

```

1 #lang racket
2
3 (define numbers (list 2 20 3 5 10))
4
5 (define words (list "cat" "penguin" "fish"))
6
7 (define (sum-numbers lst)
8   (if (= (length lst) 1)
9       (first lst)
10      (+ (first lst)
11         (sum-numbers (rest lst)))))
12
13 (sum-numbers numbers)
14
15 (define (append-strings lst)
16   (if (= (length lst) 1)
17       (first lst)
18       (string-append (first lst)
19                      (append-strings (rest lst)))))
20
21 (append-strings words)
22
23 (define (append-strings-tail lst res)
24   (if (= (length lst) 1)
25       (string-append (first lst) res)
26       (append-strings-tail (rest lst) (string-append (first lst) res))))
27
28 (append-strings-tail words "")
29
30 (define (sum-numbers-fold lst)
31   (foldl (lambda (x y)(+ x y))
32         0
33         lst))
34
35 (sum-numbers-fold numbers)
36
37 (define (append-strings-fold lst)
38   (foldl (lambda (x y)(string-append y x))
39         ""
40         lst))
41
42 (append-strings-fold words)
43
44 (define (count-cats lst)
45   (foldl (lambda (str total)(if (equal? str "cat")
46                               (+ 1 total)
47                               total))
48         0
49         lst))
50
51 (count-cats words)

```

```

52
53 (define (append-strings-foldr lst)
54   (foldr (lambda (x y)(string-append x y))
55         ""
56         lst))
57
58 (append-strings-foldr words)
59
60 (define (prod-numbers lst)
61   (foldl (lambda (x y) (* x y)) 1 lst))
62
63 (prod-numbers numbers)
64
65 (define (my-and lst)
66   (foldl (lambda (x y)(if x
67                         (if y #t #f)
68                         #f))
69         #f
70         lst))
71
72 (my-and (list #t (= 0 0)))
73
74 (define (my-xor lst)
75   (foldl (lambda (x res)(if x
76                           (if res #f #t)
77                           #t))
78         #f
79         lst))
80
81 (define notxor-bools (list #t #t #t #f #f))
82
83 (define xor-bools (list #f #f #t #f #f))
84
85 (my-xor xor-bools)
86
87 (my-xor notxor-bools)
88
89 (define (my-xor-2 lst)
90   (if (= (length (filter (lambda(x)x) lst)) 1)
91       #t
92       #f))
93
94 (my-xor-2 xor-bools)
95
96 (my-xor-2 notxor-bools)
97
98 (define (xor-fold lst)
99   (foldl (lambda (x y)(if x
100          (if (first y)
101              (list #f 1)
102              (list #t (first (rest y))))

```

```
103 |                                     y))
104 |         (list #f 0)
105 |         lst))
106 |
107 |     (xor-fold xor-bools)
108 | (xor-fold notxor-bools)
109 |
110 | (define (my-xor-3 lst)
111 |   (let ((res (xor-fold lst)))
112 |     (if (and (first res) (= (first (rest res)) 0))
113 |         #t
114 |         #f)))
115 |
116 | (my-xor-3 notxor-bools)
117 |
118 | (my-xor-3 xor-bools)
119 |
```