```
fun interp(env::Env, store::Store, e::Expr) -> Vands
  | e-box(init) =>
    vsi = interp(env, store, init)
    ls = alloc(vsi.s, vsi.v)
    vs(v-box(ls.l), ls.s)
  | e-get-box(box) =>
    -> vsb = interp(env, store, box)
       cases(Value) vsb.v:
         | v-box(l) => vs(deref(vsb.s, l), vsb.s)
         | else => raise(not-a-box)
       end
```

```
data Vands:
  | vs(v::Val,
       s::Stor)
```

```
| e-get-box(
    box::Expr)
```

x = e

x := e

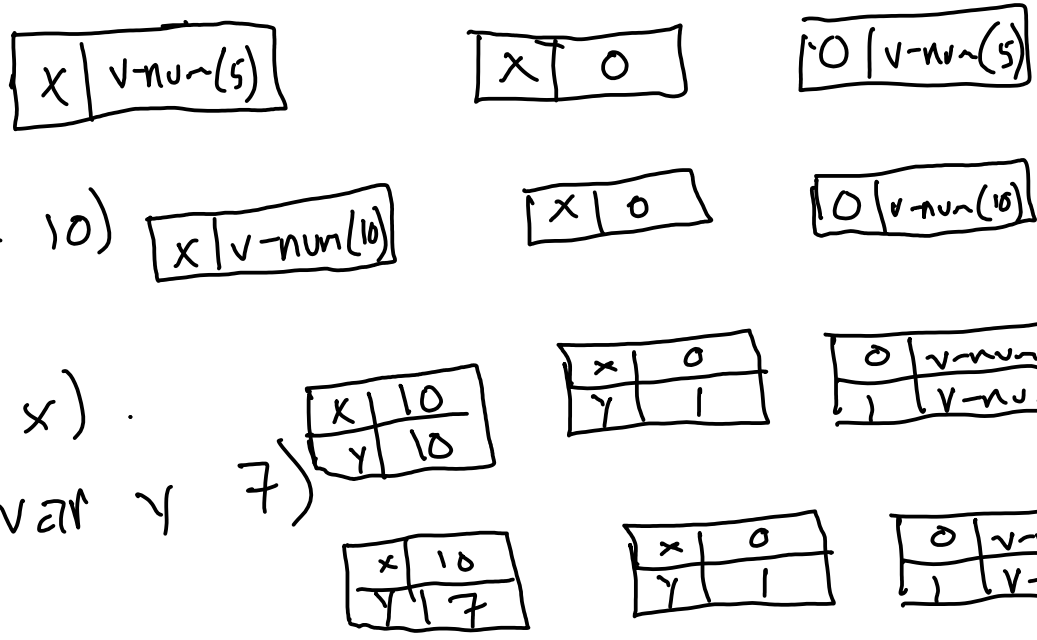
return = 7

5 + x = 7

<exp> = (set-box <exp> <exp>)

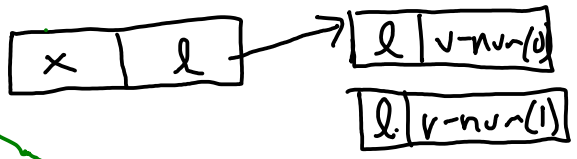
| (set-var <id> <exp>)

```
(let (x 5)
  (block
    (set-var x 10)
    x
    (let (y x)
      (set-var y 7)
      x))))
```



x = 5
x = 10
y = x
y = 7
x

```
(let (x 0)
  (let (f (lam ()
            (block
              (set-var x (+ x 1))
              x)))
```

