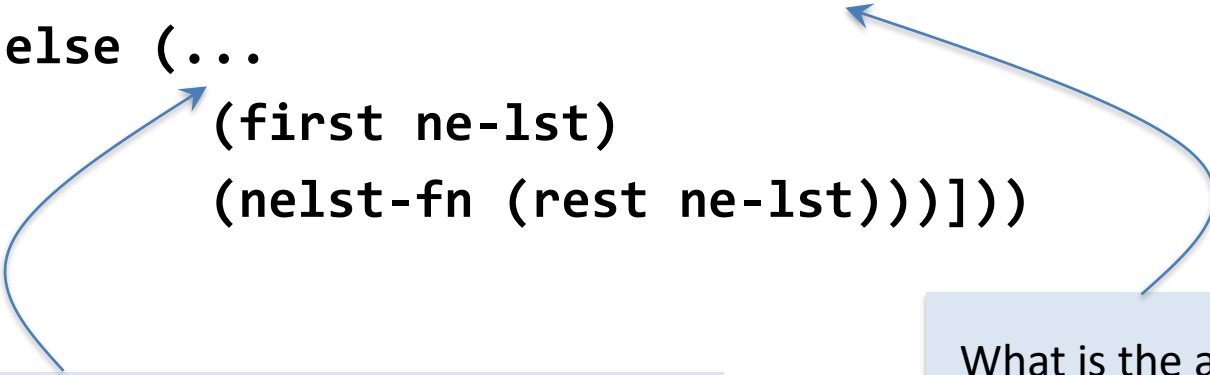
CONSTRUCTOR TEMPLATES for NonEmptyXList

- (cons X empty)
- (cons X NonEmptyXList)

In your assignments, you don't need to write down a separate interpretation for NonEmptyXList; a NonEmptyXList always represents a non-empty sequence of X's in the standard way.

Template Questions for Non-Empty Lists

```
;; nelst-fn : NonEmptyXList -> ??  
(define (nelst-fn ne-1st)  
  (cond  
    [(empty? (rest ne-1st)) (... (first ne-1st))]  
    [else (...  
            (first ne-1st)  
            (nelst-fn (rest ne-1st))]))])
```



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

```
;; intlist-max : NonEmptyListOfInteger -> Integer
;; GIVEN: a non-empty list of integers,
;; RETURNS: the largest element of the list
(define (intlist-max ne-1st)
  (cond
    [(empty? (rest ne-1st)) (first ne-1st)]
    [else (max
             (first ne-1st)
             (intlist-max (rest ne-1st)))]))
```

Example: average

`nl-avg` : `NonEmptyNumberList` -> `Number`

Given a non-empty `NumberList`, returns its average

`(nl-avg (cons 11 empty)) = 11`

`(nl-avg (cons 33 (cons 11 empty))) = 22`

`(nl-avg (cons 33 (cons 11 (cons 11 empty)))) = 55/3`

Example: average

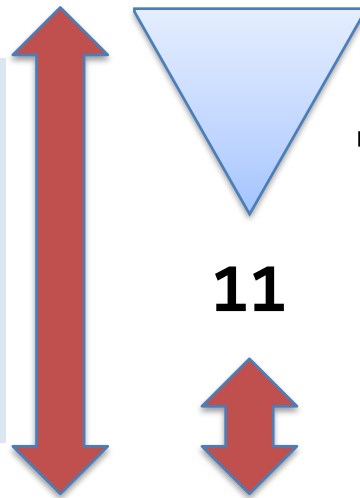
```
;; nl-avg : NonEmptyNumberList -> Number
;; Given a non-empty NumberList, returns its average
;; strategy: use template for NonEmptyNumberList
(define (nl-avg ne-1st)
  (cond
    [(empty? (rest ne-1st)) (first ne-1st)]
    [else (... ←
            (first ne-1st)
            (nl-avg (rest ne-1st)))]))
```

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?

But wait: there's no way to answer that question!

- $(\text{n1-avg } (\text{list } 33 \ 11 \ 11)) = 55/3$

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.



$$\rightarrow (\dots \ 33 \ 11) = 55/3$$

- $(\text{n1-avg } (\text{list } 33 \ 11)) = 22$

$$\rightarrow (\dots \ 33 \ 11) = 22$$

- Can't have both!

Try something simpler!

`nl-avg : NonEmptyNumberList -> Number`

Given a non-empty `NumberList`, returns its average

Strategy: combine simpler functions

```
(define (nl-avg lst)
```

```
  (/ (nl-sum lst) (nl-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. Watch out for situations like this!

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 `XList` if that makes sense (even if the list in the problem never happens to be empty).
- If you're using a `NonEmptyXList` template, and you have duplicated code, that's a sign that it should be a plain old `XList`.

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

- Study `04-3-non-empty-lists.rkt` in the Examples folder.
- If you have questions about this lesson, ask them on the Discussion Board.
- Go on to the next lesson.