CS 135 Fall 2019

Tutorial 05: More Lists and Recursion
Announcements

• There will be no tutorials or assignments due during reading week.

• Assignment 5 is due on **Tuesday, October 22, 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** and **Quoted Notation** for lists
- Process two lists in a single function
- Understand and work with **dictionaries**
- Understand and process **Two-Dimensional Data** represented by **Nested Lists**
Review: List Abbreviations

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

The expression:

\[(\text{cons } \text{exp1} \ (\text{cons } \text{exp2} \ (\ldots \ (\text{cons } \text{expn} \ \text{empty}) \ldots )))\]

Can be shortened to: \((\text{list } \text{exp1} \ \text{exp2} \ldots \ \text{expn})\)

Example: \((\text{cons } 1 \ (\text{cons } 'a \ (\text{cons } 32 \ (\text{cons } "hello" \ \text{empty})\ldots )))\)

Is equal to: \((\text{list } 1 \ 'a \ 32 \ "hello"\)\)
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).

\[
\text{(define (foo n)} \\
\quad \text{(cond [ (= n 1) empty]} \\
\qquad \text{[else (list n (foo ( - n 1)))]})
\]

\[
\text{(foo 3)} \Rightarrow \text{(list 3 (list 2 empty))}
\]

\[
\text{(list 3 (list 2 empty))} \Rightarrow \text{(cons 3 (cons (cons 2 (cons empty empty)) empty))}
\]
Clicker Question 1

(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 2

(cons (cons 5 empty)
  (cons 3 (cons (cons 2 (cons 5 empty))
    (cons 4 empty))))

Which is the equivalent list to the cons statement above?

A  (list 5 3 2 5 4)
B  (list (list 5) (list 3 2 5) 4)
C  (list (list 5) 3 (list 2 5) 4)
D  (list 5 3 (list 2 5) 4)
E  (list (list 5) (list 3) (list 2) (list 5) (list 4))
Clicker Question 3

(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)

;; my-dict-template: Dict → Any

(define (my-dict-template dict)
  (cond [(empty? dict) . . . ]
        [else (. . . (first (first dict)) . . . ; first key
                      . . . (second (first dict)) . . . ; first value
                      (my-dict-template (rest dict)))]))
Problem 1: Update Contact

You have reconnected with a “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. And will produce 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 1: Update Contact - Design Recipe

(define emergency-contacts (list (list 22222 "UW-Police") (list 911 "Regular-Police")))

;; (update-contact contact-info phone-book) Consumes a contact-info
;; and a phone-book, and will produce a new updated phone-book.
;; update-contact: (list Nat Str) Phone-Book → Phone-Book
;; Example:
(check-expect (update-contact (list 1337 "Leet") emergency-contacts)
             (list (list 22222 "UW-Police") (list 911 "Regular-Police") (list 1337 "Leet")))

(define (update-contact contact-info phone-book) . . .)

;; Tests:
(check-expect (update-contact (list 911 "Medics") emergency-contacts)
             (list (list 22222 "UW-Police") (list 911 "Medics")))
Problem 2: Same List

Racket comes with many comparative functions for many different data types however, comparing lists is one area where Racket falls short. `equal?` could be used to compare two lists, but we can be more specific. Define a function that checks each element of two lists in lockstep to determine if they are the same.

For the sake of simplicity, both arguments will be lists of Natural Numbers.
Problem 2: Same List: Design Recipe

(define test-list (list 8 0 0 8 1 3 5))

;; (list=? lst1 lst2) Consumes lst1 and lst2 both lists of
;; natural numbers and determines if they are the same list.
;; list=?: (listof Nat) (listof Nat) → Bool
;; Examples:
(check-expect (list=? test-list test-list) true)

(define (list=? lst1 lst2) . . . )

;; Tests:
(check-expect (list=? (list) (list)) true)
(check-expect (list=? (list) (list 0)) false)
(check-expect (list=? (list 0 1) (list 1 0)) false)
Review - Tables

We can use a list of lists to represent a 2-dimensional table.

For example, a (3 X 4) table would look like:

(list (list '1 '2 '3)
       (list '4 '5 '6)
       (list '7 '8 '9)
       (list 'A '0 'N))
Problem 3: Diagonal

Define a function that consumes a Nat “length” and produces a square table of that length where all entries on the diagonal (top-left to bottom-right) being 1 and the rest are 0.

(draw-diagonal 0) ⇒ empty
(draw-diagonal 4) ⇒ (list
  (list 1 0 0 0)
  (list 0 1 0 0)
  (list 0 0 1 0)
  (list 0 0 0 1))