This assignment is **due in Week 13** and should be submitted to Gradescope. All work must be **done individually** without consulting anyone else's solutions in accordance with the University's "Academic Dishonesty and Plagiarism" policies.

Go to the last page of this document and read the **Submission Instructions**. For clarifications and updates, monitor "Assignment FAQ".

Problem 1. (10 marks) Provide truth tables for the following formulas:

1. $\neg(p \to q) \wedge p$
2. $((p \wedge q) \vee (\neg p \wedge q)) \leftrightarrow ((p \to r) \wedge (\neg p \to r))$

Problem 2. (10 marks) For each of the following formulas, provide an equivalent one in conjunctive normal form:

1. $(a \wedge b) \vee (a \wedge c) \vee (a \wedge d) \vee (b \wedge c) \vee (b \wedge d) \vee (c \wedge d)$
2. $(a \wedge b \wedge \neg c) \vee (a \wedge \neg b \wedge c) \vee (\neg a \wedge b \wedge c)$

Problem 3. (15 marks) In the following Python code snippets, all variables are Boolean, and the functions return Boolean values. These are called "Boolean functions". It is known that every Boolean function is equivalent to a formula of propositional logic. This might be called a "fundamental theorem of propositional logic".

For each of the functions, write an equivalent propositional logic formula over the basic syntax.

For instance, for the code:

```
def sample_function(a,v):
    if a and v:
        return False
    else:
        return True
```

a correct answer would be $\neg(a \wedge v)$ or, any equivalent formula, e.g., $(\neg a \vee \neg v)$.

Here are the functions.

```
def function1(x,y,z):
    if x:
        if y:
            if z:
                return True
            else:
                return False
        else:
            return True
    else:
        return False
```

```python
def function2(w,x,y,z): # you should check how Python adds Booleans
    return x + y - z == 1 + w
```

```python
def function3(e, r, w, h, v, b):
    if e:
        if r or (w and not h):
            if not v:
                return True
            else:
                return False
        else:
            if not r and h:
                return False
            else:
                return True
    else:
        if b and r:
            return True
        else:
            return False
```

Problem 4. (15 marks) Prove the following equivalences using the Table of Equivalences. For full marks, your proof should use as few lines as possible.

1. $(\neg p \to q) \equiv (\neg q \to p)$
2. $((p \to q) \to (q \to p)) \equiv (q \to p)$
3. $(((\neg p \vee q) \wedge (r \vee p)) \wedge (r \vee q)) \equiv ((\neg p \vee q) \wedge (r \vee p))$

Problem 5. (15 marks) You are the royal architect, and the King has commissioned you to design a mosaic floor for his court. The floor is an $n \times n$ square, and you have been provided with exactly k rectangular tiles, each with specific dimensions. Can you arrange the tiles to create an elegant design that gains the King's approval? To start, you simply want to check if every tile can be used, with no overlaps (you do not require that the entire floor is tiled).

More specifically: Given a square of side length n, and a list of k rectangles and their widths and heights $W_1, H_1, W_2, H_2, \cdots, W_k, H_k$ (this data is called an *instance*), decide whether we can put all the rectangles in the square without overlap and no rotating, and, if so, where in the $n \times n$ square we can place the rectangles (this is called a *solution*). Note that it is not necessary to cover every cell in the square. We call this problem the Mosaic problem.

For example, given the following *instance*: $n = 3$, and the three rectangles $W_1 = 2, H_1 = 2, W_2 = 2, H_2 = 1$ and $W_3 = 1, H_3 = 2$, one possible *solution* is to lay out the rectangles as in the illustration below:

1	1	3
1	1	3
	2	2

Cells that have a number i denotes that the ith rectangle (whose widths and heights are W_i and H_i) cover that cell.

Your task is to write a Python program that will convert an instance of the Mosaic problem into a Boolean formula such that the formula is satisfiable iff the instance has a solution, and moreover, that solutions to the Mosaic Problem correspond to solutions to the Boolean satisfiability problem. The size of the formula should be polynomial in n and k.

Write:

1. A function that converts an instance of the Mosaic problem to a formula in CNF. This function should have the property that the solutions of the given instance correspond to the satisfying assignments of the formula and vice versa. In particular, the instance has no solution exactly when the formula is unsatisfiable.

