1. (5 points) First-order Predicate Calculus. In theory, at least.

Consider first-order theory (program) $T$.

a. Is it possible that $T \models student(parke)$ and $T \not\models \neg student(parke)$?
   If it is possible, provide an example $T$ such that it is the case. Otherwise, explain why it is not possible.

b. Is it possible that $T \not\models student(parke)$ and $T \not\models \neg student(parke)$?
   If it is possible, provide an example $T$ such that it is the case. Otherwise, explain why it is not possible.

Consider a datalog database $D$.

c. Is it possible that $D \models \neg student(parke)$?
   Why or why not?

d. Do $D \vdash \neg student(parke)$ and $D \not\models student(parke)$ mean the same thing?
   Why or why not?

e. Is every first-order propositional theory (“program”) equivalent to some CNF propositional theory (“program”)?
   If no, show an example of a first-order theory that cannot be written in CNF. If yes, are there any drawbacks to representing first-order theories in CNF?
2. (5 points) **Rules. Bar-flies.** (Thanks to Zahir Tari.)

Suppose that we have the following predicates.

- `frequents (Drinker, Bar)`: The drinker frequently visits this bar.
- `serves (Bar, Beer)`: The bar serves this type of beer.
- `likes (Drinker, Beer)`: The drinker likes this type of beer.

Define the following predicates via rules using the predicates above (and any that you define).

a. `happy (D)`: The drinker `D` frequents at least one bar which serves a beer that he / she likes.

b. `very_happy (D)`: Every bar that the drinker `D` visits serves at least one beer he / she likes.

c. `should_visit (D, B)`: The bar `B` serves at least one beer that the drinker `D` likes.

d. `sad (D)`: No bar that the drinker `D` visits serves a beer that he / she likes.

e. `very_sad (D)`: No bar serves a beer that he / she likes.

You may assume that each drinker at least frequents one bar. Make certain that your rules are safe.
3. (5 points) **Queries in Datalog & Datalog¬. Enrol now in Datalog U.!**

**Exercise**

Consider the following schema.

\[
\begin{align*}
\text{student}(s\#, \text{sname}, \text{dob}, d\#) \\
\text{FK}(d\#) \text{ refs dept} & \quad \text{// Student’s major} \\
\text{prof}(p\#, \text{pname}, d\#) \\
\text{FK}(d\#) \text{ refs dept} & \quad \text{// Professor’s home department} \\
\text{dept}(d\#, \text{dname}, \text{building}, p\#) \\
\text{FK}(p\#) \text{ refs prof} & \quad \text{// Department’s chair} \\
\text{course}(d\#, \text{no}, \text{title}) \\
\text{FK}(d\#) \text{ refs dept} & \quad \text{// Course offered by this department} \\
\text{class}(d\#, \text{no}, \text{term}, \text{year}, \text{section}, \text{room}, \text{time}, p\#) \\
\text{FK}(d\#, \text{no}) \text{ refs course} & \quad \text{// Class is an offering of this course} \\
\text{FK}(p\#) \text{ refs prof} & \quad \text{// Instructor of class} \\
\text{enrol}(s\#, d\#, \text{no}, \text{term}, \text{year}, \text{section}, \text{grade}) \\
\text{FK}(s\#) \text{ refs student} & \quad \text{// This student is enrolled in} \\
\text{FK}(d\#, \text{no}, \text{term}, \text{year}, \text{section}) \text{ refs class} & \quad \text{// this class}
\end{align*}
\]

‘FK’ above stands for *foreign key*. These indicate foreign-key constraints in the schema.

Write the following queries in Datalog (and Datalog¬). You may use auxiliary predicates and rules. (You may reuse auxiliary predicates and rules in following sub-questions.)

A common convention is to use ‘\_’ as a variable name when the variable is unimportant for the query; e.g., `class(D, N, \_, \_, \_, \_, \_, \_)`. By convention, two occurrences of ‘\_’ are different variables and may take on different values (even though they seem to have the same “name”). You may find this convention useful.

Be careful that all your rules are *safe*, including rules that you write that use negation.

a. Which students have taken some course twice?

b. Which students have taken a course with a department chair?

Note that a professor may teach classes outside of his or her department. Also note that a student may take classes in a department outside of his or her major’s department.

c. Which students have never taken a course in his or her major (\text{dept})?

d. Which students have taken all of the courses offered by a department?

e. Which students have taken at least five courses in their major (\text{dept})?

You shall need to use arithmetics (e.g., ‘\#’, ‘\<\’) here. Assume that course numbers (\text{no}) can be compared; e.g., \text{M} < \text{N}. Use the predicate \text{is} to equate numbers; e.g., \text{J} is \text{I} + 1.
4. **Datalog Modeling. As easy as rolling off a log.** (5 points)

The puzzle *Sū Doku*—or just *sudoku*—is to fill in the blank cells of a 9×9 matrix with the numerals 1, . . . , 9 such that no row has the same numeral twice, no column has the same numeral twice, and no *block* has the same numeral twice. The 9×9 matrix is tiled by nine 3×3 matrices, each called a block.

A typical sudoku puzzle has some of the cells already filled in (the *givens*) so that there exists exactly one solution. For example,

```
5 3 | 7 |
6   | 1 9 5 |
9 8 | 6 |

8   | 6 3 |
4   | 8 3 1 |
7   | 2 6 |
6   | 2 8 |
4 1 9 | 5 |
8   | 7 9 |
```

with the solution shown on the right.

Write a Datalog program for sudoku. Let each cell in the sudoku matrix be represented by a variable:

```
X₀ X₁ X₂ X₃ X₄ X₅ X₆ X₇ X₈
X₉ X₁₀ X₁₁ X₁₂ X₁₃ X₁₄ X₁₅ X₁₆ X₁₇
X₁₈ X₁₉ X₂₀ X₂₁ X₂₂ X₂₃ X₂₄ X₂₅ X₂₆
X₂₇ X₂₈ X₂₉ X₃₀ X₃₁ X₃₂ X₃₃ X₃₄ X₃₅
X₃₆ X₃₇ X₃₈ X₃₉ X₄₀ X₄₁ X₄₂ X₄₃ X₄₄
X₄₅ X₄₆ X₄₇ X₄₈ X₄₉ X₅₀ X₅₁ X₅₂ X₅₃
X₅₄ X₅₅ X₅₆ X₅₇ X₅₈ X₅₉ X₆₀ X₆₁ X₆₂
X₆₃ X₆₄ X₆₅ X₆₆ X₆₇ X₆₈ X₆₉ X₇₀ X₇₁
X₇₂ X₇₃ X₇₄ X₇₅ X₇₆ X₇₇ X₇₈ X₇₉ X₈₀
```

Do not use negation, arithmetics (e.g., “/=”, “<”), or function symbols (which are not in Datalog proper anyway).

One predicate should be *sudoku* that takes arguments *X₀, . . . , X₈₀*. For a given puzzle, one could then query for the solution; e.g.,

```
← sudoku (5, 3, X₂, X₃, . . . , 9).
```

Note that your sudoku “program” need not be efficient in any way. It just needs to specify logically and correctly the problem. So try to keep it quite simple. You may use ellipses (e.g., “…” , “;”) where appropriate and when easily understood to make your answer briefer.

**Hint:** Be clever in defining the permutations of 1, . . . , 9.