```
#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)
```

```
34    (println "Within body!")
35    (if (= n 1)
36        1
37        (multiply n (fac (subtract n 1)))))
38
39  (fac 5)
40
41  (define (tail-fac n)
42    (letrec ((helper (lambda (x res)
43                       (println "Within body!")
44                       (if (= x 1)
45                           res
46                           (helper (subtract x
46  1)(multiply x res))))))
47    (helper n 1)))
48
49  (tail-fac 5)
50
51  (define (loop-forever)
52    (loop-forever))
53
54  ;;(loop-forever)
55
56  (define (three x)
57    3)
58
59  (three (loop-forever))
60
61  (define (our-if test tb fb)
62    (if test
63        tb
64       fb
65        ))
66
67  (our-if (= 4 4) (printf "true!\n") (printf
67  "false\n!"))
```