# Cells in PicoLisp

```
Idea:     @tankf33der
Review:   @Regenaxer
Revision: 24

June 2020
CC0
```

# Fundamental overview

## CELL

```
+-----+-----+
| CAR | CDR |
+-----+-----+
```

**The PicoLisp reference says:**

1. A cell is a pair of machine words, which traditionally are called CAR and CDR in the Lisp terminology.

2. These words can represent either a numeric value (scalar) or the address of another cell (pointer).

3. All higher level data structures are built out of cells.

```
+-----+-----+
| CAR | CDR |
+-----+-----+
```

**The PicoLisp reference says:**

1. A cell is a pair of machine words, which traditionally are called CAR and CDR in the Lisp terminology.

2. These words can represent either a numeric value (scalar) or the address of another cell (pointer).

3. All higher level data structures are built out of cells.

```
// src/pico.h
typedef struct cell {
   struct cell *car;
   struct cell *cdr;
} cell, *any;
```

**Yes, two identical types**

```
+-----+-----+
| CAR | CDR |
+-----+-----+
```

**The PicoLisp reference says:**

1. A cell is a pair of machine words, which traditionally are called CAR and CDR in the Lisp terminology.

2. These words can represent either a numeric value (scalar) **or** the address of another cell (pointer).

3. All higher level data structures are built out of cells.

```
// src/pico.h
typedef struct cell {
   struct cell *car;
   struct cell *cdr;
} cell, *any;
```

Yes, two identical types

**Yes, can store identical values**

```
+-----+-----+
| CAR | CDR |
+-----+-----+
```

**The PicoLisp reference says:**

1. A cell is a pair of machine words, which traditionally are called CAR and CDR in the Lisp terminology.

2. These words can represent either a numeric value (scalar) or the address of another cell (pointer).

3. All higher level data structures **are built** out of cells.

```
// src/pico.h
typedef struct cell {
   struct cell *car;
   struct cell *cdr;
} cell, *any;
```

Yes, two identical types

Yes, can store identical values

**Yes, cells are everywhere**

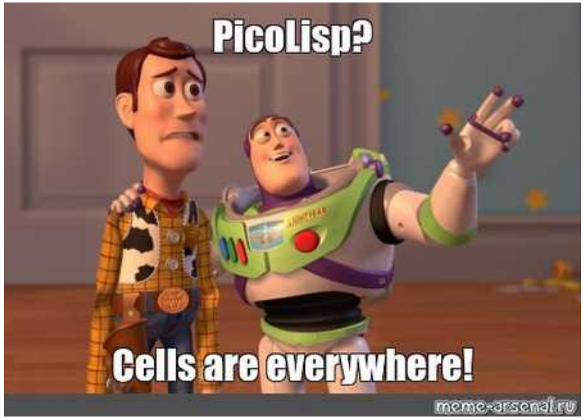
```
+-----+-----+
| CAR | CDR |
+-----+-----+
```

**The PicoLisp reference says:**

1. A cell is a pair of machine words, which traditionally are called CAR and CDR in the Lisp terminology.

2. These words can represent either a numeric value (scalar) or the address of another cell (pointer).

3. All higher level data structures are built out of cells.



```
// src/pico.h
typedef struct cell {
   struct cell *car;
   struct cell *cdr;
} cell, *any;
```

Yes, two identical types

Yes, can store identical values

Yes, cells are everywhere

```
// src/pico.h
typedef struct heap {
   cell cells[CELLS];
   struct heap *next;
} heap;
```

