Important: Because of name clashes with various Python modules, the file for your Python code is now called hwcode . py rather than code . py.

Several of the problems in this assignment are tied to each other. You will still lose credit points for wrong answers, even if these are caused by previous wrong answers, so be careful.

Part 1: Truth Tables

\textbf{LATEX Note:} The following truth table in the PDF output:

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$((\neg A) \rightarrow (\neg B))$</th>
<th>$(((\neg A) \rightarrow B) \rightarrow A)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

can be obtained with the following \LaTeX input:

\begin{center}
\begin{tabular}{*{2}{c} | *{12}{c}}
$A$ & $B$ & $((\lognot A) \rightarrow (\lognot B))$ & $(((\lognot A) \rightarrow B) \rightarrow A)$ \\
| T & T & F & T & F & T & F & T & T & \\
| T & F & F & T & T & F & T & T & \\
| F & T & T & F & F & T & T & T & \\
| F & F & F & T & T & T & T & F & T & \\
\end{tabular}
\end{center}

\LaTeX commands for the truth values \texttt{\logT} and \texttt{\logF}, as well as for the logical operators \texttt{\lognot}, \texttt{\logthen} and several others are defined for your convenience in macros . sty. \textbf{Of course, getting the ampersands right is crucial for things to line up properly.}

\textbf{Problem 1.1 (tag: truth)}

Write a truth table for the following propositional logic formula using the format exemplified above.

$((A \rightarrow B) \rightarrow ((B \rightarrow (C \rightarrow D)) \rightarrow ((A \land C) \rightarrow D))$

This formula is fully parenthesized to avoid ambiguities. Do not skip any of the parentheses in your table. Remember to number the columns (see the bottom row of the sample table above) and to place dividing lines just like in the sample table: one vertical line between the variable list and the formula, one horizontal line just below the header, and one horizontal line just above the column numbers. Do not add any other lines or decorations.

\textbf{Problem 1.2 (tag: imply)}

Is the formula given in the previous problem an implication? Why or why not?
Part 2: Valid Inference

Problem 2.1 (tag: cases)

When studying valid inference, we sometimes reasoned by cases (see the Feb 23 class notes). The structure of this type of argument is captured by the following inference rule:

\[
\begin{align*}
A & \to C \\
B & \to C \\
A \lor B & \\
\hline
C
\end{align*}
\]

In words, if \( C \) follows from both \( A \) and \( B \) and either \( A \) or \( B \) is true, then \( C \) must be true.

Write a truth table that shows that this inference rule is valid. Explain briefly and clearly why your truth table lets you draw this conclusion. Use the same truth-table format as in previous problems.
Part 3: The Multinomial Coefficients

Let \( n, k \) be natural numbers with \( k \leq n \). The binomial coefficient

\[
\binom{n}{k} = \frac{n!}{k!(n-k)!}
\]

counts the number of distinct \( k \)-subsets of an \( n \)-set. It can be rewritten as follows if \( k_1 = k \):

\[
\binom{n}{k_1, k_2} = \frac{n!}{k_1!k_2!}
\]

with the constraint that \( k_1 + k_2 = n \).

Thus, \( k_1 \) is the same as \( k \), and \( k_2 \) is the same as \( n - k \).

The second expression is more symmetric, since the roles of \( k_1 \) and \( k_2 \) are perfectly interchangeable. It makes the following equality obvious:

\[
\binom{n}{k} = \binom{n}{n-k}
\]

and is then more aptly described as the number of ways to partition a set of \( n \) elements into two subsets of \( k_1 \) and \( k_2 \) elements. The following result should not come as a surprise:

**Multinomial-Coefficient Theorem**

Let \( s > 0 \) and \( k_1 \geq 0, \ldots, k_s \geq 0 \) be natural numbers, and define

\[
n = k_1 + \ldots + k_s .
\]

The number of ways to partition a set of \( n \) elements into \( s \) subsets of \( k_1, \ldots, k_s \) elements each is

\[
\binom{n}{k_1, \ldots, k_s} = \frac{n!}{k_1! \cdots k_s!} .
\]

**Problem 3.1 (tag: direct)**

Prove the multinomial-coefficient theorem for an \( n \)-set \( S \) by counting the number of distinct ways to select the elements for each of the \( s \) subsets (plus some algebra). You may assume the result for \( s = 2 \) to be known, since you have studied the binomial coefficients. Show your reasoning and calculations.

