How could we find the largest element in a list of numbers?

It's meaningless to ask what the largest element of the empty list is. We'll assume our input is non-empty.
Our base case will be when there's only one element in the list.

First, we can define a helper function max-2 that computes the maximum of two inputs:

```
(define max-2
  (lambda (x y)
    (if (>= x y) x y)))
(tester '(max-2 3 4))
(tester '(max-2 4 3))
```
Now we're ready to define `fetch-largest` for lists of arbitrary length!

\[
\text{(define fetch-largest}
\]
\[
\quad (\lambda (\text{lst})
\]
\[
\quad \quad (\text{if (null? (rest lst))}
\]
\[
\quad \quad \quad \quad ;; \text{Base case:}
\]
\[
\quad \quad \quad \quad ;; \text{If there's only one element, return it!}
\]
\[
\quad \quad \quad \quad (\text{first lst})
\]
\[
\quad \quad \quad \quad ;; \text{Recursive case}
\]
\[
\quad \quad \quad \quad (\text{max-2 (first lst)}
\]
\[
\quad \quad \quad \quad \quad (\text{fetch-largest (rest lst)))))}
\]
\[
\text{)}
\]
\[
\text{(tester 'fetch-largest '(3))}
\]
\[
\text{(tester 'fetch-largest '(3 4))}
\]
\[
\text{(tester 'fetch-largest '(4 3))}
\]
\[
\text{(tester 'fetch-largest '(1 2 3 2 0 9 1 6 8))}
\]

Last class we wrote proc-all, which is equivalent to the built-in map function.

We can combine map and `fetch-largest` to find the largest value of a procedure applied to the elements of a list:

\[
\text{(fetch-largest}
\]
\[
\quad (\text{map sin}
\]
\[
\quad \quad '(0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0))
\]

Factorial

The \textit{factorial} notation \(n!\) represents the product of all positive integers from 1 to \(n\), inclusive,

\[
\text{n! = 1 \times 2 \times 3 \times \cdots \times (n-1) \times n}
\]

Factorials are frequently computed in \textit{combinatorics} when we're counting possibilities.
The factorial function can be defined recursively:

**Recursive step:** \( n! = n \cdot (n - 1)! \)

**Base step:** \( n = 1 \), in which case \( n! = 1! = 1 \)

---

**Problem: factorial**

(defn factorial [n] ...?...)

> (factorial 4)
   
24

> (* 4 (* 3 (* 2 (* 1 1)))))
   
24

> (factorial 3)
   
6

> (* 3 (* 2 (* 1 1)))
   
6

> (factorial 1)
   
1

> (factorial 0)
   
1

---

**Recursive definition of factorial**

(defn factorial [n]
  (if (= n 0)
    1
    (* n (factorial (- n 1))))

(tester '(factorial 1))
(tester '(factorial 2))
(tester '(factorial 3))
(tester '(factorial 4))

---

**Procedure calls and return values**

(factorial 3) 6
   ↑
   ↓
(factorial 2) 2
   ↑
   ↓
(factorial 1) 1
   ↑
   ↓
(factorial 0) 1
Problem: Sum of squares

Write a function to add the squares of the first n numbers: \(1^2 + 2^2 + \cdots + n^2\)

What's the base case?

Recursive case?

(define sum-squares
  (lambda (n)
    (if (= n 1)
      1
      (+ (* n n)
         (sum-squares (- n 1))))))

(tester '(sum-squares 1))
(tester '(sum-squares 2))
(tester '(sum-squares 4))

Problem: Multiply a number by itself n times

(define power
  (lambda (base exponent)
    ...?...))

> (power 2 0)
1
> (power 2 1)
2
> (power 2 2)
4
> (power 2 3)
8

Constructing solution to larger problem from solution to smaller one

\[
\begin{align*}
  b^n &= b \cdot (b \cdot b \cdot b \cdots b) \\
  b^n &= b \cdot b^{n-1}
\end{align*}
\]
Recursive definition of power

(define power
  (lambda (base exponent)
    (if (= exponent 0)
        1
        (* base
          (power base (- exponent 1))))))

Acknowledgments

This lecture incorporates material from:
  Tom Ellman
  Luke Hunsberger