Goals of this tutorial

You should be able to...

- use list abbreviations and quote notation for lists.
- process two lists in a function
- understand and process two-dimensional data represented by nested lists.
- understand how insertion sort works

Review: List Abbreviations

List abbreviations are available in language level Beginning Student with List Abbreviations, and all subsequent levels.

The expression

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

can be abbreviated to

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

Example:

\[(\text{cons } 1 \text{ (cons } 32 \text{ (cons "hello" empty))})\]

is equivalent to

\[(\text{list } 1 \text{ 'a 32 "hello"})\]
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 3 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))))

Group Problem - common-elems?
Consider the following data definition for a NumSet:

;; A NumSet is a (listof Num)
;; requires: the numbers are strictly increasing

Write a function, common-elems?, that consumes 2 NumSets and determines whether or not they have any numbers in common. Do not use helper functions.
Review - Association Lists

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

(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 - remove-al

Write a function, remove-al, that consumes an association list alst and a number k. It produces the same association list but with the key k removed. If k is not in the association list, there will be no changes to alst. 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))

Insertion Sort Trace

We will perform a condensed trace of an insertion sort:

(define (sort lon)
  (cond [(empty? lon) empty]
        [else (insert (first lon) (sort (rest lon)))]))

(define (insert n slon)
  (cond [(empty? slon) (cons n empty)]
        [(<= n (first slon)) (cons n slon)]
        [else (cons (first slon) (insert n (rest slon)))]))

(sort (list 5 3 9 2 5 7 1 4))
Insertion Sort Trace
(sort (list 5 3 9 2 5 7 1 4))
=> (insert 5 (sort (list 3 9 2 5 7 1 4)))

5 3 9 2 5 7 1 4

Insertion Sort Trace
(sort (list 5 3 9 2 5 7 1 4))
=> (insert 5 (sort (list 3 9 2 5 7 1 4)))
=> (insert 5 (insert 3 (sort (list 9 2 5 7 1 4))))

5 3 9 2 5 7 1 4

Insertion Sort Trace
=> (insert 5 (sort (list 3 9 2 5 7 1 4)))
=> (insert 5 (insert 3 (sort (list 9 2 5 7 1 4))))
=> (insert 5 (insert 3 (insert 9 (sort (list 2 5 7 1 4)))))

5 3 9 2 5 7 1 4
Insertion Sort Trace

\[
\Rightarrow (\text{insert } 5 \ (\text{insert } 3 \ (\text{sort } (\text{list } 9 \ 2 \ 5 \ 7 \ 1 \ 4))))
\]

\[
\Rightarrow (\text{insert } 5 \ (\text{insert } 3 \ (\text{insert } 9 \ (\text{sort } (\text{list } 2 \ 5 \ 7 \ 1 \ 4)))))
\]

\[
\Rightarrow (\text{insert } 5 \ (\text{insert } 3 \ (\text{insert } 9 \ (\text{insert } 2 \ (\text{sort } (\text{list } 5 \ 7 \ 1 \ 4))))))
\]

Insertion Sort Trace

\[
\Rightarrow (\text{insert } 5 \ (\text{insert } 3 \ (\text{insert } 9 \ (\text{insert } 2 \ (\text{sort } (\text{list } 7 \ 1 \ 4))))))
\]

\[
\Rightarrow (\text{insert } 5 \ (\text{insert } 3 \ (\text{insert } 9 \ (\text{insert } 2 \ (\text{insert } 5 \ (\text{sort } (\text{list } 7 \ 1 \ 4)))))))
\]

Insertion Sort Trace

\[
\Rightarrow (\text{insert } 5 \ (\text{insert } 3 \ (\text{insert } 9 \ (\text{insert } 2 \ (\text{insert } 5 \ (\text{insert } 7 \ (\text{sort } (\text{list } 1 \ 4))))))))
\]

\[
\Rightarrow (\text{insert } 5 \ (\text{insert } 3 \ (\text{insert } 9 \ (\text{insert } 2 \ (\text{insert } 5 \ (\text{insert } 7 \ (\text{sort } (\text{list } 1 \ 4))))))))
\]
Insertion Sort Trace

\[
=> (insert 5 (insert 3 (insert 9 (insert 2
(insert 5 (insert 7 (sort (list 1 4))))))))
\]

\[
=> (insert 5 (insert 3 (insert 9 (insert 2
(insert 5 (insert 7 (insert 1 (sort (list 4)))))))))
\]
Insertion Sort Trace