\LaTeX Note: The expression

\[
\binom{n}{k_1, \ldots, k_s}
\]

can be obtained with the following \LaTeX command:

\[
\text{\textbackslash binom\{n\}\{k_1, \ldots, k_s\}}
\]

**Problem 3.2 (tag: bookkeeper)**

Use the multinomial-coefficient theorem to compute the number \( n \) of unique permutations of the word BOOKKEEPER. Show your reasoning and calculations.

**Problem 3.3 (tag: multinomial)**

Write a Python function with header

\[
\text{def multinomial(kseq):}
\]

that takes an iterable (a tuple or a list, say) \( \text{kseq} \) that represents the sequence \( (k_1, \ldots, k_s) \) and returns the value of the multinomial

\[
\binom{n}{k_1, \ldots, k_s} \quad \text{where} \quad n = k_1 + \ldots + k_s .
\]

You may assume that all the elements of the sequence are nonnegative integers, and that at least one of them is nonzero. (Thus, there is no need to include argument checks.)

To compute combinatorial quantities, you may use the functions in the \texttt{sympy} module as explained in homework 5, or develop your own functions.

**What to turn in:** Display your code and the results of the tests from \texttt{hwcode.py}.
Part 4: Quantification

Given a predicate $P$ with $n$ variables $x_1, \ldots, x_n$, a formula is a version of $P$ where all variables are quantified. For instance, for $n = 3$, the following is a formula:

$$\forall x_2 \exists x_3 \exists x_1 P(x_1, x_2, x_3).$$

Note that all variables are written out in order as arguments to $P$, but can appear in any order in the quantifiers. Every variable must be quantified exactly once. The quantified variables form the prefix of the formula, and the predicate itself is called the matrix:

$$\forall x_2 \exists x_3 \exists x_1 P(x_1, x_2, x_3).$$

Each variable $x_i$ belongs to a predefined set $U_i$ called the universe for that variable, so that the formula above is really a shorthand for the following:

$$\forall x_2 \in U_2 \exists x_3 \in U_3 \exists x_1 \in U_1 P(x_1, x_2, x_3).$$

The universal quantifier $\forall$ is equivalent to a conjunction over the universe of the quantified variable. For instance,

$$\forall x_1 P(x_1) \iff P(u^{(1)}_1) \cap P(u^{(1)}_2) \cap \ldots$$

where the terms $u^{(1)}_1, u^{(1)}_2, \ldots$ are all the elements of $U_1$. Similarly, the existential quantifier $\exists$ is equivalent to a disjunction:

$$\exists x_1 P(x_1) \iff P(u^{(1)}_1) \cup P(u^{(1)}_2) \cup \ldots$$

Both conjunction and disjunction have an infinite number of terms when the universe is an infinite set.

The order in which variables are quantified in the prefix of a formula matters. For instance, any two of the following formulas are generally not equivalent to each other:

$$\forall x_1 \forall x_2 P(x_1, x_2)$$
$$\forall x_1 \exists x_2 P(x_1, x_2)$$
$$\forall x_2 \exists x_1 P(x_1, x_2)$$
$$\exists x_1 \forall x_2 P(x_1, x_2)$$
$$\exists x_2 \forall x_1 P(x_1, x_2)$$
$$\exists x_1 \exists x_2 P(x_1, x_2)$$

However, formulas that differ by permuting quantifiers of the same type are equivalent if the two quantifiers being swapped are not separated by a quantifier of the other type. For instance,

$$\forall x_1 \forall x_2 P(x_1, x_2) \iff \forall x_2 \forall x_1 P(x_1, x_2)$$
$$\exists x_1 \exists x_2 P(x_1, x_2) \iff \exists x_2 \exists x_1 P(x_1, x_2).$$

These observations are straightforward to prove. For instance, to see that

$$\forall x_1 \exists x_2 P(x_1, x_2) \text{ and } \exists x_2 \forall x_1 P(x_1, x_2)$$

are not equivalent to each other, you can take a simple example with two-element universes:

$$U_1 = \{ a_1, a_2 \} \text{ and } U_2 = \{ b_1, b_2 \}$$

and then convert the formulas to conjunctions or disjunctions:

$$\forall x_1 \exists x_2 P(x_1, x_2) \iff [P(a_1, b_1) \cup P(a_1, b_2)] \cap [P(a_2, b_1) \cup P(a_2, b_2)]$$
$$\exists x_2 \forall x_1 P(x_1, x_2) \iff [P(a_1, b_1) \cap P(a_2, b_1)] \cup [P(a_1, b_2) \cap P(a_2, b_2)].$$

Note that the first quantifier ($\forall$ in the first case above) corresponds to the outermost connective ($\cap$), and the corresponding variable ($x_1$) varies across brackets but not within a bracket. The truth values of the two expressions on the right are obviously different in general.
Problem 4.1 (tag: switching)

Let 

\[ X = \{2, 5\} \quad \text{and} \quad Y = \{3, 4\} \]

be the universes for two variables \( x \in X \) and \( y \in Y \), and let \( P(x, y) \) be the predicate that is true if and only if \( x < y \).

