Exercise 2:
Propositional Logic Semantics

In this exercise, you are asked to assign semantics to propositional logic formulae. Specifically, most of the exercise is focused on taking a propositional formula and a model for it, where the model is assignment of a truth value to each atomic proposition in the formula, and evaluating the truth of the formula in that model.

Definition. Let ϕ be a propositional formula, and let S be the set of variables in it. A **model** M for ϕ is a function that assigns a truth value to every variable in S. I.e., $M : S \to \{True, False\}$.

Given a propositional formula ϕ and a model M for it, we define the truth value of the formula ϕ in the model M inductively in the natural way (using the standard interpretation of the Boolean operators *not*, *and*, and *or* and the Boolean constants *True* and *False*). The file `propositions/semantics.py` contains a list of functions that you are asked to implement for dealing with this. A formula is represented as an instance of the class `Formula` that was defined in Exercise 1. We will represent a model as a Python `dict` (dictionary) that maps every variable name to a Boolean value.

Task 1. Implement the missing code for the function `evaluate(formula, model)`, which returns the truth value of the given formula in the given model. For example, taking the formula `f` that represents '~(p&q7)' and the model `{'p': True, 'q7': False}`, this function should return the Boolean value `True`.

The **truth table** of a given formula simply lists the truth value of the formula for each possible model for the formula. Before starting to deal with truth tables, we need to be able to list all the possible models over a given set of variables.

Task 2. Implement the missing code for the function `all_models(variables)`, which takes a list of variable names and returns an iterator over all possible models over the variables in the list.[1] The order of the models is lexicographic according to the order of the variables in the given list, where `False` precedes `True`. Thus, for example, `all_models(['p', 'q'])` should return an iterator over the four models `{'p': False, 'q': False}`, `{'p': False, 'q': True}`, `{'p': True, 'q': False}`, `{'p': True, 'q': True}`, in this order.

Task 3. Implement the missing code for the function `truth_values(formula, models)`, which returns the list of respective truth values of the given formula in the models

[1] One way to solve this task is to construct the requested iterator using a generator, while another is to construct it by returning `iter(list)` where `list` is a list of the models to iterate over. While some may find the latter conceptually simpler to implement, for a large number of variables it consumes a lot of memory (2^n memory, where n is the number of variables) compared to the former (which consumes n memory). (Iterating over all models in any implementation would take 2^n time.)

iterated over by the given iterator. For example, `truth_values(f, all_m)`, where `f` is the formula that represents '~(p&q7)' and `all_m` is the iterator that was returned in the example from Task 2, should return `[True, True, True, False]`.

Task 4. Implement the missing code for the function `is_tautology(formula)`, which returns whether the given formula is a **tautology**, i.e., whether it has a true value in every model. For example, `is_tautology(f)` where `f` is the formula that represents '~(p&q7)' should return `False`, while `is_tautology(g)` where `g` is the formula that represents '(x|~x)' should return `True`.

Task 5. Implement the missing code for the function `print_truth_table(formula)`, which prints the truth table of the given formula according to the format used in the following example: `print_truth_table(f)`, where `f` is the formula that represents '~(p&q7)', should print:

```
| p | q7 | ~(p&q7) |
|---|----|---------|
| F | F  | T       |
| F | T  | T       |
| T | F  | T       |
| T | T  | F       |
```

The columns should be sorted alphabetically by the names of the variables.

All of the tasks so far culminated in Task 5, which performed the task of taking a formula and producing its truth table. The next two tasks implement the "reversed" task, which is somewhat surprisingly also possible, of taking a given truth table, and creating — synthesizing — a propositional formula with this truth table. Our first step will be to create a formula whose truth table has a single row with value *True* (with all other rows having value *False*).

Task 6. Implement the missing code for the function `synthesize_for_model(model)`, which returns a propositional formula that has a true value for the given model and a false value for every other model over the same variables. You may assume that the given model contains at least one variable.

Task 7. Implement the missing code for the function `synthesize(models, values)`, which returns a propositional formula that has the given list of truth values in the given list of models. Thus, for example, `synthesize(all_models(['p', 'q7']), [True, True, True, False])` should return a formula that is equivalent to '~(p&q7)' in the sense of having the same truth table. You may assume that all of the given models are over the same nonempty set of variables. Hint: use the function `synthesize_for_model` that you implemented in Task 6.