```
#lang racket

(define numbers (list 7 6 10 1))

(define (add-five l)
  (if (empty? l)
      l
      (cons (+ (first l) 5)
            (add-five (rest l)))))

(add-five numbers)

(define (add-seven l)
  (if (empty? l)
      l
      (cons (+ (first l) 7)
            (add-seven (rest l)))))

(add-seven numbers)

(define (add-n l n)
  (if (empty? l)
      l
      (cons (+ (first l) n)
            (add-n (rest l) n))))

(add-n numbers 8)

(define (exclaim l)
  (if (empty? l)
      l
      (cons (string-append (first l) "!")
            (exclaim (rest l)))))

(define animals (list "lions" "tigers" "bears"))
```

```
36
37 (exclaim animals)
38
39 (lambda (x y) (+ x y))
40
41 (lambda (x y) (string-append x y))
42
43 (map (lambda (x) (+ x 5)) numbers)
44
45 (map (lambda (x) (string-append x "!")) animals)
46
47 (define (generic-add lst f x)
48  (map (lambda (y) (f y x)) lst))
49
50 (generic-add numbers + 5)
51
52 (generic-add numbers * 5)
53
54 (generic-add animals string-append "!")
55
56
57 (define (apply-to-cat f)
58   (f "cat"))
59
60 (define (add-exclamation x)
61   (string-append x "!"))
62
63 (apply-to-cat add-exclamation)
64
65 (apply-to-cat (lambda (x) (string-length x)))
66
67 (map (lambda (x) #t) numbers)
68
69 (map (lambda (x) #t) animals)
70
```

```
(define (is-divisible lst n)
  (map (lambda (x) (= (modulo x n) 0)) lst))

(is-divisible numbers 7)
```