For each of the six possible formulas that quantify \( x \) and \( y \) for matrix \( P(x, y) \), state if the formula is true or false.

Remarks:

The expansion of quantifiers explained in the introduction above is straightforward, but be careful to use it correctly. You will get full credit for every correct truth value and no credit for any incorrect truth value. No partial credit will be given.

There are various ways to get this right. I did not trust my own patience, so I wrote Python code to generate the truth values automatically. If you do that, check your code with a few formulas for which you know the answer.

Do not show your expansions or code. Just replace the question marks in assignment.tex with “true” or “false” (without the quotation marks!) as appropriate.
Part 5: Counting Quantifications

In this part, we count the number \( q(n) \) of possible distinct quantifications of a formula with \( n \) variables and \( n \) quantifiers,

\[
Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \ P(x_1, x_2, \ldots, x_n)
\]

where each \( Q_i \) is either \( \forall \) or \( \exists \). You may recall that this question arose during a discussion in class.

Recall the following facts from the previous introduction:

1. The prefix has \( n \) quantifiers, and each variable \( x_i \) must appear exactly once in the prefix.
2. Changing the quantifier of any variable \( x_i \) from a \( \forall \) to a \( \exists \) or vice versa yields a different quantification.
3. Switching the order of two quantified variables \( Q_i x_i \) and \( Q_j x_j \) in the prefix of the formula yields a different quantification, except when \( Q_i \) is the same as \( Q_j \) and as all the quantifiers between \( Q_i \) and \( Q_j \).

To reason about prefixes, we introduce the notion of a run of quantifiers, that is, a sequence of consecutive quantifiers of the same type and of maximal length. For instance, reading the prefix

\[
\forall x_8 \forall x_5 \forall x_7 \exists x_4 \exists x_2 \forall x_6 \forall x_1 \exists x_3
\]

from left to right, we find a run of 3 \( \forall \), a run of 2 \( \exists \), a run of 2 \( \forall \), and a run of 1 \( \exists \). We say that the type of this prefix is the sequence \((3, 2, 2, 1)\).

To get a sense for how many quantifications there are, we first derive a lower and an upper bound on \( q(n) \) in the next problem. We then come up with an exact expression for \( q(n) \) and write Python code to compute its value as a function of \( n \).

Problem 5.1 (tag: lower)
Explain briefly and clearly why \( q(n) \geq 2^n \).
[Hint: Keep all variables in the prefix in a fixed order, say \( x_1, x_2, \ldots, x_n \).]

Problem 5.2 (tag: upper)
Explain briefly and clearly why \( q(n) \leq 2^n n! \).
[Hint: What sequence(s) of \( n \) quantifiers yields the maximum number of distinct quantifications by switching the order of its variables in the prefix?]

Problem 5.3 (tag: qtype)
The two previous problems show that the number of distinct quantifications \( q(n) \) of a matrix with \( n \) variables satisfies the following bounds:

\[
2^n \leq q(n) \leq 2^n n! .
\]

In this and the next few problems we devise a formula for computing \( q(n) \) exactly.

Let \( k \) be an integer between 0 and \( 2^n - 1 \) inclusive. We associate an \( n \)-tuple of quantifiers to \( k \) as follows:

- Pad the binary representation of \( k \) with zeros on the left so that it is exactly \( n \) bits long. For instance, if \( n = 7 \) and \( k = 23 \), then \( k \) in binary is 10111, and we pad it to 0010111.
- Associate a universal quantifier \( \forall \) to zeros and an existential quantifier \( \exists \) to ones. In the running example, the sequence of quantifiers is \((\forall, \forall, \exists, \forall, \exists, \exists, \exists)\).

