Exercise 2.37.

Suppose we represent vectors $v = (v_i)$ as sequences of numbers, and matrices $m = (m_{ij})$ as sequences of vectors (the rows of the matrix). For example, the matrix

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 4 & 5 & 6 & 6 \\ 6 & 7 & 8 & 9 \end{bmatrix}$$

is represented as the sequence `((1 2 3 4) (4 5 6 6) (6 7 8 9))`. With this representation, we can use sequence operations to concisely express the basic matrix and vector operations. These operations (which are described in any book on matrix algebra) are the following:

`(dot-product` v w`)`	returns the sum $\sum_i v_i w_i$;
`(matrix-*-vector` m v`)`	returns the vector t, where $t_i = \sum_j m_{ij} v_j$;
`(matrix-*-matrix` m n`)`	returns the matrix p, where $p_{ij} = \sum_k m_{ik} n_{kj}$;
`(transpose` m`)`	returns the matrix n, where $n_{ij} = m_{ij}$.

We can define the dot product as[1]

```
(define (dot-product v w)
  (accumulate + 0 (map * v w)))
```

Fill in the missing expressions in the following procedures for computing the other matrix operations. (The procedure `accumulate-n` is defined in exercise 2.36.)

```
(define (matrix-*-vector m v)
  (map <??> m))

(define (transpose mat)
  (accumulate-n <??> <??> mat))

(define (matrix-*-mantrix m n)
  (let ((cols (transpose n)))
    (map <??> m)))
```

Answer.

A little linear algebra shows that the product of a matrix and a vector is another vector. And the first element of the resulting vector is the dot-products of the fist row of the matrix and the vector. The second element is obtained by multiplying the second row of the matrix by the vector, and so on. In other words, we takes a sequence of all the rows in the matrix and factor each by the vector to produce the product. This idea can be capture by `map`ping a procedure which computes the dot-product of each row in the matrix and the vector

```
(define (matrix-*-vector m v)
  (map (lambda (i)
         (dot-product v i))
       m))
```

Now, consider the problem of transposing a matrix. A usual practice would be by turing the first column of the original matrix into the first row of the transposed matrix, and the second column, and so on. This is in fact equivalent to combining all the first elements of the sequences into a sequence, all the second elements into the subsequent sequence, and so on, and returns a sequence of sequences. Notice that we can use `accumulate-n` to depict this strategy by specifying a procedure which composes a sequence and setting its initial value

1. This definition uses the extended version of `map` described in footnote 12.

```
(define (transpose mat)
  (accumulate-n (lambda (x y) (cons x y))
                nil
                mat))
```

Additionally, we know mathematically that the product of two matrices M and N is still a matrix. An entry m_{ij} in the product $M\,N$ is the *dot-product* of the ith row of M and the jth column of N. This idea can be expressed in a different way in which we first obtian N^T, the transposition of matrix N, then multiply N^T by each row of M to obtain a sequence of sequences, and finally combine these sequences into the resulting matrix. And we do this by `map`ping a procedure which computes the product of N^T by each row of M

```
(define (matrix-*-matrix m n)
  (let ((cols (transpose n)))
    (map (lambda (v)
           (matrix-*-vector cols v))
         m)))
```