

```

1 #lang racket
2
3 (require racket/stream)
4
5 (define (plus-1 n)
6   (stream-cons n (plus-1 (+ n 1)))) ;; Can't use
6 ordinary list functions, because we're working with a
6 stream now
7
8 (define positive-numbers (plus-1 0))
9
10 ;; Exercise: write a stream that contains longer and
10 longer sequences of "a"
11
12 (define (append-1 str)
13   (stream-cons str (append-1 (string-append str str))))
14
15 (define my-a (append-1 "a"))
16
17 (define my-cats (append-1 "cat"))
18
19
20 (stream-first(stream-rest(stream-rest(stream-rest
20 (stream-rest my-a))))))
21
22 ;; Creating a stream from another stream using filter:
23
24 (define evens (stream-filter (lambda(x)(= 0 (modulo x
24 2))))
25                               positive-numbers))
26
27 (stream-first (stream-rest evens))
28
29 ;; Creating a stream from another string using map:
30
31 (define (string-multiply str n)
32   (if (= 0 n)
33       ""
34       (string-append str (string-multiply str (- n 1)))))
35
36 (string-multiply "cat" 3)
37
38 (define cats-2 (stream-map (lambda (x)(string-multiply

```

```

38 "cat" x)) positive-numbers))
39
40 (stream-first
40 (stream-rest(stream-rest(stream-rest(stream-r
40 est cats-2))))))
41
42 ;; Creating a stream from two streams:
43
44 ;; Map is defined for multiple lists:
45
46 (define (zip l1 l2)
47   (map (lambda (x y)(list x y)) l1 l2))
48
49 (zip (list 1 2 3)(list 4 5 6))
50
51 ;; Irritatingly, stream-map is NOT defined for multiple
51 streams:
52
53 #|(define (stream-zip l1 l2)
54   (stream-map (lambda (x y)(list x y))
55               l1
56               l2))|#
57
58 ;; (stream-zip positive-numbers (stream-rest
58 positive-numbers))
59
60 ;; But no worries--- we can write our own
61
62 (define (stream-map-n f args)
63   (stream-cons (apply f (map (lambda (x) (stream-first
63 x)) args))
64                 (stream-map-n f (map (lambda (x)
64 (stream-rest x)) args))))
65
66 (define evens-2 (stream-map-n (lambda (x y)(+ x y))
67                               (list positive-numbers
67 positive-numbers)))
68
69 (stream-first (stream-rest evens-2))
70
71 (define evens-3 (stream-map (lambda (x)(* x 2))
72                             positive-numbers))
73

```

```
74 (stream-first (stream-rest evens-3))
75
76 ;; Fibonacci
77
78 (define (fib a b)
79   (stream-cons a (fib b (+ a b))))
80
81 (define fibos (fib 0 1))
82
83 (stream-ref fibos 7)
84
85 (define fibos-2 (stream-cons 0
86   (stream-cons 1
87     (stream-map-n
88       (lambda (x y)(+ x y))
89       (list fibos-2
90         (stream-rest
91           fibos-2)))))))
92
93 (stream-ref fibos-2 7)
94
95 ;; Something cool: we have a recursive definition that
96 ;; isn't in a function!
97
98 (define facs (stream-cons 1
99   (stream-map-n (lambda (x y)(*
100     x y))
101     (list facs
102       (stream-rest (stream-rest positive-numbers)))))))
103
104
105
106
107
```