   The input will be provided as an *Instance* object which contains a number n and a list r of k-many pairs (W_i, H_i), and the output is expected to be a *Formula* object over string atoms in strict CNF form.

   This function should run reasonably quickly and should not attempt to directly solve the Mosaic problem.

2. A function that, given a satisfying assignment to the formula produced in part 1, produces a solution to the instance.

   The input will be a *Formula* object produced by your function in part 1 and an *Assignment* object that satisfies that formula, and the output will be a *Solution* object. This *Solution* object will be a list of k tuples, with the tuple at index i indicating the location of the top-left cell covered by the $i+1$st rectangle (i.e., the one with dimensions W_{i+1} and H_{i+1}).

   Cell numbering works as follows: each cell is identified by a tuple, the first element being the x-coordinate (increasing left-to-right) and the second element being the y-coordinate (increasing-top-to-bottom). The top-left cell in the $n \times n$ square is be numbered $(0, 0)$, and the bottom-right square is numbered $(n-1, n-1)$. For instance, in the example, the cell numbering is like this:

0,0	1,0	2,0
0,1	1,1	2,1
0,2	1,2	2,2

and the solution above is represented by the following Solution: $[(0,0),(1,2),(2,0)]$.

3. A function that, given a solution to an instance of the Mosaic problem, produces a satisfying assignment to the formula produced in part 1.

   The input will be a *Formula* object produced by your function in part 1 and a *Solution* object, and the output will be an *Assignment* object.

You may assume that, for $1 \leq i \leq k$, $1 \leq W_i \leq n$, and $1 \leq H_i \leq n$. The input is guaranteed to be in the correct format and does not need to be verified.

Note that parts 2 and 3 hinge on part 1.

Problem 6. (5 marks + 10 bonus marks)

You have just landed on the island of Logos, where everyone who lives there either always tells the truth or always lies. You hence use propositional logic in your everyday life to get information from them. You can ask people questions phrased in the form of propositional logic formulas, and they will tell the truth or lie about whether the formula is true. Everyone who lives on Logos can tell whether another inhabitant always tells the truth or always lies, just by looking at them.

The atoms involved in the formula are t_i, stating that the i-th person always tells the truth, f_i, stating that the i-th person always lies, and r, an atom that you want to find out the value of.

However, you cannot ask a person about their own status. That is, if asking person i, your formula cannot include the atoms t_i or f_i.

Scenario 0 (example, no marks):

You come to a road with two guards, one who always tells the truth and one who always lies. You want to know whether the road leads to the capital. The classic solution to this riddle is to ask a guard "if I asked the other guard if this is the road to the capital, what would they answer?", and then negate the answer.

We can formalise this by asking guard 1 for the truth value of the formula

$$(t_2 \wedge r) \vee (f_2 \wedge \neg r)$$

If she answers true, r is false, and if she answers false, r is true. This strategy works no matter whether guard 1 always tells the truth or always lies. This strategy solves this scenario with just 1 question.

Scenario 1:

You come to a road with three guards, one who always tells the truth, one who always lies, and one who either always tells the truth or always lies but you're not sure which.

1. (5 marks) Provide a strategy for the above scenario, in the form of a Python program. For full marks, your strategy should always give the correct answer and the worst case number of questions should be as low as possible.

Scenario 2:

Recently, a former inhabitant has returned to the island. However, on their travels, they have picked up a nasty habit: they are not guaranteed to always tell the truth or always lie, but instead answer questions at random. That is, when given a question, they will ignore the formula and just mentally flip a coin to decide whether to answer yes or no. The locals know this about the returner. (Note that for the returner, t_i and f_i will both be false.)

You come to a road with three guards: one guard who always tells the truth, one who always lies, and the returner described above.

2. (5 bonus marks) Provide a strategy for the second scenario, in the form of a Python program. For full marks, your strategy should always give the correct answer and the worst case number of questions should be as low as possible.

The final part of this problem is based on Scenario 1.

3. (5 bonus marks) Provide a lower bound on the worst case number of questions needed in scenario 1. That is, select a positive integer k and show that any strategy that always uses fewer than k questions will sometimes produce the wrong answer. For full marks, your bound k should be as high as possible.

