```
#lang racket

(define (sum-numbers lst)
  (if (= (length lst) 1)
      (first lst)
      (+ (first lst)
         (sum-numbers (rest lst)))))

(define numbers (list 1 2 3 4 5))

(sum-numbers numbers)

(define (append-strings lst)
  (if (= (length lst) 1)
      (first lst)
      (string-append (first lst)
                     (append-strings (rest lst)))))

(define animals (list "cat" "penguin" "hedgehog"))

(append-strings animals)

(define (first-fold f lst)
  (if (= (length lst) 1)
      (first lst)
      (f (first lst)
         (first-fold f (rest lst)))))

(first-fold (lambda (x y)(+ x y)) numbers)

(first-fold (lambda (x y)(string-append x y)) animals)

(define (is-all-numbers? lst)
  (if (= (length lst) 1)
      (number? (first lst))
      (and (number? (first lst))
           (is-all-numbers? (rest lst)))))

(is-all-numbers? (list #t 5 "cat"))
(is-all-numbers? numbers)

(define (sum-numbers-2 lst)
  (letrec ((helper (lambda (l acc)
                     (if (empty? l)
                         acc
                         (helper (rest l) (+ (first l) acc))))))
```

```
    (helper lst 0)))

(sum-numbers-2 numbers)

(define (is-all-numbers?-2 lst)
  (letrec ((helper (lambda (l acc)
                     (if (empty? l)
                         acc
                         (helper (rest l) (and (number? (first
l))
                                               acc))))))
    (helper lst #t)))

(is-all-numbers?-2 numbers)

(define (my-fold f acc lst)
  (if (empty? lst)
      acc
      (my-fold f
               (f (first lst)
                  acc)
               (rest lst))))

(my-fold (lambda (x acc) (string-append x acc)) "" animals)

;; built-in fold

(define (sum-numbers-foldl lst)
  (foldl (lambda (x y)(+ x y)) 0 lst))

(sum-numbers-foldl numbers)

(define (sum-numbers-foldr lst)
  (foldr (lambda (x y)(+ x y)) 0 lst))

(sum-numbers-foldr numbers)

(define (append-strings-l lst)
  (foldl (lambda (x y)(string-append x y)) "" lst))

(append-strings-l animals)

(define (append-strings-r lst)
  (foldr (lambda (x y)(string-append x y)) "" lst))

(append-strings-r animals)
```

```
92
93  ;;
94
95  (foldl (lambda (x y z) (string-append x y z)) "" animals animals)
96
97  (foldl (lambda (x y)(string-append x y)) "" (map (lambda (x)
97  (string x))
98                                                   (string->list
98  "animals")))
99
100
101 (foldl (lambda (x y)(string-append (string x) y)) ""
101 (string->list "animals"))
102
103
104
105
```