1. In this exercise you should implement some vector- and matrix operations. All functions should not only support numbers as entries but also symbols. Use functional programming, i.e. no side effects on parameters. Error handling and parameter checking can be left out for this exercise. All vector and matrix functions except matrix multiplication are one-liners.
   
   (a) Implement the functions `sym-add` and `sym-mult`. They should add up or multiply an arbitrary number of arguments. The functions should take numbers as well as symbols and Numbers should be simplified as much as possible. E.g. `(sym-add 1 2 'a 2 'b)` results in `( + 6 a b)` and `(sym-mult 1 2 3)` has the result 6.

   (b) Vectors are represented by lists. Implement the following functions and use the functions defined in exercise 1a:
   
   i. `v-add`: Add an arbitrary number of vectors.
   ii. `v-factor`: Multiply a vector with a scalar number.
   iii. `v-dot`: The dot product of two vectors.

   (c) Implement the following matrix operations. Matrices are represented as nested lists. Each row in the matrix should be a list and the matrix is a list of rows. Example:

   \[
   \begin{pmatrix}
   1 & 2 \\
   3 & 4 \\
   \end{pmatrix}
   \rightarrow
   '((1 2) (3 4))
   \]

   i. `m-add`: Add an arbitrary number of matrices.
   ii. `m-factor`: Multiply a matrix with a scalar number.
   iii. `m-transpose`: Return the transposed matrix.
   iv. `m-multiply`: Multiply two matrices.

2. Now implement a simple relational database that supports simplified SQL-like queries. Use the file `database.lisp` and fill in the required code parts.

   (a) Implement the following basic interface functions.
   - `make-table`
   - `get-schema`
   - `get-extension`
   - `get-table`
   - `add-table`
   - `drop-table`
   - `replace-table`

   (b) Now implement more complex database functions. Use only the functions defined in 2a to access the database.

   i. Implement the function `(defun join (table-1 table-2) ...)` that calculates the cross product of two tables. Don’t forget that a table consists of a schema and an extension.
   ii. Implement the function `multiple-join` that calculates the cross-product of an arbitrary number of tables.
   iii. Implement the function `(defun map-columns-to-positions (columns schema) ...)` that returns the indices of columns in a schema definition. Example:

   \[
   \text{map-columns-to-positions '(s-skey s-sname sp-pkey)}
   \Rightarrow (1 4)
   \]
   iv. Implement the function `project`.

3. Use the functions implemented in 2 to solve the following tasks:
(a) Insert the tables shown in Table 1 into an empty database.
(b) Make the following queries:
   - Which supplier (name) lives in a Bavarian city (i.e., either Neu-Ulm or Munich)?
   - Which supplier names are equal to their product names? Where does he live?
   - Which suppliers provide products that cost less than 120? Query the name of the supplier, the product and the price.
(c) Remove the product *Maultaschen* from the table *Product-Names*.
(d) Remove all tables but the table *Product-Names* from the database.