1. Assignment (10 points)

For the first assignment please form groups of two students. Groups of three students are allowed if no further group of two can be formed. The first assignment is due in week 6. You have to hand in

1. a signed academic honesty form
2. printed source code for all solved tasks with plenty of remarks, i.e., documentation should be in form of remarks – and don't be shy with it.

before demonstrating your code. Each group needs to demonstrate their code in less than 5 minutes. Please prepare the demonstration carefully so that you can demonstrate in less than 5 minutes. No code changes during the demonstration are allowed. Note that everyone in the group will receive the same mark. Make sure that everyone in the group contributes equally.

In this assignment you will implement a simple spreadsheet system either in LISP or SCHEME. A spreadsheet is represented as list of rows, i.e.,

$$(\text{row}_1 \ \ldots \ \text{row}_n) \tag{1}$$

A row is an association list that contains association pairs, i.e.,

$$((\text{key}_1 \, \text{data}_1) \ldots (\text{key}_n \, \text{data}_n)) \tag{2}$$

The keys in an association list represent specific columns (e.g., A-Z) in a row and the associated data entries represent their values. For sake of simplicity we assume that data entries in an association list are integer values only. Another assumption is that we have for one key at most one pair in a row. For example the following spreadsheet describes the speed and fuel consumption of vehicles:

speed	fuel
120	5
40	100
300	20
500	n/a

This table is translated to following list

```
(setq spreadsheet '(
 ((A 120) (B 5))
 ((A 40) (B 100))
 ((A 300) (B 20))
 ((A 500)))
)
```

Note the last entry in the table has no defined value for attribute fuel. Therefore, there is no pair in the association list for B in the 4th row.

For a spreadsheet we would like to formulate a query. This query selects rows of the input spreadsheet and outputs it to a new spreadsheet. A row of the input spreadsheet is put into the result spreadsheet if the predicate holds for the row. A query is given in form of a recursively defined list. The recursive list is an element of the following inductively defined set Q:

1. If a and b in Q then, (`s-and` $a\,b$) is in Q.
2. If a and b in Q then, (`s-or` $a\,b$) is in Q.
3. If a in Q then, (`s-not` a) is in Q.
4. If *rel* is one of the following symbols: `s-eq, s-le,s-ge, s-leq, s-geq, s-diff`; and *attr* is a column key, and x is an integer value, than ($rel\,attr\,x$) is in Q.
5. Nothing else is in Q.

For example the following list `(s-and (s-geq B 5) (s-le B 100) )` is in Q and semantically models the query: $fuel \geq 5 \wedge fuel < 100$.

The symbols `s-or`, `s-and`, and `s-not` represent the standard boolean operations. The semantics of the relations can be found in the following table:

Symbol	Semantics
s-eq	$=$
s-le	$<$
s-ge	$>$
s-leq	$\leq$
s-geq	$\geq$
s-diff	$\neq$

1. Task (3 points)

Write a function `(find-columns <query>)` that recursively traverses a query and returns a list of columns that are used in the query. For example

```
(find-columns '(s-and (s-geq A 5) (s-le A 100)))
```

should return `(A)` because `A` is the only column in the above query. Do not allow multiple occurrences of columns. Assume that the recursive list `<query>` is always an element of Q.

2. Task (7 points)

This task is more difficult: Write a function `(transformer <query>)` that takes a query as an argument and generates a *function* as output. The generated function has a row as input and returns `T` (for true) if the predicate of the query holds, otherwise it returns `nil` (for false). For example

```
>(setq tq (transformer '(s-and (s-geq B 5) (s-le B 100)))))
...
>(#tq '((A 300) (B 20)))
T
>(#tq '((A 40) (B 100)))
nil
```

With the transformer function you can write a concise row selection in LISP that selects all rows for which the predicate of the query holds, i.e.,

```
(mapcar #tq spreadsheet)
```

The execution of above query would return following rows

```
(
 ((A 120) (B 5))
 ((A 300) (B 20))
)
```

Hint: Start simple. Try to translate simple relations and from there you construct lambda terms for "not", "and", and "or".

3. Task (0 points)

This task is not taken into account for marking. However, keen students might want to attempt the third task. Try to simplify queries. Below you find example rules, but come up with your own rule set – or better can you find a proof that your rules are optimal (very very hard)!

1. Eliminate all "s-not" clauses by using De-Morgan's law and for each relation there exists its negated relation.