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
\[(\text{cons exp1} \text{ cons exp2} (\ldots (\text{cons expn empty})\ldots)))\]
can be abbreviated to
\[(\text{list exp1 exp2} \ldots \text{expn})\]

Example: \[(\text{cons 0.5} \text{ cons 'a} \text{ cons 42} (\text{cons "hi" empty})))\]
is equivalent to \[(\text{list 0.5 'a 42 "hi"})\]
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)))`
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))`
Clicker Question - Nested Lists

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

Which of the following lists is equivalent to the one 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 (list 5) (list 3) (list 2) (list 5) (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)
;;... . (second (first alst)). . . ; first value
(my-al-fn (rest alst)))

;; 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))

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, \texttt{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:

\[
\begin{align*}
(\text{range-remove } (\text{list 1 2 3 4 5}) \ 3 \ 4) & \Rightarrow (\text{list 1 2 3 5}) \\
(\text{range-remove } (\text{list 'apple 'pie 'hi}) \ 0 \ 2) & \Rightarrow (\text{list 'hi}) \\
(\text{range-remove } (\text{list "a "b "c"}) \ 0 \ 0) & \Rightarrow (\text{list "a "b "c")}
\end{align*}
\]

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, \texttt{doublefact}, that consumes a Nat and determines its double factorial. For the purpose of this question, you can use the following data definition:

\[
\begin{align*}
;; \text{ A Nat is one of:} \\
;; \ * \ 1 \\
;; \ * \ 2 \\
;; \ (+ \ 2 \ \text{Nat})
\end{align*}
\]

Group Problem: powers-of-k-alt
Write a function \texttt{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, \((\text{powers-of-k-alt} \ 6 \ 3)\) produces \((\text{list 3 \ -9 \ 27 \ -81 \ 243 \ -729})\). You may find it helpful to make \texttt{powers-of-k-alt} a wrapper function.