The Python function \( qruns(k, n) \) provided in \textit{hwcode.py} computes the resulting sequence of quantifiers, broken up into tuples. It represents \( \forall \) by the character 'A' and \( \exists \) by the character 'E'. For our running example, \( qruns \) returns

\[
( (\forall', \ 'A'), \ ('E'), \ ) \ (('A'), \ ) \ (('E', \ 'E', \ 'E'))
\]
Note that this is not a single tuple with 7 characters, but a tuple with one tuple of characters per run. So the length of the tuple is the number of runs in the sequence of quantifiers.

We then let $t(k, n)$ be the type of the sequence of $n$-variable quantifiers associated to $k$. In the example above

$$t(23, 7) = (2, 1, 1, 3).$$

Write a Python function with header¹

```python
def qtype(runs):
```

that takes a sequence `runs` output by `qruns` and returns the type of the sequence of quantifiers that `runs` represents.

Your function should be one line long (including the header) and return a tuple (not a list) of integers.

What to turn in: Display your code and the results of the tests from `hwcode.py`.

Problem 5.4 (tag: rkn)

Write and briefly explain a simple combinatorial expression that takes the number $n$ of quantifiers and an integer $k$ with $0 \leq k < 2^n$ and gives the number $r(k, n)$ of distinct quantifications that correspond to the sequence of quantifiers represented by the binary value of $k$.

[Hints: Remember the rules for distinct quantification. You may use other combinatorial results and functions introduced earlier in this assignment.]

Problem 5.5 (tag: qn)

Write an expression for the number $q(n)$ of distinct quantifications of a matrix with $n$ variables. Explain your expression briefly and clearly.

Your expression should use other results and expressions developed in this assignment.

Problem 5.6 (tag: q)

Write a Python function with header

```python
def q(n):
```

that computes the quantity $q(n)$ defined in the previous problem.

Your code should use other results and expressions developed in this assignment.

Hint: You may want to check that the values your function computes are within the bounds you proved earlier, and that $q(1) = 2$ and $q(2) = 6$. However, do not hand in the code or the results of your checks.

What to turn in: Display your code and the results of the tests from `hwcode.py`.

Problem 5.7 (tag: extra)

You earn 20 extra-credit points for a correct answer to this optional problem, and no partial credit is given.

The expression for $q(n)$ developed in the previous problems involves a summation over $2^n$ terms. Find an expression for $q(n)$ that involves at most a polynomial number of terms. That is, there should be a positive real number $c$ and a natural number $p$ such that your expression can be computed with a number of arithmetic operations that is at most proportional to $cn^p$. Show that your expression is correct (that is, explain how you derived it), and that it involves at most a polynomial number of terms (that is, derive an upper bound of the form $cn^p$ on the number of arithmetic operations, and state what $p$ is).

Warning: I don’t have a solution to this problem.

¹Your function is called `qtype` rather than just `type` because the latter is a reserved Python word.
Part 6: Negating Predicates

The problem in this part refers to the following statement $S$:

$$\text{Nobody likes everyone’s clothes.}$$

We take this statement to hold for some group of people that we do not care to specify, so you can leave the domains of all variables unspecified. Let the predicate $L(x, y)$ mean “$x$ likes $y$’s clothes.”

The point of this exercise is that it may not always be clear how to express the negative of a given predicate. Does the negative of $S$ mean that everyone likes everyone’s clothes? Or that somebody likes somebody’s clothes? Or something else? Formalizing the statement in predicate logic, negating the result, removing negations from the prefix of the formula, and translating back into English shows the correct answer, as shown through the problems for this part.

Problem 6.1 (tag: negate)

Take the following four steps. Number each step in your answer.

1. Express statement $S$ as a fully quantified formula in predicate logic involving the predicate $L(x, y)$. Your formula should have no negation symbol to the left of any quantifier. Negations elsewhere are OK.
   
   Hint: First, write the formula in any form you like. If it has negations in the wrong places, Section 3.3 of the textbook and Section 6.1 of Carlo’s notes on logic show how to switch the negation symbol with one quantifier.

2. Write a predicate-logic formula that expresses the negation $\neg S$ of statement $S$. Again, your formula should have no negation symbol to the left of any quantifier.

3. Write an English sentence that reflects the negated formula $\neg S$ verbatim. In other words, just translate the symbols literally into English, in their order of appearance. Say “person $x$” rather than just “$x$.” This will result in an awkward English sentence, but we’ll fix that later.

4. Rewrite the negated statement $\neg S$ as a clean, well-formed, and simple English sentence.