**Exercise 1, Scheme (8 pts)**

Define a `<*>` operator for lists, defined as in Haskell’s Applicative Functors. 
E.g. 
```scheme
<*>(list (lambda (x) (+ 1 x)) (lambda (x) (* 2 x)))
'(1 2 3)
```

is the list `'(2 3 4 2 4 6)`.

**Exercise 2, Haskell (13 pts)**

Given a list of lists, define a function `transpose` that returns a list containing: a list of all the first elements, then a list of all the second elements, and so on. Lists can be assumed non empty, but can be of different lengths. Write all the types of the defined functions.

E.g. 
```
transpose [[1,2],[3],[4,5,6]]
```

is the list `[[1,3,4],[2,5],[6]]`.

**Exercise 3, Prolog (10 pts)**

Define a predicate that can be used to get all the “atomic” elements of a term.

E.g. 
```
?- determ(1+2*3, X).
```

X = (+) ;
X = 1 ;
X = (*) ;
X = 2 ;
X = 3 ;
false.
Solutions

Scheme
(define (concatmap f ls)
  (foldr append '() (map f ls)))

(define (<*> fs xs)
  (concatmap (lambda (f) (map f xs)) fs))

Haskell
transpose :: [[a]] -> [[a]]
transpose [] = []
transpose ls = let hs = map head ls
               ts = filter (not . null) $ map tail ls
               in hs : transpose ts

Prolog
determ(T, T) :- atomic(T), !.
determ(T, Y) :- T =.. [X|Xs], (Y = X ; deterl(Xs, Y)).

deterl([], _) :- !, fail.
deterl([X|Xs], Y) :- determ(X,Y) ; deterl(Xs,Y).