CS 135 Winter 2019

Tutorial 5: More lists and recursion
Goals of this tutorial

You should be able to...

- use list abbreviations and quoted notation for lists.
- work with nested lists
- process Association Lists
- process a list with number(s)
- apply recursion on a number
- understand and process two-dimensional data represented by nested lists.
Review: List Abbreviations

List abbreviations are available in language level Beginning Student 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 abbreviated to

$$(\text{list } \text{exp1} \ \text{exp2} \ \ldots \ \text{expn})$$

Example: $$(\text{cons } 0.5 \ (\text{cons } \text{'}a\text{ (cons } 42 \ (\text{cons } \text{"hi" empty}))))$$
is equivalent to $$(\text{list } 0.5 \ \text{'}a\text{ } 42 \ \text{"hi"})$$
Review: List Abbreviation

cons and list have different results and different purposes.

We use list to construct a list of fixed size (whose length is known when we write the program).

We use cons to construct a list from one new element (the first) and a list of arbitrary size (whose length is known only when the second argument to cons is evaluated during the running of the program).
Review: Quoting Lists

If lists built using `list` consist of just `symbols`, `strings`, and `numbers`, they may be further abbreviated using quotes.

```
(cons 'red (cons 'blue (cons 'green empty)))
```
can be written
```
'(red blue green).
```

```
(list 5 4 3 2)
```
can be written
```
'(5 4 3 2),
```
because quoted numbers evaluate to numbers; that is, '
```
1
```
is the same as 1.

The same goes for strings:
```
(list "hi" "bye")
```
can be written as
```
'("hi" "bye")
```

Now we can write `empty` as `(list)` or `'( )`.
Clicker Question - List Translation

Given this list:

(list 1 'blue (list 2 3))

What is the equivalent cons statement?

A  (cons 1 (cons 'blue (cons (cons 2 (cons 3 empty)) 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 - Nested Lists

\[(\text{cons} \ (\text{cons} \ 5 \ \text{empty})
  \ (\text{cons} \ 3 \ (\text{cons} \ (\text{cons} \ 2 \ (\text{cons} \ 5 \ \text{empty}))
  \ (\text{cons} \ 4 \ \text{empty}))))\]

Which of the following lists is equivalent to the one above?

A \ (\text{list} \ 5 \ 3 \ 2 \ 5 \ 4)
B \ (\text{list} \ (\text{list} \ 5) \ (\text{list} \ 3 \ 2 \ 5) \ 4)
C \ (\text{list} \ (\text{list} \ 5) \ 3 \ (\text{list} \ 2 \ 5) \ 4)
D \ (\text{list} \ (\text{list} \ 5) \ (\text{list} \ 3) \ (\text{list} \ 2) \ (\text{list} \ 5) \ (\text{list} \ 4))
Clicker Question - Nested Lists

(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 - Association Lists

;; An association list (AL) is one of:
;;  * empty
;;  * (cons (list Num Str) AL)

;; my-al-fn: AL → Any
(define (my-al-fn alst)
  (cond
   [(empty? alst) . . .]
   [else (. . . (first (first alst)). . . ; first key
              . . . (second (first alst)). . . ; first value
              (my-al-fn (rest alst)))]))
Group Problem - association lists

Just like how we can remove keys from a dictionary, we could remove keys from an association list. Write a function, `remove-al`, that consumes an association list, `alst` and a number `k`. It produces the same AL but with the key-value pair corresponding to `k` removed from the association list. If `k` is not in the association list, there will be no changes to it. Note that keys in an association list are unique.
Group Problem - ones-on-diagonal

We can use a list of lists to represent a 2-dimensional table. For example, here is a table with 3 rows and 3 columns:

(list (list 5 4 3)
   (list 1 2 3)
   (list 0 2 3))
Group Problem - ones-on-diagonal

Write a function, ones-on-diagonal, that consumes a Nat, n, and produces a table with n rows and n columns, where all the entries on the diagonal are 1 and the rest are 0.

(ones-on-diagonal 0) ⇒ empty
(ones-on-diagonal 4) ⇒

(list (list 1 0 0 0)
 (list 0 1 0 0)
 (list 0 0 1 0)
 (list 0 0 0 1))
Group Problem - Range Remove

Write a function, range-remove, that consumes a list of elements and 2 distinct indices $\text{ind1} \leq \text{ind2}$ and remove all of the elements with indices $i \in [\text{ind1}, \text{ind2})$. You can assume valid input. Here are a few examples:

(range-remove (list 1 2 3 4 5) 3 4) $\Rightarrow$ (list 1 2 3 5)
(range-remove (list 'apple 'pie 'hi) 0 2) $\Rightarrow$ (list 'hi)
(range-remove (list "a" "b" "c") 0 0) $\Rightarrow$ (list "a" "b" "c")
Group Problem - Double Factorial

Mathematically, the formulas for double factorial are as follows. If n is even, then
\[ n!! = n(n - 2)(n - 4)\ldots(4)(2). \]
If n is odd, then
\[ n!! = n(n - 2)(n - 4)\ldots(3)(1). \]
Write a function, `doublefact`, that consumes a Nat and determines its double factorial. For the purpose of this question, you can use the following data definition:

```haskell
;; A Nat is one of:
;; * 1
;; * 2
;; (+ 2 Nat)
```
Group Problem: powers-of-k-alt

Write a function powers-of-k-alt that consumes natural numbers \( b \) and \( k \), and produces a list of length \( b \) containing 
\[
(k^1, -k^2, k^3, -k^4, \ldots, \pm k^b),
\]
where the even powers of \( k \) are negated. For example, (powers-of-k-alt 6 3) produces (list 3 −9 27 −81 243 −729). You may find it helpful to make powers-of-k-alt a wrapper function.