CS 135 Winter 2019
Tutorial 04: Mixed Data Types, Lists and Recursion

Goals of this tutorial
You should be able to...

• understand and write data definitions for lists
• understand and use the template for processing lists to write recursive functions consuming this type of data.

Group Problem: Mixed Data
Before we start reviewing lists, let's review an example of a problem dealing with mixed data. Consider the following data definitions:

(define-struct snow (inches good-for-snowman?))
;; A Snow is a (make-snow Num Bool)

(define-struct ice (slippery? salted?))
;; An Ice is a (make-ice Bool Bool)

;; A Wednesday-Off is one of:
;; * Snow
;; * Ice
Group Problem: Mixed Data
This is what a template function may look like for a mixed data type problem:

(define (wednesday-off-template wed-off)
  (cond
   [(snow? wed-off)
    (... (snow-inches wed-off) ...
      (snow-good-for-snowman? wed-off) ...)]
   [(ice? wed-off)
    (... (ice-slippery? wed-off) ...
      (ice-salted? wed-off) ...)])

Group Problem: Mixed Data
Write a function process which consumes a Wednesday-Off and produces
the inches of Wednesday-Off if it is a Snow and produces whether it has
been salted if Wednesday-Off is an Ice. Provide a contract.

Review: List data definition
;; A (listof X) is one of:
;; † empty
;; † (cons X (listof X))

From the data definition, a list of values of any type is either empty
or it consists of a first value followed by a list of values (the rest of
the list).

This is a recursive definition. It contains a base case, and a
recursive (self-referential) case.

Recursive types should be processed with recursive functions.
Review: Basic list constructs

- **empty**: A value representing a list with 0 items.
- **cons**: Consumes an item and a list and produces a new, longer list.
- **first**: Consumes a nonempty list and produces the first item.
- **rest**: Consumes a nonempty list and produces the same list without the first item.
- **empty?**: Consumes a value and produces true if it is empty and false otherwise.
- **cons?**: Consumes a value and produces true if it is a cons value and false otherwise.

Review: Substitution rules

If a, b, c are values and c is non-empty:

1. \( (\text{first } (\text{cons } a \ b)) \Rightarrow a \)
2. \( (\text{rest } (\text{cons } a \ b)) \Rightarrow b \)
3. \( (\text{empty? } \text{empty}) \Rightarrow \text{true} \)
4. \( (\text{empty? } c) \Rightarrow \text{false} \)
5. \( (\text{cons? } (\text{cons } a \ b)) \Rightarrow \text{true} \)
6. \( (\text{cons? } a) \Rightarrow \text{false} \)

Review: List Templates and Data Definitions

**CQ**: Which of the following goes in the blank for the listof-X-template?

```scheme
;; listof-X-template: (listof X) → Any
(define (listof-X-template loX)
  (cond
   [(empty? loX) . . .]
   [else . . .]))
```

- **A**: (. . . (first loX) . . . (rest loX) . . .)
- **B**: (. . . (first loX) . . . (listof-X-template loX) . . .)
- **C**: (. . . (rest loX) . . . (listof-X-template (first loX)) . . .)
- **D**: (. . . (first loX) . . . (listof-X-template (rest loX)) . . .)
- **E**: (. . . (first loX) . . . (listof-X-template (rest loX)) . . . loX . . .)
Group Problem - sum-num
As a warmup, based on the previous template, write a function sum-num that consumes a list of numbers and produces the sum of those numbers. Provide contract and examples.

Group Problem - longer-str
Write a function longer-str that consumes a list of strings and a target string and produces the number of strings in the list that have length greater than the target string. Provide contract and examples.

Hint: (string-length x) produces the length of the string x.

Group Problem - strings-equal?
Based on the previous template for list of X, write a function strings-equal? that consumes a list of strings and produces true if all of the strings are equal, and false otherwise. Include the contract and examples.

Hint: The template includes only one base case, but sometimes functions need multiple base cases.
Group Problem - list of Wednesday-Off

Recall:

```scheme
(define-struct snow (inches good-for-snowman?)) ;; A Snow is a (make-snow Num Bool)

(define-struct ice (slippery? salted?)) ;; An Ice is a (make-ice Bool Bool)

;; A Wednesday-Off is one of:
;; * a Snow
;; * an Ice
```

Group Problem - list of Wednesday-Off

Write a function `total-inch` that takes in a list of Wednesday-Off and produce the total number of inches of snow. Provide a contract.

```scheme
(check-expect (total-inch (cons (make-snow 10 true) (cons (make-ice false true) (cons (make-snow 2 false) empty)))) 12)
(check-expect (total-inch (cons (make-ice true true) empty)) 0)
```

Group Problem - list of Wednesday-Off

Next, write a function `make-salted-ices` that takes in a list of Boolean values and produces a list of salted ices with slippery? corresponding to the Booleans in the consumed list, in the same order. Provide a contract.

```scheme
(check-expect (make-salted-ices (cons true (cons false empty))) (cons (make-ice true true) (cons (make-ice true false) empty)))
(check-expect (make-salted-ices empty) empty)
```
Group Problem - list of Wednesday-Off

Lastly, let's write a function shovel-snow that takes in a list of snow and another snow and should return a new list, with all the snow in the consumed list that equals the second argument removed from the original list. Provide a contract.

(check-expect (shovel-snow (cons (make-snow 1 true) (cons (make-snow 2 true) (cons (make-snow 1 true) empty))))
  (cons (make-snow 1 true) empty))
(check-expect (shovel-snow (cons (make-snow 1 true) empty) (make-snow 1 false))
  (cons false (cons (make-snow 1 true) empty)))