CSE 015: Discrete Mathematics
Fall 2019
Lab Assignment #2

You have THREE LAB SESSIONS to complete this lab, the week of Oct. 21, the week of Oct. 28, and the week of Nov. 4.

YOUR SOLUTION MUST BE DEMONSTRATED TO YOUR TA DURING LAB NO LATER THAN THE LAB SESSIONS IN THE WEEK OF NOV. 4:

```
CSE-015-02L LAB W 7:30-10:20am: IN LAB NOV. 6
CSE-015-03L LAB W 10:30-1:20pm: IN LAB NOV. 6
CSE-015-04L LAB F 10:30-1:20pm: IN LAB NOV. 8
CSE-015-05L LAB M 1:30-4:20pm:  IN LAB NOV. 4
CSE-015-06L LAB W 1:30-4:20pm:  IN LAB NOV. 6
CSE-015-07L LAB F 1:30-4:20pm:  IN LAB NOV. 8
CSE-015-08L LAB W 7:30-10:20pm: IN LAB NOV. 6
CSE-015-09L LAB F 7:30-10:20am: IN LAB NOV. 8
CSE-015-10L LAB M 7:30-10:20pm: IN LAB NOV. 4
CSE-015-11L LAB R 7:30-10:20pm: IN LAB NOV. 5
```

If you demonstrate your solution to your TA in the first or second lab session, you don't need to attend the subsequent lab sessions for this assignment.

You must demonstrate your solution to YOUR TA IN YOUR LAB SESSION. You cannot demonstrate to another TA or in a lab session other than your own.

Introduction

This lab gives you further practice with the Python language on problems related to truth tables in propositional logic. It will also give you more practice manipulating Python lists.

Even for the simplest programs we write, we make use of code written by others. For this lab you are provided with code that generates truth tables.

To start, download the file `logic.py` from CatCourses and place it in your working directory (the directory where your .py files for this assignment will be located). Among other things, the file defines a `TruthTable` object. You can create and use a `TruthTable` object by importing it from `logic.py`, using the following command:

```
from logic import TruthTable
```

You can now generate truth tables for any proposition. All you need to specify is the list of variables that appear in your propositions, and the propositions themselves.

As an example, create and run the following Python script which creates and displays the truth table for the proposition $p \vee q$.

```
from logic import TruthTable
myTable = TruthTable(['p', 'q'], ['p or q'])
myTable.display()
```

This should produce the following text:

```
p       q       p or q
-----------------------
0       0       0
0       1       1
1       0       1
1       1       1
```

The `TruthTable` object can also be called with multiple propositions.

```
myTable = TruthTable(['p', 'q'], ['p or q', 'p and q'])
```

The command above generates one truth table with both propositions side by side:

```
p       q       p or q   p and q
--------------------------------
0       0       0        0
0       1       1        0
1       0       1        0
1       1       1        1
```

When using `TruthTable`, we need to specify all propositions and propositional variables as Python strings, as illustrated in the examples here. The `TruthTable` object supports the following logical connectives:

`or` ($\vee$), `and` ($\wedge$), `-` ($\neg$), `->` ($\rightarrow$), `<->` ($\leftrightarrow$).

Problem 1: Basic Truth Tables (20 pts.)

Write a Python program that creates and prints out the truth tables for all logical connectives studied in class. There should be a separate truth table for the following:

$\neg a$

$a \wedge b$

$a \vee b$

$a \rightarrow b$

$a \leftrightarrow b$

Problem 2: Equivalence in Propositional Logic (20 pts.)

As discussed in class, one way to prove whether two propositions are equivalent, is to generate truth tables for both propositions and check to see if all the rows match, in which case the propositions are equivalent, otherwise they are not.

In this exercise you are required to write a Python program that asks the user to enter two propositions, where both propositions involve the variables `p` and `q`. Your program should then determine whether or not the two propositions are equivalent, and report the result.