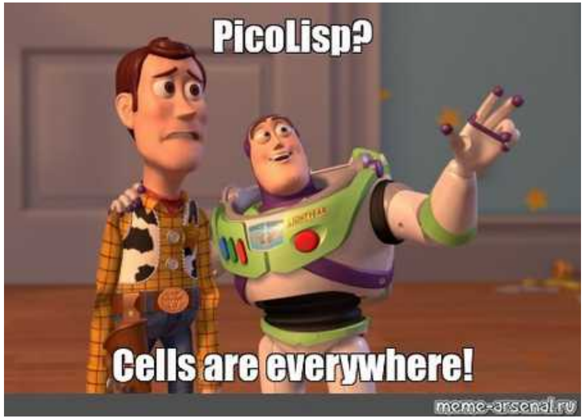
```
+-----+-----+
| CAR | CDR |
+-----+-----+
```

**The PicoLisp reference says:**

1. A cell is a pair of machine words, which traditionally are called CAR and CDR in the Lisp terminology.

2. These words can represent either a numeric value (scalar) or the address of another cell (pointer).

3. All higher level data structures are built out of cells.



```
// src/pico.h
typedef struct cell {
   struct cell *car;
   struct cell *cdr;
} cell, *any;
```

Yes, two identical types

Yes, can store identical values

Yes, cells are everywhere

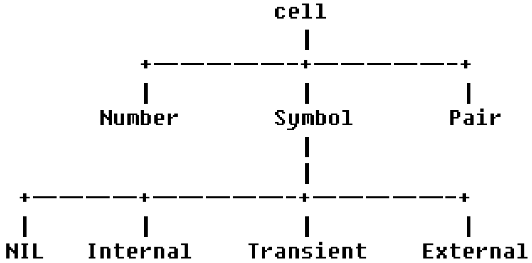
```
// src/pico.h
typedef struct heap {
   cell cells[CELLS];
   struct heap *next;
} heap;
```

Cells in heap under
full control by GC

```
                 cell
                  |
       +----------+----------+
       |          |          |
    Number     Symbol       Pair
                  |
                  |
 +------+---------+----------+
 |      |         |          |
NIL  Internal  Transient  External
```

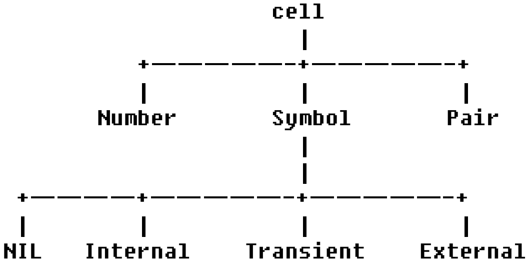
# Fundamental overview

## LIST



```
                     cell
                       |
           +-----------+-----------+
           |           |           |
       Number       Symbol      Pair
                       |
                       |
 +---------+-----------+-----------+
 |         |           |           |
NIL   Internal    Transient    External
```

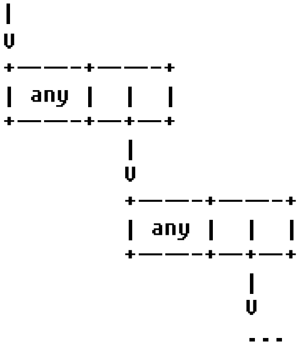
A list is not part of data type hierarchy.



```
                    cell
                      |
          +-----------+-----------+
          |           |           |
       Number       Symbol       Pair
                      |
                      |
 +--------+-----------+-----------+
 |        |           |           |
NIL   Internal    Transient    External
```

**The PicoLisp reference provides recursive definition:**

A list is a sequence of one or more cells (cons pairs), holding numbers, symbols, or cons pairs.



```
|
V
+-----+-----+
| any |  |  |
+-----+--+--+
         |
         V
         +-----+-----+
         | any |  |  |
         +-----+--+--+
                  |
                  V
                  ...
```

```
                 cell
                   |
         +---------+---------+
         |         |         |
      Number     Symbol     Pair
                   |
                   |
+------+---------+---------+
|      |         |         |
NIL  Internal  Transient  External
```

