```
def lisp_eval(expr, Global_Env):
    if isatom(expr):
        return expr
    if issymbol(expr):
        return Global_Env.lookup(expr)
    #### now we're dealing with code
    if not iscode(expr):
        return Egregious_Error_Message
    Global_Stack = Stack(expr, Global_Env)
    cur_Stack = Global_Stack

    while not Global_Stack.isdone():
        if not cur_Stack.fn_found:
            if isatom(cur_Stack.expr.car):
                cur_Stack.fn = cur_Stack.expr.car
                cur_Stack.fn_found = T
            elif issymbol(cur_Stack.expr.car):
                cur_Stack.fn = cur_Stack.lookup(cur_stack.expr.car)
                cur_Stack.fn_found = True
            else:
                cur_Stack.receiving_fn = True
                cur_Stack = Stack(cur_Stack.expr.car,
                                  cur_Stack,
                                  Environment(cur_Stack.Env))
        elif not cur_Stack.has_args:
            if cur_Stack.receiving_arg:
                cur_Stack.args_pylist.append(ret)
                cur_Stack.arg_ptr = cur_Stack.arg_ptr.cdr
                cur_Stack.receiving_arg = False
            elif cur_Stack.arg_ptr == None:
                cur_Stack.has_args = True
            elif isform(cur_Stack.fn):    #### args are not eval'd
                cur_Stack.args_pylist.append(cur_Stack.args_ptr.car)
                cur_Stack.args_ptr = cur_Stack.args_ptr.cdr
            elif isatom(cur_Stack.args_ptr.car):  #### function territory
                cur_Stack.args_pylist.append(cur_Stack.args_ptr.car)
                cur_Stack.args_ptr = cur_Stack.args_ptr.cdr
            elif issymbol(...):
                ..... = (cur_Stack.Env.lookup(.....car))
                ..... = .....cdr
            else:
                cur_Stack.receiving_arg = True
                cur_Stack = Stack(cur_Stack.arg_ptr.car, cur_Stack, Environment(cur_Stack.Env))
```

```
class Stack():
    def __init__(self, expr):
        self.expr = expr
        self.find_fnc = T
        self.build_args = F
        self.call_fnc = F
        self.Env = Env
        self.address = 0
    def isdone(self):
        return ...
```

has_fn, fn_found, args_built, args=[], receiving_fn=F, receiving_arg=F

parent, Global Env

Stack.args

expr, fn, Env, parent

is_done(), has_fn, receiving_fn, has_args, args_pylist, args_ptr

if cur_Stack.receiving_fn:
cur_Stack.fn = ret
cur_Stack.receiving_fn = F
has_fn = T
if isclass instance?

def fn(cur_stack, Env):
end

(def (f x) (+ x 1))

Stacks do not track return values in any way, evals does that

Implicit do

eval never considers an atom

Stacks have children

single children!

eval always searches for next op

cur_stack

"(f (+ 1 1))"

text

(token interpretation)

symbol list (F (+ 1 1))

eval #'F #'+ 1 1

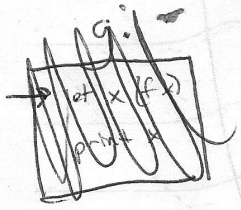
apply #'+ 1 1 = 2

arguments evaluated before new func. hits the stack

~~no callstack~~

eval builds errors using the callstack

NIL is None



function call

1) eval notices the cons
2) eval looks up the first element : a function
3) eval evals each argument and ~~collects~~ binds the values in a dropped env
4) eval puts the body of the func on the callstack
5) eval evals each expr in the body in turn
6) The last expr. is returned at #1

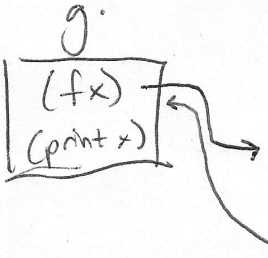
"hang call" "funstack" will be recursive?

not recursive

no

keep a pointer in each stacks to current inst.

new "block" references parent block



USE WHILE

also hang and drop

cur_block (= cur line)

hold by global stack

one ret if one block returns, and the parent is also done, ret is not overwritten