```
#lang racket

(require racket/stream)

(define (plus-1 n)
  (stream-cons n (plus-1 (+ n 1)))) ;; Can't use ordinary list functions, because we're working with a stream now

(define positive-numbers (plus-1 0))

;; 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 my-a (append-1 "a"))

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


(stream-first(stream-rest(stream-rest(stream-rest (stream-rest my-a)))))

;; Creating a stream from another stream using filter:

(define evens (stream-filter (lambda(x)(= 0 (modulo x 2)))
                             positive-numbers))

(stream-first (stream-rest evens))

;; Creating a stream from another string using map:

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

(string-multiply "cat" 3)

(define cats-2 (stream-map (lambda (x)(string-multiply
```

```
"cat" x)) positive-numbers))

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

;; Creating a stream from two streams:

;; Map is defined for multiple lists:

(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 (stream-zip l1 l2)
  (stream-map (lambda (x y)(list x y))
              l1
              l2))|#

;; (stream-zip positive-numbers (stream-rest positive-numbers))

;; 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))))

(define evens-2 (stream-map-n (lambda (x y)(+ x y))
                              (list positive-numbers positive-numbers)))

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

(define evens-3 (stream-map (lambda (x)(* x 2))
                            positive-numbers))

```

```
(stream-first (stream-rest evens-3))

;; Fibonacci

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

(define fibos (fib 0 1))

(stream-ref fibos 7)

(define fibos-2 (stream-cons 0
                             (stream-cons 1
                                          (stream-map-n
(lambda (x y)(+ x y))

                             (list fibos-2

                                   (stream-rest
fibos-2))))))

(stream-ref fibos-2 7)

;; Something cool: we have a recursive definition that
isn't in a function!

(define facs (stream-cons 1
                          (stream-map-n (lambda (x y)(*
x y))
                                        (list facs

(stream-rest (stream-rest positive-numbers))))))


(stream-ref facs 4)
```