Lists & lists
lists.25

{nil}

A Familiar Place (door to N) N: White Room

White Room bookshelves, desk (computer), couch
glass box, genie (sleeping)
↳ "Break glass to wake owner"
Computer: green btn "RUN"; yellow btn "RESET"
break box; say yes

book "How to Program in Lisp"
1st problem: start up machine & define twentyseven to have value 27
ask genie to 'check' or 'repeat' problem
(define twentyseven 27)

punctuation ( ) ; : '

& push rem [:q to quit
:m for more
:? for other ":" commands]
:r Redisplay problem
:c Cancel expression
:g Force garbage collect
:e Display environment

2nd prob: Define values for cat and dog
so that cat & dog are equal? but not eqv?
Furthermore, cdr(cat) and cdr(dog) must be eqv?

(eqv? v w) Rets t if v & w are both nil, or are the same atom, or were created at the same time
(equal? v w) Rets t if v & w are eqv?, or are lists of the same length all of whose terms are equal?

~~pop 5~~
~~mom 5~~
~~cat (mom 27)~~
~~dog (pop 27)~~

~~(define greeting '(hi there))~~
~~(define alpha (list 'hi 'there))~~
mom 5
~~dad~~ (list (+ 1 4) ~~5~~ Sid)
list mom ~~dad~~ Sid

```
(define cat (list 'hi))
(define dog (list 'hi))
```

or

```
(define tail '(end))
(define cat (cons 'head tail))
(define dog (cons 'head tail))
```

3rd Define abs to be the absolute fn for integers

(abs 4) ⇒ 4
(abs -5) ⇒ 5
(abs 0) ⇒ 0

```
(define fred (lambda (n) (+ n 1)))
```

(n) = list of args; (+ n 1) = fn expression

```
(cond
  (test result)
  (test result)
)
```

```
(cond
  ((< x 0) (- 0 x))
  (t x)
)
```

4th Define sum to add a list of integers

eg: (sum '(8 2 3)) ⇒ 13
(sum nil) ⇒ 0

(length list)

```
(define sum (lambda args (cond
  ((if #args is ≤ 0) 0)
  ((if #args is 1) arg)
  (t (+ (car args) (sum (cdr args))))
)))
```

```
(define nil2zero (lambda (x) (cond
  ((equal? x nil) 0)
  (t x)
)))
```

(nil2zero nil) → 0
(nil2zero 5) → 5

```
(define sum (lambda (x) (cond
  ((null? x) 0)
  (t (+ (car x) (sum (cdr x))))
)))
```

~~(cond~~
~~(null? x) 0)~~
~~(t (+ (car args) (sum (cdr args))))~~
~~))~~

Next (megasum '((8) 5 (2 ()) (9 1) 3)) ⇒ 28

```
(define megasum (lambda (x) (cond
  ((null? x) 0)
  ...
  ((list? x) (+ (megasum (car x)) (megasum (cdr x))))
  (t x)
)))
```

Define max to return maximum of list of integers
eg. (max '(5 14 -3)) ⇒ 14

```
(define max (lambda (x) (cond
   ((= (length x) 1) (car x))
   ((> (car x) (max (cdr x))) (car x))
   (t (max (cdr x)))
)))
```

find: define pocket (pocket nil) ⇒ 8
(pocket #) ⇒ fn (that works like pocket, but # instead of 8.)

```
(define pocket (lambda (x) cond
   ((null? x) 8)
   (t (lambda (y) cond
      ((null? y) x)
```

```
(define mom (lambda (x y) cond
   ((null? x) y)
   (t (mom x x))
)))
(define pocket (lambda (x) (mom x 8))
```