\[
\begin{align*}
\Rightarrow & \quad (\text{insert} \ 5 \ (\text{insert} \ 3 \ (\text{insert} \ 9 \ (\text{insert} \ 2 \\
& \quad (\text{insert} \ 5 \ (\text{insert} \ 7 \ (\text{insert} \ 1 \ (\text{insert} \ 4 \ empty))))))) \\
\Rightarrow & \quad (\text{insert} \ 5 \ (\text{insert} \ 3 \ (\text{insert} \ 9 \ (\text{insert} \ 2 \\
& \quad (\text{insert} \ 5 \ (\text{insert} \ 7 \ (\text{insert} \ 1 \ (\text{list} \ 4)))))))) \\
\Rightarrow & \quad (\text{insert} \ 5 \ (\text{insert} \ 3 \ (\text{insert} \ 9 \ (\text{insert} \ 2 \\
& \quad (\text{insert} \ 5 \ (\text{insert} \ 7 \ (\text{list} \ 1 \ 4)))))}})
\end{align*}
\]

\[
\begin{array}{c}
5 \ 3 \ 9 \ 2 \ 5 \ 7 \ 1 \\
\phantom{5} \ 4
\end{array}
\]

Insertion Sort Trace

\[
\begin{align*}
\Rightarrow & \quad (\text{insert} \ 5 \ (\text{insert} \ 3 \ (\text{insert} \ 9 \ (\text{insert} \ 2 \\
& \quad (\text{insert} \ 5 \ (\text{insert} \ 7 \ (\text{list} \ 1 \ 4))))))) \\
\Rightarrow & \quad (\text{insert} \ 5 \ (\text{insert} \ 3 \ (\text{insert} \ 9 \ (\text{insert} \ 2 \\
& \quad (\text{insert} \ 5 \ (\text{list} \ 1 \ 4)))))}})
\end{align*}
\]

\[
\begin{array}{c}
5 \ 3 \ 9 \ 2 \ 5 \ 7 \\
\phantom{5} \ 1 \ 4
\end{array}
\]

Insertion Sort Trace

\[
\begin{align*}
\Rightarrow & \quad (\text{insert} \ 5 \ (\text{insert} \ 3 \ (\text{insert} \ 9 \ (\text{insert} \ 2 \\
& \quad (\text{insert} \ 5 \ (\text{list} \ 1 \ 4)))))}})
\end{align*}
\]

\[
\begin{array}{c}
5 \ 3 \ 9 \ 2 \ 5 \\
\phantom{5} \ 1 \ 4 \ 7
\end{array}
\]
Insertion Sort Trace

\[
\begin{align*}
\Rightarrow & \quad \text{(insert 5 (insert 3 (insert 9 (insert 2 }
\text{(insert 5 (list 1 4 7))))))) \\
\Rightarrow & \quad \text{(insert 5 (insert 3 (insert 9 (insert 2 }
\text{(list 1 4 5 7))))))}
\end{align*}
\]

\[
\begin{array}{c}
5 \quad 3 \quad 9 \quad 2 \\
1 \quad 4 \quad 5 \quad 7
\end{array}
\]

\[
\begin{align*}
\Rightarrow & \quad \text{(insert 5 (insert 3 (insert 9 (list 1 2 4 5 7))))} \\
\Rightarrow & \quad \text{(insert 5 (insert 3 (list 1 2 4 5 7 9)))}
\end{align*}
\]

\[
\begin{array}{c}
5 \quad 3 \quad 9 \\
1 \quad 2 \quad 4 \quad 5 \quad 7
\end{array}
\]

\[
\begin{align*}
\Rightarrow & \quad \text{(insert 5 (list 1 2 4 5 7 9)))}
\end{align*}
\]

\[
\begin{array}{c}
5 \quad 3 \\
1 \quad 2 \quad 4 \quad 5 \quad 7 \quad 9
\end{array}
\]
Insertion Sort Trace

\[ \Rightarrow (\text{insert } 5 \ (\text{insert } 3 \ (\text{list } 1 \ 2 \ 4 \ 5 \ 7 \ 9))) \]
\[ \Rightarrow (\text{insert } 5 \ (\text{list } 1 \ 2 \ 3 \ 4 \ 5 \ 7 \ 9)) \]

Group Problem (Optional) - is-prefix-of?

Write a function, \texttt{is-prefix-of?}, that consumes a list of \texttt{Sym} called \texttt{pattern}, and another list of \texttt{Sym} called \texttt{word}. It produces \texttt{true} if the symbols in \texttt{pattern} occur in the same order at the start of \texttt{word}, and \texttt{false} otherwise.