CS 135 Winter 2020

Tutorial 05: More Lists and Recursion

Announcements

- There will be no tutorials or assignments due during reading week.
- Assignment 5 is due on **Tuesday, February 25**, at **9:00 PM**.
- Office hours will be changing during reading week, look at the course website for updated information.

Goals of this tutorial

You should be able to...

- Use **List Abbreviations** for lists
- Process two lists in lockstep in a single function
- Understand and work with **dictionaries** and their variants
- Understand and process **Two-Dimensional Data** represented by **Nested Lists**

Clicker Question1: Debugging

How many errors are there in the following function?

```
;; (sum-until-even lon) produces the sum of all numbers before the first even number.
;; sum-until-even: (listof Num) → Int

(define (sum-until-even lon)
    (cond [(empty? lon) 0]
          [(even? (first lon)) (sum-until-even (rest lon))]
          [else (+ (first lon) (sum-until-even (rest lon)))]))
```

A It is a perfect function!

B 1

C 3

D 4

E Too many to count.

Review: List Abbreviations

List abbreviations are available in language level “beginning students with List Abbreviations”, and all subsequent levels.

The expression:

(cons exp1 (cons exp2 (... (cons expn empty) ...)))

Can be shortened to: (list exp1 exp2 ... expn)

Example: (cons 1 (cons 'a (cons 32 (cons "hello" empty))))

Is equal to: (list 1 'a 32 "hello")

Review: Cons vs. List

cons and list work differently and serve different purposes. We use list to construct a list of fixed length (whose length is known when we are writing the program). We use cons to construct a list from one new element (the first) and a list of arbitrary length, (whose length is known only when the second argument to cons is evaluated during the running of the program).

```
(define (foo n)
        (cond [(= n 1) empty]
              [else (list n (foo (- n 1)))]))
```

(foo 3) $\Rightarrow$ (list 3 (list 2 empty))

(list 3 (list 2 empty)) $\Rightarrow$ (cons 3 (cons (cons 2 (cons empty empty)) empty))

Clicker Question

(list 1 'blue (list 2 3))

Which is the equivalent cons notation for the list above?

A (cons 1 (cons 'blue (cons (cons 2 (cons 3 empty)) empty)))

B (cons 1 'blue (cons 2 3 empty) empty)

C (cons 1 (cons 'blue (cons 2 (cons 3 empty))))

D (cons 1 (cons 'blue (cons 2 3)))

E (cons 1 (cons 'blue (cons (cons (cons 2 (cons 3 empty)) empty) empty) empty) empty)

Clicker Question

```
(define lonum (list (list 5) (list 4 3) (list 2 1)))
```

Which of the following would produce a value of 3?

A `(rest (first (rest lonum)))`

B `(first (rest (rest lonum)))`

C `(first (rest (rest (rest lonum))))`

D `(rest (rest (first (rest lonum))))`

E `(first (rest (first (rest lonum))))`

Review - Dictionaries

```
;; A Dict is one of:
;; * empty
;; * (cons (list Nat Str) Dict)
```

The Nat-Str list is called a key-value pair, where Nat is the key and Str is the value.

Problem 1: Update Contact

You have reconnected with an “old flame”, they gave you their number, and you aren’t sure if you still have it in your phone. Define a function that takes in someone’s contact information (list Nat Str), and a Phone-Book. It produces a new Phone-Book with the contact’s name updated if their number is found, or their contact information appended to the end if it is not found.

Within the Phone-Book, each phone number must be unique.

```
;; A Phone-Book is one of:
;; * empty
;; * (cons (list Nat Str) Phone-Book)
```

Problem 2: One-Zero?

Write a function one-zero? that checks each element of two lists of **Natural Numbers** with same length in **lockstep** to determine if for every position in the list, exactly one of the two lists has element 0.

i.e.

- (one-zero? (list 1 0 2) (list 0 1 0)) $\rightarrow$ true.
- (one-zero? (list 0 1) (list 0 0)) $\rightarrow$ false since both lists contain 0 on the first position.
- (one-zero? (list 0 1) (list 2 1)) $\rightarrow$ false since neither list contains 0 on the second position.

Extra pratice: Sort Shapes

```
;; Shape is a (list (anyof 'triangle 'rectangle) Num Num)
;; where the first Num is the side length, and the second Num is
;; the height, both must be greater than 0.
```

Write a function called sort-shapes that uses **insertion sort** to sort a list of Shapes in **non-decreasing** order of area. If two shapes have the same area, they should appear in the same order as in the original list.