#lang lazy

(define (try a b)
  (println "I'm in the function!"
  (if (= a 0)
    1
    b))

(try 0 (/ 1 0))

(define (trace op)
  (printf (string-append op "!\n")))

(define squares (map (lambda (x) (trace "Map") (* x x)) (list 1 2 3)))

(squares)

(first squares) ;; calculate the value of the first item

(first squares) ;; Racket remembers what the value is, doesn't recalculate

(define (add x y)
  (trace "Addition"
  (+ x y))

(define (subtract x y)
  (trace "Subtraction"
  (- x y))

(define (multiply x y)
  (trace "Multiply"
  (* x y))

(define (fac n)
(println "Within body!"
(if (= n 1)
  1
  (multiply n (fac (subtract n 1))))))

(fac 5)

(define (tail-fac n)
  (letrec ((helper (lambda (x res)
            (println "Within body!"
              (if (= x 1)
                res
                (helper (subtract x 1)(multiply x res))))))
    (helper n 1)))

(tail-fac 5)

(define (loop-forever)
  (loop-forever))

;;(loop-forever)

(define (three x)
  3)

(three (loop-forever))

(define (our-if test tb fb)
  (if test
    tb
    fb
  ))

(our-if (= 4 4) (printf "true!\n") (printf "false\n"))