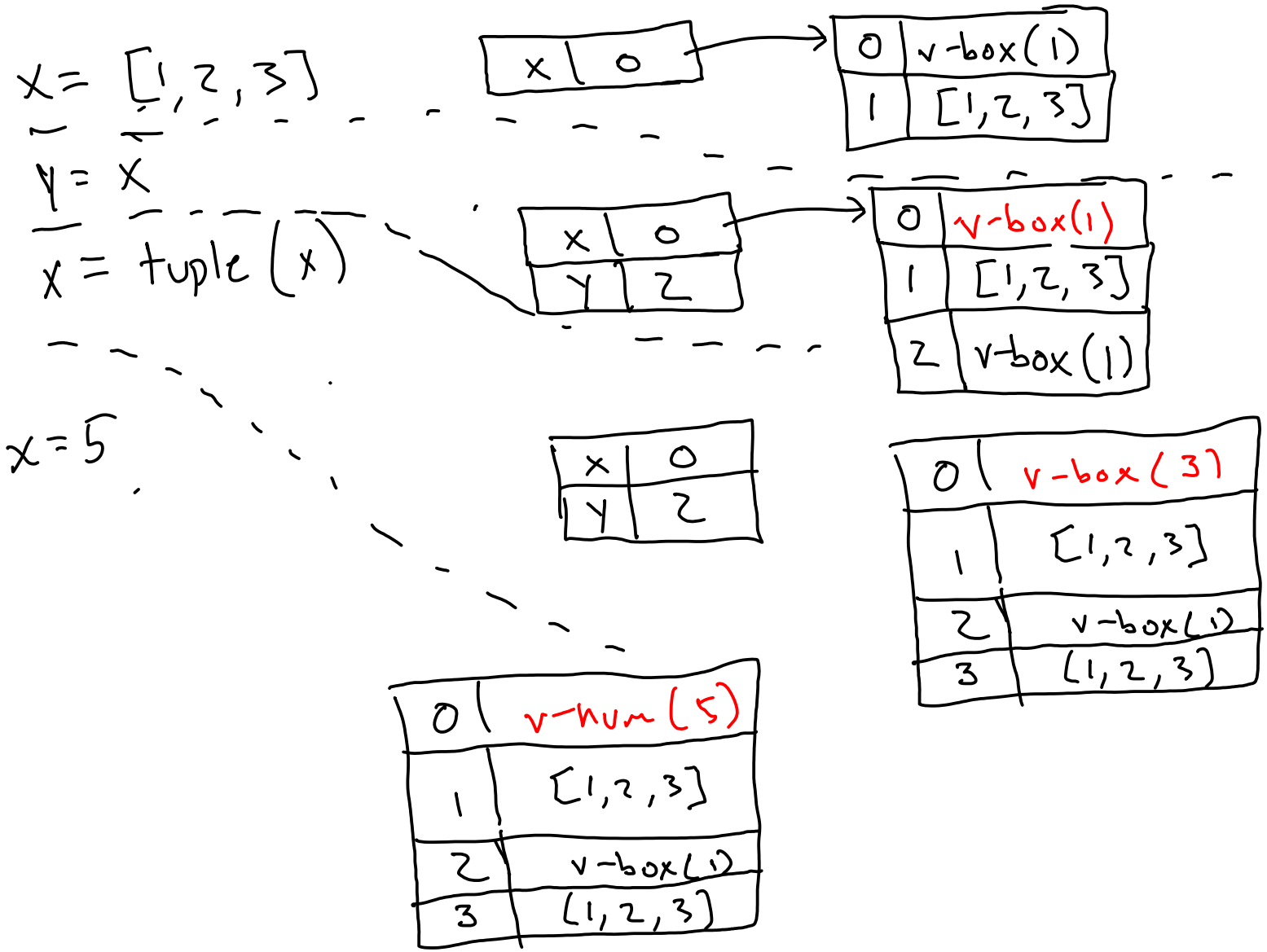



```
for i in range(0, 10):
    fs.append(lambda: i)
```

v-num(0)

```
(block
  (f) → 1
  (f) → 1, 2
  (f)))) → 1, 3
```

Functions aren't


x = [1,2,3]
y = x
x = tuple(x)
x = 5
x | 0
0 | v-box(1)
1 | [1,2,3]
x | 0
y | 2
0 | v-box(1)
1 | [1,2,3]
2 | v-box(1)
x | 0
y | 2
0 | v-box(3)
1 | [1,2,3]
2 | v-box(1)
3 | [1,2,3]
0 | v-num(5)
1 | [1,2,3]
2 | v-box(1)
3 | [1,2,3]

letrec

```
(let ((len2 0))
  (let ((length (lam (l)
                  (if (is-empty l)
                      0
                      (+ 1 (len2 (rest l)))))))
    (block
      (set-var len2 length)
      (length (link 1 (link 2 empty))))))
```