**The PicoLisp reference provides recursive definition:**

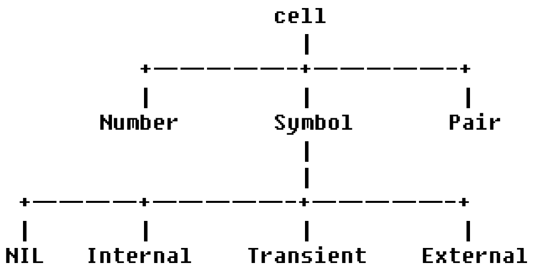
A list is a sequence of one or more cells (cons pairs), holding numbers, symbols, or cons pairs.

```
               cell
                |
       +--------+--------+
       |        |        |
    Number    Symbol    Pair
                |
                |
  +-------+---------+--------+
  |       |         |        |
 NIL  Internal  Transient  External
```

**The PicoLisp reference provides recursive definition:**

A list is a sequence of one or more cells (cons pairs), holding numbers, symbols, or cons pairs.

**List is like wagon train**



**The PicoLisp reference provides recursive definition:**

A list is a **sequence** of one or more cells (cons pairs), holding numbers, symbols, or cons pairs.

**Remember!**

This is a **list**
if CDR of last cell
points to NIL

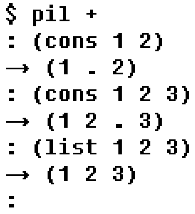
```
+---+---+    +---+---+
| 1 | --+--->| 2 |---+---> NIL
+---+---+    +---+---+
```

```
+---+---+
| 1 | 2 |
+---+---+
```

If atom in CDR then this is a dotted pair

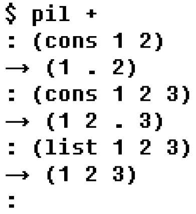
# Construct and view

```
$ pil +
: (cons 1 2)
→ (1 . 2)
: (cons 1 2 3)
→ (1 2 . 3)
: (list 1 2 3)
→ (1 2 3)
:
```



```
$ pil +
: (cons 1 2)
→ (1 . 2)
: (cons 1 2 3)
→ (1 2 . 3)
: (list 1 2 3)
→ (1 2 3)
:
```

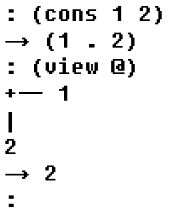
CONStruct a cell or sequence of cells are straightforward.



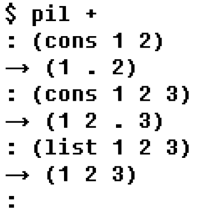
```
$ pil +
: (cons 1 2)
-> (1 . 2)
: (cons 1 2 3)
-> (1 2 . 3)
: (list 1 2 3)
-> (1 2 3)
:
```

Construct a cell or sequence of cells are straightforward.

Function **view** will help understand cell structure:



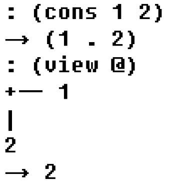
```
: (cons 1 2)
-> (1 . 2)
: (view @)
+-- 1
|
2
-> 2
:
```



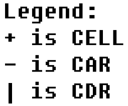
```
$ pil +
: (cons 1 2)
→ (1 . 2)
: (cons 1 2 3)
→ (1 2 . 3)
: (list 1 2 3)
→ (1 2 3)
:
```

Construct a cell or sequence of cells are straightforward.

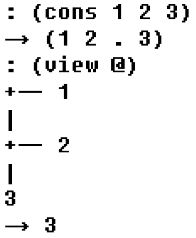
Function **view** will help understand cell structure:



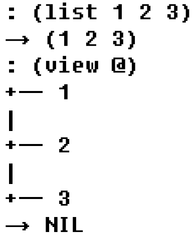
```
: (cons 1 2)
→ (1 . 2)
: (view @)
+-- 1
|
2
→ 2
```



```
Legend:
+ is CELL
- is CAR
| is CDR
```

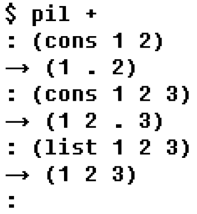


```
: (cons 1 2 3)
→ (1 2 . 3)
: (view @)
+-- 1
|
+-- 2
|
3
→ 3
```



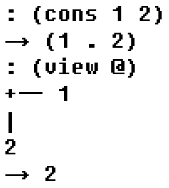
```
: (list 1 2 3)
→ (1 2 3)
: (view @)
+-- 1
|
+-- 2
|
+-- 3
→ NIL
```

Construct a cell or sequence of cells are straightforward.

