```
#lang racket
5

(lambda () (printf "Hello world!"))

(define hello-world (lambda () (printf "Hello world!")))

(hello-world)

((lambda () (printf "Hello world!")) )

5

(define x 5)


(define sum-squares (lambda (lst)
                      (if (= 1 (length lst))
                          (* (first lst) (first lst))
                          (+ (* (first lst) (first lst))
                             (sum-squares (rest lst)))))))

(define numbers (list 1 2 3))

(define (square x)
  (* x x))

(define sum-squares-2 (lambda (lst)
                        (if (= 1 (length lst))
                            (square (first lst))
                            (+ (square (first lst))
                               (sum-squares-2 (rest lst)))))))

(sum-squares-2 numbers)

```

```
(define (sum-squares-3 lst)
  (let ((square-1 (lambda (x) (* x x))))
    (if (= 1 (length lst))
        (square-1 (first lst))
        (+ (square-1 (first lst))
           (sum-squares-3 (rest lst))))))

(sum-squares-3 numbers)

(define (reverse-helper str n)
  (if (= n (string-length str))
      ""
      (string-append (reverse-helper str (+ n 1))
                     (string (string-ref str n)))))

(reverse-helper "cat" 0)

(define (reverse str)
  (reverse-helper str 0))

(define (reverse-2 str)
  (letrec ((helper (lambda (str x) (if (= x
(string-length str))
                                       ""
                                       (string-append
(helper str (+ x 1))

(string (string-ref str x)))))))
    (helper str 0)))

(reverse-2 "cat")

```