```
    (say (last (’ (a b c d e))))
    (say (last (’ (a))))
```

```
(define data (’ ((name tim)(age  53)(gemder  m))))
(say data)
(set! data (update (’ name) (’ george) data))
(say data)
(set! data (update (’ mood) (’ happy) data))
(say data)
```

```
#!/usr/bin/python
#Barry Martin
#Lab 9

################ Lispy: Scheme Interpreter in Python

## (c) Peter Norvig, 2010; See http://norvig.com/lispy.html

################ Symbol, Env classes

from __future__ import division

Symbol = str

class Env(dict):
    "An environment: a dict of {’var’:val} pairs, with an outer Env."
    def __init__(self, parms=(), args=(), outer=None):
        self.update(zip(parms,args))
        self.outer = outer
    def find(self, var):
        "Find the innermost Env where var appears."
        return self if var in self else self.outer.find(var)
    def depth(self):
        n = 0
        x = self.outer
        while x:
            n += 1
            x = x.outer
        return n

def add_globals(env):
    "Add some Scheme standard procedures to an environment."
    import math, operator as op
    env.update(vars(math)) # sin, sqrt, ...
    env.update(
     {’say’: lambda x: say(x), ’quit’ : goodbye,
      ’+’:op.add, ’−’:op.sub, ’*’:op.mul, ’/’:op.div, ’not’:op.not_,
      ’>’:op.gt, ’<’:op.lt, ’>=’:op.ge, ’<=’:op.le, ’=’:op.eq,
      ’equal?’:op.eq, ’eq?’:op.is_, ’length’:len, ’cons’:lambda x,y:[x]+y,
      ’car’:lambda x:x[0],’cdr’:lambda x:x[1:], ’append’:op.add,
      ’list’:lambda *x:list(x), ’list?’: lambda x:isa(x,list),
      ’null?’:lambda x:x≡[], ’symbol?’:lambda x: isa(x, Symbol),
      ’last’:lambda x:x[−1], ’update’:lambda key,val,l:update(key,val,l)})
    return env

def say(x): print x
def goodbye(): print ";; Bye."; quit()
def update(key,val,l):
    f=False
    for i in l:
        if i[0]≡key:
            i[1]=val
            f=True
    if f≡False:
        l.append([key,val])
    return l

global_env = add_globals(Env())

isa = isinstance

################ eval

def eval(x, env=global_env,lvl=0):
    "Evaluate an expression in an environment."
    this = x[0] if isa(x,list) else x
    if isa(x, Symbol):              # variable reference
        return env.find(x)[x]
    elif ¬ isa(x, list):          # constant literal
        return x
    elif x[0] ≡ ’load’:
      tmp=eval(x[1],env,lvl+1)
      return eload(tmp)
```

```
    elif  x[0] ≡ ’quote’ ∨  x[0] ≡ "’":
        (_, exp) = x
        return exp
    elif x[0] ≡ ’if’:             # (if test conseq alt)
        (_, test, conseq, alt) = x
        return eval((conseq if eval(test, env) else alt), env,lvl+1)
    elif x[0] ≡ ’set!’:           # (set! var exp)
        (_, var, exp) = x
        env.find(var)[var] = eval(exp, env,lvl+1)
    elif x[0] ≡ ’define’:         # (define var exp)
        (_, var, exp) = x
        env[var] = eval(exp, env,lvl+1)
    elif x[0] ≡ ’lambda’:         # (lambda (var*) exp)
        (_, vars, exp) = x
        return lambda *args: eval(exp, Env(vars, args, env),lvl+1)
    elif x[0] ≡ ’begin’:          # (begin exp*)
        for exp in x[1:]:
            val = eval(exp, env,lvl+1)
        return val
    else:                          # (proc exp*)
        head =x[0]
        exps = [eval(exp, env,lvl+1) for exp in x]
        print to_string(head),to_string(exps[1:])
        proc = exps.pop(0)
        #print ">calling", proc
        output = proc(*exps)
        print to_string(output)
        return output

################ parse, read, and user interaction

def read(s):
    "Read a Scheme expression from a string."
    return read_from(tokenize(s))

parse = read

def tokenize(s):
    "Convert a string into a list of tokens."
    return s.replace(’(’,’ ( ’).replace(’)’,’ ) ’).split()

def read_from(tokens):
    "Read an expression from a sequence of tokens."
    if len(tokens) ≡ 0:
        raise SyntaxError(’unexpected EOF while reading’)
    token = tokens.pop(0)
    if ’(’ ≡ token:
        L = []
        while tokens[0] ≠ ’)’:
            L.append(read_from(tokens))
        tokens.pop(0) # pop off ’)’
        return L
    elif ’)’ ≡ token:
        raise SyntaxError(’unexpected )’)
    else:
        return atom(token)

def atom(token):
    "Numbers become numbers; every other token is a symbol."
    try: return int(token)
    except ValueError:
        try: return float(token)
        except ValueError:
            return Symbol(token)

def to_string(exp):
    "Convert a Python object back into a Lisp−readable string."
    return ’(’+’ ’.join(map(to_string, exp))+’)’ if isa(exp, list) else str(exp)

def repl(prompt=’lis.py> ’):
    "A prompt−read−eval−print loop."
    print ";; LITHP ITH LITHTENING ...(v0.1)"
    while True:
        val = eval(parse(raw_input(prompt)))
        if val is ¬ None: print to_string(val)

def eload(f) :
  for part in  open(f):
    eval(parse(part))

import sys
if len(sys.argv) > 1:
  eload(sys.argv[1])
else:
  repl()
  quit()
```

```
say (((name tim) (age 53) (gemder m)))
[['name', 'tim'], ['age', 53], ['gemder', 'm']]
None
update (name george ((name tim) (age 53) (gemder m)))
((name george) (age 53) (gemder m))
say (((name george) (age 53) (gemder m)))
[['name', 'george'], ['age', 53], ['gemder', 'm']]
None
update (mood happy ((name george) (age 53) (gemder m)))
((name george) (age 53) (gemder m) (mood happy))
say (((name george) (age 53) (gemder m) (mood happy)))
[['name', 'george'], ['age', 53], ['gemder', 'm'], ['mood', 'happy']]
None
last ((a b c d e))
e
say (e)
e
None
last ((a))
a
say (a)
a
None
```