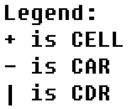


```
$ pil +
: (cons 1 2)
→ (1 . 2)
: (cons 1 2 3)
→ (1 2 . 3)
: (list 1 2 3)
→ (1 2 3)
:
```

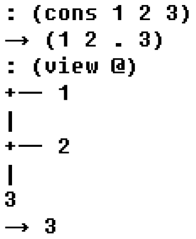
Function **view** will help understand cell structure:



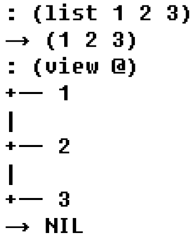
```
: (cons 1 2)
→ (1 . 2)
: (view @)
+-- 1
|
2
→ 2
```



```
Legend:
+ is CELL
- is CAR
| is CDR
```



```
: (cons 1 2 3)
→ (1 2 . 3)
: (view @)
+-- 1
|
+-- 2
|
3
→ 3
```

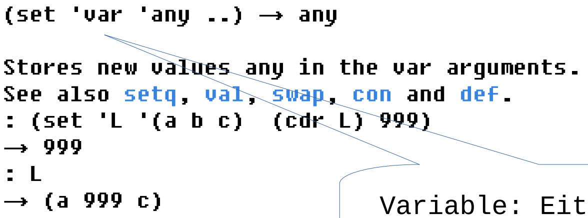


```
: (list 1 2 3)
→ (1 2 3)
: (view @)
+-- 1
|
+-- 2
|
+-- 3
→ NIL
```

After practice you will manipulate and view structures in mind.
Nothing special, right?

# Modify CAR

The PicoLisp reference for function **set** says:

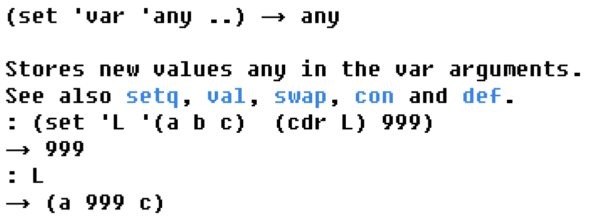


```
(set 'var 'any ..) → any

Stores new values any in the var arguments.
See also setq, val, swap, con and def.
: (set 'L '(a b c)  (cdr L) 999)
→ 999
: L
→ (a 999 c)
```

Variable: Either a symbol
or a cons pair

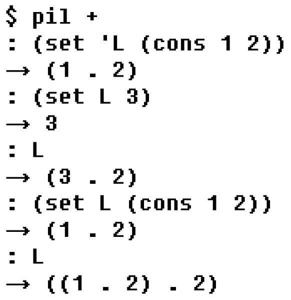
The PicoLisp reference for function **set** says:



```
(set 'var 'any ..) → any

Stores new values any in the var arguments.
See also setq, val, swap, con and def.
: (set 'L '(a b c)  (cdr L) 999)
→ 999
: L
→ (a 999 c)
```