Notes: You may assume that any proposed strategy for this problem is deterministic. Any such strategy can be characterised by the questions it asks depending on what it hears back from the guards. For example, if the strategy asks at most 2 questions, then it can be characterised by questions $(F, g), (F_t, g_t), (F_f, g_f)$ and decisions $D_{tt}, D_{tf}, D_{ft}, D_{ff}$. Given such a strategy, you must then show it does not work by showing there exists at least one case where it gives the wrong answer.

For this bonus problem, marks will only be awarded for substantial progress. Attempts that do not make any substantial progress will receive zero marks.

Submission Instructions

Problem 1

Each file must contain a truth table for a propositional formula.

The following is an example structure of a file:

```
p q o
0 1 0
0 0 0
1 0 0
1 1 1
```

The first line lists the variables (in this case p, q) followed by a lower-case letter 'o', not zero (standing for "output"). Each subsequent line lists a row of the truth-table.

Problems 2 and 3

Each file must contain a propositional formula using the following symbols: Use `&` (ampersand) for conjunction, `|` (stroke) for disjunction, $\sim$ (tilde) for negation, parentheses `()`, and atoms $p, q, r, \cdots$.

Problem 4

You will use the Logic Tutor Webapp (LT) for this question.

Instructions on how to use LT are here: https://edstem.org/au/courses/17553/discussion/2313067

For each question, you will submit a link to your solution on LT, as well as a PDF of your solution.

Problem 5, Problem 6.1, 6.2 format:

Problems 5 and 6 (parts 1 and 2) ask you to write a Python program. A scaffold is provided on Ed to form the basis of your submission. In the file a4p5.py or a4p6.py, implement the functions shown, as well as any additional code needed for your solution. The other files (objects5.py and formulas6.py) contain definitions of object classes, and should not be modified, but should be read to understand how to interact with the classes.

Input will be provided in the form of arguments to the functions given. Do not modify the name or arguments of these functions. Comments will describe the types of objects these functions are expected to take in. Output is expected in the return value of these functions. Comments will describe the types of objects these functions are expected to return.

Some sample testing code is provided, which will be executed when the file is run directly (but not during grading). The sample test corresponds to a public test available on Gradescope. You may run this test locally, and may write additional tests in this section as desired.

Do not attempt to deceive the marking system or otherwise interact with the autograding environment beyond the input/output of the functions provided. Any attempt to do so, if detected, will result in an immediate zero for the question and possibly further consequences.

Problem 6.3 format:

Problem 6.3 is handgraded. You will submit a single **typed pdf (no pdf containing text as images, no handwriting)**. Start by typing your student ID at the top of the first page of each pdf. Do **not** type your name. Do not include a cover page. Submit only your answers to the questions. Do **not** copy the questions. Your pdf must be readable by Turnitin.

Equivalences for propositional logic

(Idempotent Laws)	$F \equiv (F \wedge F)$
	$F \equiv (F \vee F)$
(Commutative Laws)	$(F \wedge G) \equiv (G \wedge F)$
	$(F \vee G) \equiv (G \vee F)$
(Associative Laws)	$(F \wedge (G \wedge H)) \equiv ((F \wedge G) \wedge H)$
	$(F \vee (G \vee H)) \equiv ((F \vee G) \vee H)$
(Absorption Laws)	$(F \wedge (F \vee G)) \equiv F$
	$(F \vee (F \wedge G)) \equiv F$
(Distributive Laws)	$(F \wedge (G \vee H)) \equiv ((F \wedge G) \vee (F \wedge H))$
	$(F \vee (G \wedge H)) \equiv ((F \vee G) \wedge (F \vee H))$
(de Morgan's Laws)	$\neg(F \wedge G) \equiv (\neg F \vee \neg G)$
	$\neg(F \vee G) \equiv (\neg F \wedge \neg G)$
(Double Negation Law)	$\neg\neg F \equiv F$
(Validity Law)	$(F \vee \top) \equiv \top$
	$(F \wedge \top) \equiv F$
(Unsatisfiability Law)	$(F \vee \bot) \equiv F$
	$(F \wedge \bot) \equiv \bot$
(Constant Laws)	$\top \equiv (F \vee \neg F)$
	$\bot \equiv (F \wedge \neg F)$
(Negating constants Laws)	$\neg\top \equiv \bot$
	$\neg\bot \equiv \top$
(Conditional Law)	$(F \to G) \equiv (\neg F \vee G)$
(Bi-conditional Law)	$(F \leftrightarrow G) \equiv ((F \to G) \wedge (G \to F))$