RECURSION

1. AST annotation (circularity between closures & environment)
2. interpreter (extended-rec-env)
3. Y-combinator (λ-calculus).

(define (G f)
(lan

(define G
(lambda (f)
(lambda (n)
(if (= n 0)
1
(* n (f (- n 1)))))))))

$G : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$

$G\ !\ = (G\ !) = !$

! is a fixed point of G.

(Y g → f)

(define fs
(lambda (f)
(f f)))

(define gs
(lambda (g)
(g g)

```
(define (Y g)
  (let ((s (lambda (x)
```

Y(g):

s = x → g(x(x))

s(s)

G(f):

(n) → if n==0 then 1 else n * f(n-1)

Y(g):

s(s)

where s = (x) → g(x(x))

Y(G)

= s(s)

= g(s(s))

= g(g(s(s)))

Ya(g):

s(s)

where s = (x) → (n) → g(x(x))(n)

cases syntax define-datatype

cond number? null? list? symbol?

(car)

(first second) (member) (cdr) (cons) (caadr) ??

(match) (match-lambda) (match-let).

```
(define-datatype ast ast?
  [num (n number?)]
  [plus (left ast?) (right ast?)]
  [minus (left ast?) (right ast?)]
  [mul (left ast?) (right ast?)])

(λ (a)
  (cases ast a
    [num (n) n]
    [plus (left right) (+ left right)]
      (let
        ([l (eval left)]
         [r (eval right)])
        (+ l r))]
    [minus (left right)
      (let
        ([l (eval left)]
         [r (eval right)])
        (+ l r))]
    [mul ... ])).
```

MEMQ.

(memq value list) → index

(apply). #t #f (boolean?)