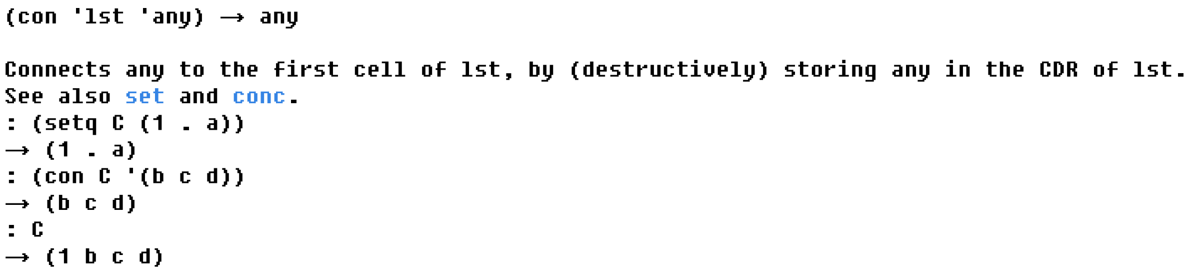
In case of cell it modify CAR:



```
$ pil +
: (set 'L (cons 1 2))
→ (1 . 2)
: (set L 3)
→ 3
: L
→ (3 . 2)
: (set L (cons 1 2))
→ (1 . 2)
: L
→ ((1 . 2) . 2)
```

# Modify CDR

The PicoLisp reference for function **con** says:

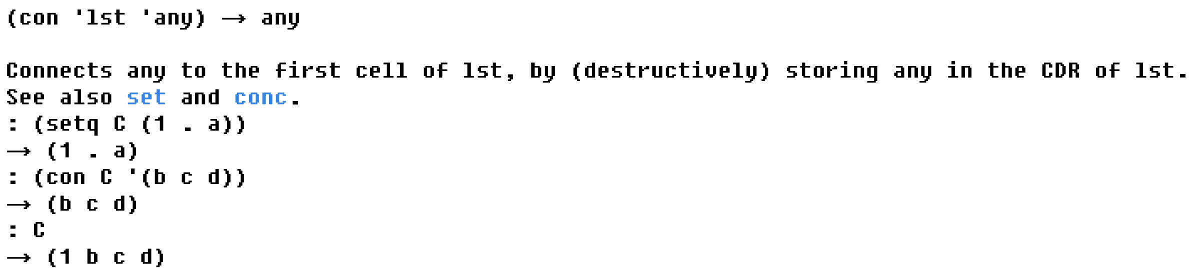


(con 'lst 'any) -> any

Connects any to the first cell of lst, by (destructively) storing any in the CDR of lst.
See also set and conc.

```
: (setq C (1 . a))
-> (1 . a)
: (con C '(b c d))
-> (b c d)
: C
-> (1 b c d)
```

The PicoLisp reference for function **con** says:



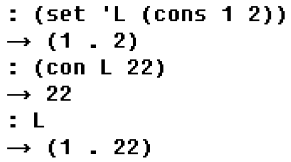
```
(con 'lst 'any) -> any

Connects any to the first cell of lst, by (destructively) storing any in the CDR of lst.
See also set and conc.
: (setq C (1 . a))
-> (1 . a)
: (con C '(b c d))
-> (b c d)
: C
-> (1 b c d)
```

**Remember:**

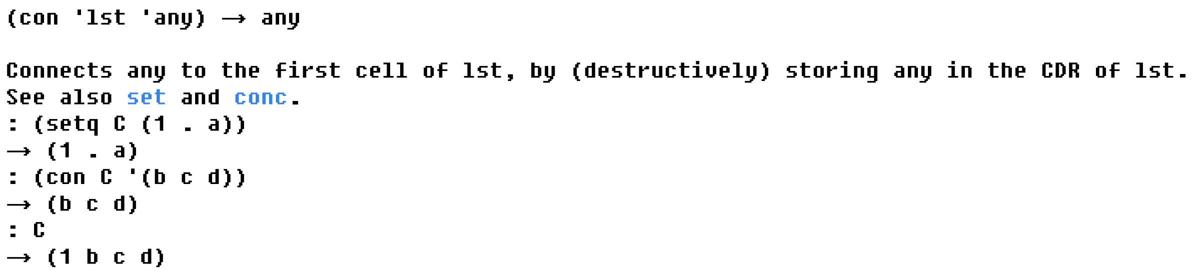
o) modify CDR of dotted pair is just modification

o) modify CDR of list is **DESTRUCTIVENESS** of sequence



```
: (set 'L (cons 1 2))
-> (1 . 2)
: (con L 22)
-> 22
: L
-> (1 . 22)
```

