Four ways of implementing sum-of-reciprocals. The problem is to handle the exception of dividing by 0. If one of the numbers is 0, we would like to return 0 from the entire computation, or (failing that), at least generate an error message.

(define a '(2 3 4 5))
(define b '(2 3 0 5))

; naive implementation
(define (sr1 args)
  (cond ((null? args) 0)
        ((zero? (car args)) (error "List contains a 0"))
        (else (+ (/ 1 (car args)) (sr1 (cdr args))))))

This generates an error message, but does not return cleanly.

(sr1 a) => 77/60
(sr1 b) => Error: List contains a 0

; make it tail-recursive
(define (sr2 args)
  (define (sr2_internal args accum)
    (cond ((null? args) accum)
          ((zero? (car args)) 0)
          (else (+ (/ 1 (car args)) (sr2_internal (cdr args) accum))))
  )

(sr2 a) => 77/60
(sr2 b) => 0

; continuation-passing style
(define (sr3 args)
  (define (sr3_internal args k)
    (cond ((null? args) (k 0))
          ((zero? (car args)) 0)
          (else (sr3_internal (cdr args) k (+ (/ 1 (car args)) v))))
     )

(sr3 a) => 77/60
(sr3 b) => 0

; using call/cc
(define (sr4 args)
  (call/cc (lambda (return)
    (define (sr4_internal args)
      (cond ((null? args) 0)
            ((zero? (car args)) (return 0))
            (else (+ (/ 1 (car args)) (sr4_internal (cdr args))))
      )
    )
  )
)

(sr4 a) => 77/60
(sr4 b) => 0