Exercise 1, Scheme (11 pts)

In the academic literature, there is a concept of pictures defined as rectangular arrays of symbols, e.g.

```
  abb
  bab
```

Such pictures can of course be memorized by row, as lists of list, e.g. the previous picture is `(a b b)(b a b))`. Consider the language L of pictures where symbols are from the set `{0,1}`, and are square pictures with 1 on the diagonal and 0 elsewhere (e.g. `((1 0 0)(0 1 0)(0 0 1)))`.

Define a procedure, called `genFig`, which takes a natural number n and returns the picture of L with side n.

Exercise 2, Haskell (12 pts)

Consider the language of pictures L as in Exercise 1. Define the `checkFig` function, which takes a list of lists p and returns `Just n`, where n is the side of p, if p is a member of L; Nothing otherwise. Write all the types of the defined functions.

Exercise 3, Prolog (8 pts)

Define a predicate which takes two lists L1 and L2 of numbers and a value v, and returns two other lists: one with the values of L1 and L2 that are less than v, the other with the values of L1 and L2 that are greater than v (the order does not matter). Values in L1 and L2 equal to v are discarded.

E.g.

```
?- arrange([1,2,3], [2,7,1,-5,8], 2, X, Y).
X = [1, 1, -5],
Y = [3, 7, 8].
```
Solutions

Scheme
(define (genRow len pos)
  (let loop ((v '())
    (k 0))
    (if (< k len)
      (loop (cons (if (= pos k) 1 0) v)
        (+ k 1))
      v)))

(define (genFig n)
  (let loop ((f '())
    (k 0))
    (if (< k n)
      (loop (cons (genRow n k) f)
        (+ k 1))
      f)))

Haskell
checkOne :: [Int] -> Int -> Int -> Bool
checkOne ls pos len = checkOne' ls pos len 0 where
  checkOne' [] _ len _ = len == v
  checkOne' (1:es) x len x' = x == x' && checkOne' es x len (x'+1)
  checkOne' (0:es) x len x' = x /= x' && checkOne' es x len (x'+1)
  checkOne' _ _ _ _ = False

checkFig :: [[Int]] -> Maybe Int
checkFig fig = if checkFig' fig (length fig) 0 then Just len else Nothing where
  checkFig' [] _ _ = True
  checkFig' (r:rs) len p = checkOne r p len && checkFig' rs len (p+1)

Prolog
part([X|L],Y,[X|L1],L2) :- X < Y, !, part(L,Y,L1,L2).
part([X|L],Y,L1,[X|L2]) :- X > Y, !, part(L,Y,L1,L2).
part([X|L],X,L1,L2) :- !, part(L,X,L1,L2).
part([],_[],_[]).

arrange(L1, L2, V, S1, S2) :- part(L1,V,S11,S12),
  part(L2,V,S21,S22),
  append(S11,S21,S1),
  append(S12,S22,S2).