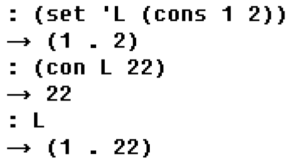
The PicoLisp reference for function **con** says:



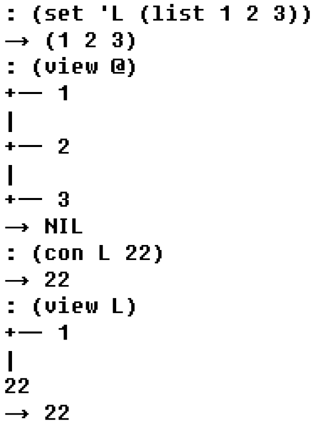
```
(con 'lst 'any) → any

Connects any to the first cell of lst, by (destructively) storing any in the CDR of lst.
See also set and conc.
: (setq C (1 . a))
→ (1 . a)
: (con C '(b c d))
→ (b c d)
: C
→ (1 b c d)
```

**Remember:**
o) modify CDR of dotted pair is just modification
o) modify CDR of list is **DESTRUCTIVENESS** of sequence

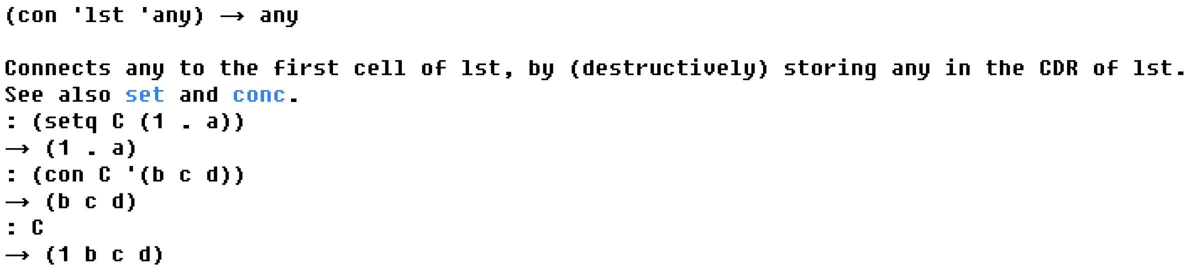


```
: (set 'L (cons 1 2))
→ (1 . 2)
: (con L 22)
→ 22
: L
→ (1 . 22)
```



```
: (set 'L (list 1 2 3))
→ (1 2 3)
: (view @)
+-- 1
|
+-- 2
|
+-- 3
→ NIL
: (con L 22)
→ 22
: (view L)
+-- 1
|
22
→ 22
```

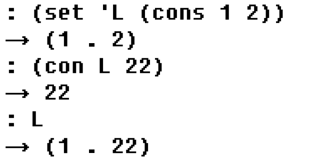
The PicoLisp reference for function **con** says:



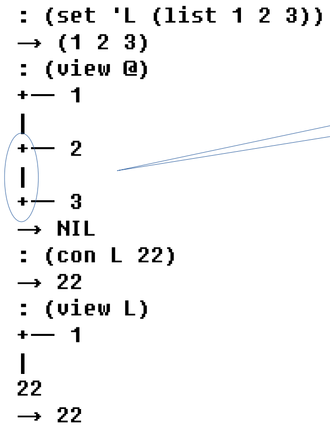
```
(con 'lst 'any) -> any

Connects any to the first cell of lst, by (destructively) storing any in the CDR of lst.
See also set and conc.
: (setq C (1 . a))
-> (1 . a)
: (con C '(b c d))
-> (b c d)
: C
-> (1 b c d)
```

