```
#lang racket

(require racket/stream)

(define (plus-1 n)
  (stream-cons n (plus-1 (+ n 1))))

(define pos-integers (plus-1 0))
pos-integers

(stream-first pos-integers)  ; Can't use ordinary list functions, because we're working with a stream now
(stream-rest pos-integers)

;; Exercise: write a stream that contains longer and longer sequences of "a"

(define (append-1 str)
  (stream-cons str (append-1 (string-append str str))))

(define cats (append-1 "cat"))

(stream-first cats)
(stream-first (stream-rest cats))

;; Creating a stream from another stream using filter:

(define evens (stream-filter (lambda (x)(= 0 (modulo x 2))) pos-integers))

```

```
(stream-first evens)
(stream-first (stream-rest (stream-rest
(stream-rest evens))))

;; Creating a stream from another string
using map:

(define (endless-strings str)
  (stream-cons str (endless-strings str)))

(define catstream (endless-strings "cat"))
(stream-first catstream)
(stream-first (stream-rest (stream-rest
(stream-rest catstream))))

(define (string-multiply str n)
  (if (= 0 n)
      ""
      (string-append str (string-multiply
str (- n 1)))))

(define cats-2 (stream-map (lambda
(x)(string-multiply "cat" x))
pos-integers))

cats-2
(stream-first cats-2)
(stream-first (stream-rest (stream-rest
(stream-rest cats-2))))

;; Creating a stream from two streams:

;; Map is actually defined for multiple
```

```
data structures:

(define (zip l1 l2)
  (map (lambda (x y)(list x y)) l1 l2))

(zip (list 1 2 3)(list 4 5 6))

;; Irritatingly, stream-map is NOT
defined for multiple streams:

;(define evens-2 (stream-map (lambda (x
y)(+ x y)) pos-integers pos-integers))

;; But no worries--- we can write our own

(define (stream-map-n f args)
  (stream-cons (apply f (map (lambda
(x)(stream-first x)) args))
               (stream-map-n f (map
(lambda (x)(stream-rest x)) args))))

(apply + (map (lambda (x)(stream-first
x)) (list pos-integers pos-integers)))

(define evens-2 (stream-map-n (lambda (x
y)(+ x y)) (list pos-integers
pos-integers)))

(stream-first (stream-rest (stream-rest
evens-2)))

(define (fib a b)
  (stream-cons a (fib b (+ a b))))
```

```
78
79  (define fibos (fib 0 1))
80
81  fibos
82
83  (stream-ref fibos 7)
84
85  (define fibos-2 (stream-cons 0
85  (stream-cons 1 (stream-map-n (lambda (x
85  y)(+ x y)) (list fibos-2 (stream-rest
85  fibos-2))))))
86
87  fibos-2
88  (stream-ref fibos-2 7)
89
90  ;; Something cool: we have a recursive
90  definition that isn't in a function!
91
92  (define facs (stream-cons 1 (stream-map-n
92  (lambda (x y)(* x y)) (list facs
92  (stream-rest (stream-rest
92  pos-integers))))))
93  facs
94  (stream-ref facs 0)
95  (stream-ref facs 1)
96  (stream-ref facs 2)
97  (stream-ref facs 3)
98
99  (define p (stream-cons 1 (stream-map-n +
99  (list p p))))
100 (stream-ref p 0)
101 (stream-ref p 1)
102 (stream-ref p 2)
```

```
103 | (stream-ref p 3)
```