Here is an example of how a user might interact with your program:

```
Enter proposition 1: p and q
Enter proposition 2: q and p

The propositions are equivalent
```

As mentioned above, the file `logic.py` defines an object called `TruthTable`. You create instances of this object by calling the object name with initialization parameters:

```
myTable = TruthTable(['p', 'q'], ['p or q'])
```

`TruthTable` expects two lists as input: one listing the propositional variables and another listing the propositions.

`TruthTable` computes the truth table and stores it in the attribute `table` which is also a list.

You can access this list using `myTable.table`. For example, the following prints the list:

```
myTable = TruthTable(['p', 'q'], ['p or q'])
print(myTable.table)
```

The above outputs the `table` of the proposition $p \wedge q$:

```
[[[0, 0], [0]], [[0, 1], [0]], [[1, 0], [0]], [[1, 1], [1]]]
```

As you can see, it is one big list, made up of exactly 4 smaller lists. Each one of the smaller lists represents a row of the truth table. Each row in turn is made up of exactly 2 lists. The first one is a list of truth values of the propositional variables, and the second one is the overall truth value of the proposition(s), given the truth values in the last list.

If we take the above `table` and consider the first row, that is the list `[[0, 0], [0]]`. Bearing in mind that this is for the formula $p \wedge q$, the way to interpret this list is that when the variables p and q are `[0, 0]`, respectively, then the proposition $p \wedge q$ has a truth value of `[0]`, or simply `0`. Remember, this is represented as a list with a single element in it because the truth table may have more than one proposition being evaluated.

Again, `table` is a Python list. The following will print the entries that correspond to just the first row in `table`

```
myTableList = myTable.table
print(myTableList[0])
```

This will produce

```
[[0, 0], [0]]
```

The following will print the entries that correspond to just the last row in `table`

```
myTableList = myTable.table
print(myTableList[-1])
```

This will produce

```
[[1, 1], [1]]
```

To iterate through and print every row

```
for row in myTableList:
    print(row)
```

This will produce

```
[[1, 1], [1]]
[[0, 0], [0]]
[[0, 1], [1]]
[[1, 0], [1]]
[[1, 1], [1]]
```

Remember that `table` is a list of lists. To print just the portion of the table that corresponds to the propositions being evaluated, you would use

```
for row in myTableList:
    propList = row
    print( propList[1])
```

This will produce

```
[0]
[1]
[1]
[1]
```

Now back to the problem of checking whether two propositions are equivalent. Remember that we can use `TruthTable` to create a truth table with multiple propositions. For example, to generate a single truth table for both $p \wedge q$ and $q \wedge p$ we would use:

```
myTable = TruthTable(['p', 'q'], ['p and q', 'q and p'])
```

This would create the following `table`:

```
[[[0, 0], [0, 0]], [[0, 1], [0, 0]], [[1, 0], [0, 0]], [[1, 1], [1, 1]]]
```

It is still made up of 4 lists, each of which represents a row of the truth table. Each row is still made up of two lists, as before. The first of these lists is still a specific combination of truth values, and the second list is the overall truth value of each proposition. Since we generated a truth table for 2 propositions, the list has 2 results in it. This time, taking the second row, which is `[[0, 1], [0, 0]]`, we can see that when $p = 0$ and $q = 1$ then $p \wedge q$ evaluates to false, and $q \wedge p$ also evaluates to false, hence both `0` values in the list. Examining the whole table, we can see that the two propositions are indeed equivalent because, for every row, the result part always contains the same two values, meaning that for any possible combination of truth values, both propositions evaluate to the same result.

You solution to problem 2 should:

- Prompt the user for two propositions involving the two propositional variables p and q:

  ```
  Enter proposition 1:
  Enter proposition 2:
  ```

- Create a truth table that includes these two propositions.
- Check whether the two propositions are equivalent or not by accessing the `table` attribute of the truth table you created.
- Output `The propositions are equivalent` or `The propositions are not equivalent`.

Some notes:

- You cannot "hardcode" the two propositions. They must be input by the user.
- The order of p and q can differ between the two propositions. Make sure your solution concludes that $p \wedge q$ and $q \wedge p$ are equivalent.
- You might want to first print `table` to check that it is correct before parsing it.