Part 1: Python Programming Problem

Background:

We will be studying two distinct "levels" of logic: *propositional* logic and *predicate* logic. In propositional logic, variables stand for complete propositions, but we are usually only interested in the boolean *truth values* of those propositions: true or false. Propositional logic evaluates the truth or falsity of *propositional formulas* made up of propositional variables combined using the *logical operators* AND, OR, and NOT, e.g., (**A** AND (NOT **B**))

Operators: In Java terms, this would correspond to determining the values of Java expressions made up of only boolean variables combined using the operators **`&&`** for AND, **`||`** for OR, and **`!`** for NOT. You have learned about these, so the basic idea should be familiar. These logical operators in Python are spelled out -- **`and`**, **`or`**, and **`not`** -- but they work the same. For example, the Python expression **`(A and B)`** will evaluate to **`True`** if two variables named **`A`** and **`B`** both have **`True`** (or at least "truthy") values, and will evaluate to **`False`** otherwise.

Environments: A particular assignment of boolean values to all the propositional variables in a formula is called an *environment*. Using Python dictionary notation for environments, we would say that **`(A and B)`** is **`True`** in the environment **`{'A': True, 'B': True}`** but is **`False`** in the environment **`{'A': True, 'B': False}`**. Python provides a function **`eval`** that can evaluate the truth value of logical formulas in specific environments:

```
formula = '(A and B)'
environ = {'A': True, 'B': False}
print(eval(formula, environ)) # prints "False"
```

Note that the formula and the names of the logical variables can be given as Python strings. Easy peasy.

Satisfiability: Of particular interest is the *satisfiability* of propositional formulas. An *unsatisfiable formula* is false in all environments, whereas a *satisfiable formula* is true in *at least one* environment. Furthermore, a satisfiable formula is a *tautology* if it is true in *every* environment. There has been a lot of Computer Science research to find quicker ways to tell whether huge logical formulas with tens of thousands of variables are satisfiable or not. This is of important practical use in several fields, including chip design, AI, and operations research. Determining satisfiability is considered a very difficult problem ("NP-hard") for such large formulas.

For small formulas with just a dozen variables, however, we can determine satisfiability by "brute force": enumerate all possible environments and test each one. If there are only two variables, for example, there are only 4 possible environments. Even for, say, 12 variables, that's only a few thousand environments – tiny in modern computer terms.

Program Requirements:

1. Your program's file shall be named **`satisfy.py`**.
2. This program shall take the name of a text file as its first and only command line argument.
3. The contents of this text file may be *any logical formula* in valid Python notation, where all the logical variable names will be single capital letters.
4. Your program shall read the logical formula in that text file.
5. Your program shall extract a *set* of all *distinct* logical variable names from the logical formula (i.e., the same name might appear more than once in the formula)
6. Your program shall enumerate all possible environments for that set of variables as dictionaries
7. Your program shall evaluate the formula for each environment
8. You program shall report (i.e., print on one complete line) how many environments satisfied the formula, and how many did not, e.g.: "Satisfied: 3; Not Satisfied: 5"
9. After thorough testing with several logical formulas, upload your program file to the provided Canvas drop box. Your professor will test it with several more logical formulas.

The only challenging requirement is #6. You need to consider how to enumerate all possible assignments of Boolean values to N variables. Here are some hints. Consider all the possible truth value assignments for 1, 2 and 3 variables:

N=1
A
F
T

N=2	
B	**A**
F	F
F	T
T	F
T	T

N=3		
C	**B**	**A**
F	F	F
F	F	T
F	T	F
F	T	T
T	F	F
T	F	T
T	T	F
T	T	T

Binary Counter
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1

For one variable, there are only 2 environments, for two there are 4, and for three there are 8. The number of environments doubles for each additional variable. In the scheme above, the first variable **A** always alternates between `F` and `T`; a second variable B alternates between two consecutive `F's` and two consecutive `T's`; a third variable C would alternate between 4 consecutive `F's` and 4 consecutive `T's`. Each time the value in one column flips from `T` back to `F`, the value in the column immediately to the left changes. That suggests a recursive algorithm…

Is there an even cleverer way to generate such a pattern of Boolean values? Consider the last table above showing the least significant 3 binary digits of the first 8 natural numbers (zero through seven). Compare the digits in those numbers with the Boolean values in the "N=3" table. See a pattern? Hmmm…. Maybe you could use a simple counter to generate the Boolean patterns, especially since Python doesn't require `True` and `False` specifically, only *truthy* and *falsey* values, like …. 0 and 1. All you need is a way to extract a particular binary digit from a number. Ideas??

Part 2: A Few Math Proofs:

If you are not comfortable with typing math notations (e.g., Word's equation editor), you may write these out by hand and turn in a **clean** PDF scan with good contrast and legibility. A poorly lit phone scan (black on brown, etc.) is not acceptable; there are plenty of well-lit places on campus. There are also several programs that will square up a phone scan and tweak its appearance – use one if you submit a scan!

1. Prove by contradiction that $\sqrt[9]{12}$ is irrational.

2. Prove by cases that `|r + s| <= |r| + |s|` for any real numbers `r` and `s`.

3. It's a fact that the Arithmetic Mean is at least as large as the Geometric Mean, namely,

$$\frac{a+b}{2} \geq \sqrt{ab}$$

for all nonnegative real numbers a and b. But there's something objectionable about the following proof of this fact. What's the objection, and how would you fix it?

$$\begin{aligned} a + b &\stackrel{?}{\geq} 2\sqrt{ab}, && \text{so} \\ a^2 + 2ab + b^2 &\stackrel{?}{\geq} 4ab, && \text{so} \\ a^2 - 2ab + b^2 &\stackrel{?}{\geq} 0, && \text{so} \\ (a-b)^2 &\geq 0 && \text{which we know is true.} \end{aligned}$$

The last statement is true because $a - b$ is a real number, and the square of a real number is never negative. This proves the claim. ■

(I actually see 2 problems here, one of computation and one of the way the inference was presented. Can you see both?)