Naive list reversal

(define reverse (xs)
  (if (null? xs)
      '()
      (append (reverse (cdr xs))
              (list1 (car xs)))))

Reversal by accumulating parameters

(define revapp (xs ys)
  ; return (append (reverse xs) ys)
  (if (null? xs)
      ys
      (revapp (cdr xs)
               (cons (car xs) ys))))

(define reverse (xs) (revapp xs '()))
A-list example

-> (find 'Building
   '(((Course 105) (Building Barnum)
      (Instructor Ramsey)))
Barnum
-> (val nr (bind 'Office 'Halligan-222
      (bind 'Courses '(105 150TW)
         (bind 'Email 'comp105-grades '()))))
((Email comp105-grades)
 (Courses (105 150TW))
 (Office Halligan-222))
-> (find 'Office nr)
Halligan-222
-> (find 'Favorite-food nr)
()
Laws of association lists

(find k (bind k v a-l)) = v
(find k (bind k’ v a-l)) = (find k a-l), provided k != k’
(find k ’()) = ’() --- bogus!
μScheme vs Impcore

New abstract syntax:

LET (keyword, names, expressions, body)
LAMBDA X (formals, body)
APPLY (exp, actuals)
Introduce local names into environment

(let ([x1 e1] ...
      [xn en])
  e)

Square brackets mean the same as round, but are easier to see
What McCarthy might have done

(let ([val x1 e1]
    ...
    [val xn en])
    e)

(But semantics of let, let*, letrec is much simpler)
Function escapes!

-> (define to-the-n-minus-k (n k)
  (let
    ([x-to-the-n-minus-k (lambda (x)
      (let
        ([x-to-the-n-minus-k (lambda (x)
          (- (exp x n) k)])]
        x-to-the-n-minus-k))]
    x-to-the-n-minus-k))
-> (val x-cubed-minus-27 (to-the-n-minus-k 3 27))
-> (x-cubed-minus-27 2)
-19
No need to name the escaping function

-> (define to-the-n-minus-k (n k)
   (lambda (x) (- (exp x n) k)))

-> (val x-cubed-minus-27 (to-the-n-minus-k 3 27))
-> (x-cubed-minus-27 2)
-19
The zero-finder

(define findzero-between (f lo hi)
  ; binary search
  (if (>= (+ lo 1) hi)
      hi
      (let ([mid (/ (+ lo hi) 2)])
        (if (< (f mid) 0)
            (findzero-between f mid hi)
            (findzero-between f lo mid))))
  (define findzero (f) (findzero-between f 0 100))
Cube root of 27 and square root of 16

\[
\text{\texttt{cube root of 27}} \text{ and } \text{\texttt{square root of 16}}
\]

\[
\rightarrow (\text{\texttt{findzero (to-the-n-minus-k 3 27)}}) \\
3
\]

\[
\rightarrow (\text{\texttt{findzero (to-the-n-minus-k 2 16)}}) \\
4
\]
Lambda questions

(define combine (p? q?)
  (lambda (x) (if (p? x) (q? x) #f)))

(define divvy (p? q?)
  (lambda (x) (if (p? x) #t (q? x)))))

(val c-p-e (combine prime? even?))
(val d-p-o (divvy prime? odd?))

(c-p-e 9) == ?       (d-p-o 9) == ?
(c-p-e 8) == ?       (d-p-o 8) == ?
(c-p-e 7) == ?       (d-p-o 7) == ?
Lambda answers

(define combine (p? q?)
    (lambda (x) (if (p? x) (q? x) #f)))

(define divvy (p? q?)
    (lambda (x) (if (p? x) #t (q? x))))

(val c-p-e (combine prime? even?))
(val d-p-o (divvy prime? odd?))

(c-p-e 9) == #f  (d-p-o 9) == #t
(c-p-e 8) == #f  (d-p-o 8) == #f
(c-p-e 7) == #f  (d-p-o 7) == #t