**Remember:**
o) modify CDR of dotted pair is just modification
o) modify CDR of list is **DESTRUCTIVENESS** of sequence



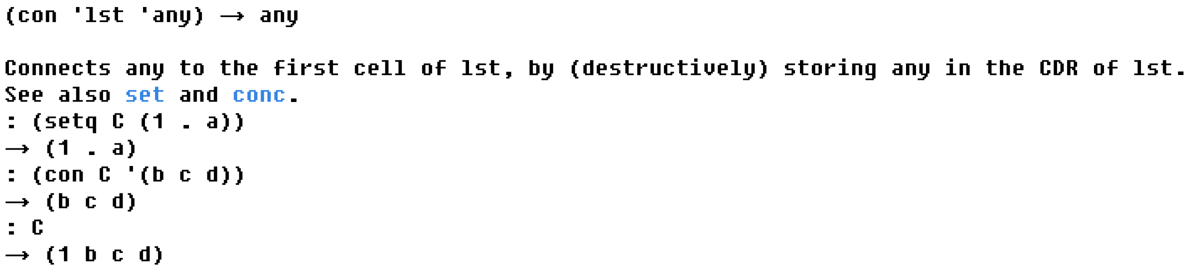
```
: (set 'L (cons 1 2))
-> (1 . 2)
: (con L 22)
-> 22
: L
-> (1 . 22)
```



```
: (set 'L (list 1 2 3))
-> (1 2 3)
: (view @)
+-- 1
|
+-- 2
|
+-- 3
-> NIL
: (con L 22)
-> 22
: (view L)
+-- 1
|
22
-> 22
```

Access path to two wagons is lost
and they will be GC eventually

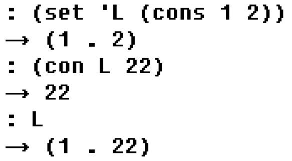
The PicoLisp reference for function **con** says:



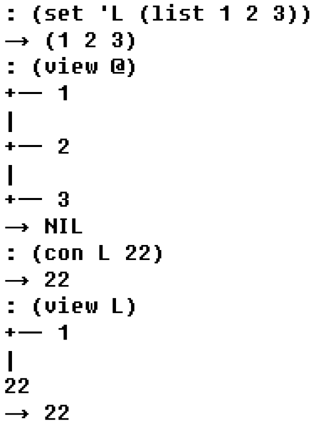
```
(con 'lst 'any) → any

Connects any to the first cell of lst, by (destructively) storing any in the CDR of lst.
See also set and conc.
: (setq C (1 . a))
→ (1 . a)
: (con C '(b c d))
→ (b c d)
: C
→ (1 b c d)
```

**Remember:**
o) modify CDR of dotted pair is just modification
o) modify CDR of list is **DESTRUCTIVENESS** of sequence



```
: (set 'L (cons 1 2))
→ (1 . 2)
: (con L 22)
→ 22
: L
→ (1 . 22)
```



```
: (set 'L (list 1 2 3))
→ (1 2 3)
: (view @)
+-- 1
|
+-- 2
|
+-- 3
→ NIL
: (con L 22)
→ 22
: (view L)
+-- 1
|
22
→ 22
```

Any destructive functions behaves the same way.
No dark corners anymore.

Now you have everything to understand listing of destructive function **chain**:

```
$ pil +
: (make (link 1 2) (view (made)) (chain 3) (view (made)))
+-- 1
|
+-- 2
+-- 1
|
+-- 2
|
3
-> (1 2 . 3)
: (make (link 1 2) (view (made)) (chain (cons 3)) (view (made)))
+-- 1
|
+-- 2
+-- 1
|
+-- 2
|
+-- 3
-> (1 2 3)
```

Happy coding!