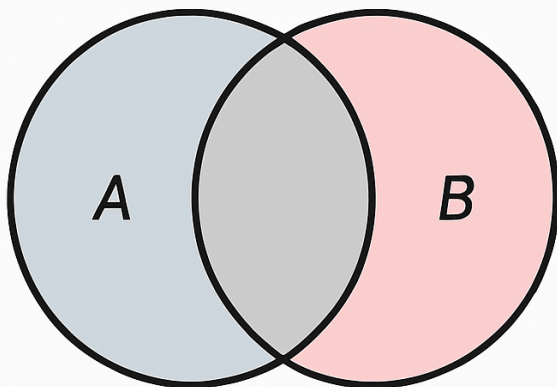


De Morgan's Laws

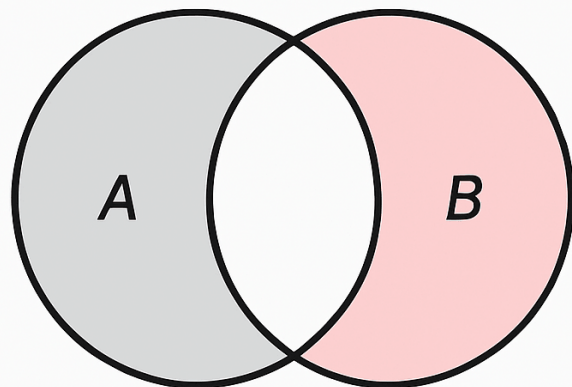
[Read this note online](#)

De Morgan's laws are two equivalences that relate negation, conjunction, and disjunction in logic and set theory. They tell you how to push a negation across a complex statement: the negation of an AND becomes an OR of negations, and the negation of an OR becomes an AND of negations. Formalized by the British mathematician Augustus De Morgan in the 1840s, they're foundational in logic, set theory, Boolean algebra, digital circuit design, and database queries.

De Morgan's laws let you negate a compound condition while preserving its meaning. For set complements, the universe must be specified.



$$\neg(A \wedge B) = \neg A \vee \neg B$$



$$\neg(A \vee B) = \neg A \wedge \neg B$$

De Morgan's laws relate negation, conjunction, and disjunction in Boolean algebra and set theory.

The Two Laws

For any two propositions P and Q , or equivalently for any two sets A and B :

First law: $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$. In **set theory**: $\overline{A \cap B} = \overline{A} \cup \overline{B}$.

Second law: $\neg(P \vee Q) \equiv \neg P \wedge \neg Q$. In set theory: $\overline{A \cup B} = \overline{A} \cap \overline{B}$.

In words: 'NOT (A AND B)' is the same as 'NOT A OR NOT B'; and 'NOT (A OR B)' is the same as 'NOT A AND NOT B'.

Worked Examples in Plain English

Example 1. 'It is not true that I have both an umbrella and a raincoat' is logically the same as 'I don't have an umbrella OR I don't have a raincoat'. (The first form rules out the both-true case; the second form says at least one of them is false, these mean the same thing.)

Example 2. 'There is no one who is both rich and famous' rewrites as 'Everyone is not rich OR not famous'.

Example 3. 'I don't drink coffee or tea' (negating an OR) means 'I don't drink coffee AND I don't drink tea'.

Why They're True: Truth Tables

For two propositions, there are four possible truth assignments. The truth table for $\neg(P \wedge Q)$ gives the same column of values as the truth table for $\neg P \vee \neg Q$, they're true in exactly the same cases. The other law works the same way. Truth tables aren't elegant, but they're conclusive.

- **P:** T
- **Q:** T
- **$P \wedge Q$:** T
- **$\neg(P \wedge Q)$:** F
- **$\neg P$:** F
- **$\neg Q$:** F
- **$\neg P \vee \neg Q$:** F

- **P:** T
- **Q:** F

- $\mathbf{P \wedge Q: F}$
- $\mathbf{\neg(P \wedge Q): T}$
- $\mathbf{\neg P: F}$
- $\mathbf{\neg Q: T}$
- $\mathbf{\neg P \vee \neg Q: T}$
- $\mathbf{P: F}$
- $\mathbf{Q: T}$
- $\mathbf{P \wedge Q: F}$
- $\mathbf{\neg(P \wedge Q): T}$
- $\mathbf{\neg P: T}$
- $\mathbf{\neg Q: F}$
- $\mathbf{\neg P \vee \neg Q: T}$
- $\mathbf{P: F}$
- $\mathbf{Q: F}$
- $\mathbf{P \wedge Q: F}$
- $\mathbf{\neg(P \wedge Q): T}$
- $\mathbf{\neg P: T}$
- $\mathbf{\neg Q: T}$
- $\mathbf{\neg P \vee \neg Q: T}$

Columns 4 and 7 match, the first law is verified. The same procedure verifies the second law.

Set-Theoretic Form

For sets A and B inside a universe U , the complement of an intersection equals the union of complements, and the complement of a union equals the intersection of complements:

$$\overline{A \cap B} = \overline{A} \cup \overline{B}, \quad \overline{A \cup B} = \overline{A} \cap \overline{B}$$

The pictures (shaded Venn diagrams) for the two sides of each equation are identical, which is a visual proof.

Boolean Algebra and Digital Logic

In digital electronics, signals are represented by 0 and 1, and gates implement AND, OR, NOT. De Morgan's laws translate directly:

$$\overline{AB} = \overline{A} + \overline{B}, \quad \overline{A + B} = \overline{A} \cdot \overline{B}$$

In circuit terms: a NAND gate is equivalent to an OR gate with inverted inputs; a NOR gate is equivalent to an AND gate with inverted inputs. This equivalence is critical because real hardware factories typically build a chip using only NAND gates (or only NOR gates), and De Morgan's laws let designers express any logical function using one type of gate.

Applications

- **Simplifying logic.** De Morgan's laws together with distributive and absorption laws let engineers reduce complex Boolean expressions to minimal forms, the basis of Karnaugh maps and Quine-McCluskey minimization.
- **Database queries.** 'WHERE NOT (status = active AND priority = high)' is equivalent to 'WHERE status <> active OR priority <> high'. Rewriting via De Morgan's laws often clarifies query intent and helps the optimizer choose better plans.
- **Programming.** The condition `!(a && b)` is the same as `!a || !b`. Refactoring early-return conditions via De Morgan often makes branches easier to read.
- **Mathematical proof.** Negating a 'for all' statement gives 'there exists' (and vice versa) when applied to predicates. The connection extends De Morgan's laws to quantified logic: $\neg \forall x P(x) \equiv \exists x \neg P(x)$, and $\neg \exists x P(x) \equiv \forall x \neg P(x)$.

Related study notes: Set Theory, Boolean Algebra, Logic Gates, Truth Tables.

The related [OpenStax mathematics textbooks](#) provide additional explanations and exercises for the mathematical definitions used here.

Negating a Compound Condition

A laboratory requires both a valid ID and a safety certificate. Failing the entry condition means an invalid ID, a missing certificate, or both. Negating "ID AND certificate" gives "not ID OR not certificate," with an inclusive OR.

A second rule allows either a staff pass or a visitor pass. Failing that condition requires having neither. Negating "staff OR visitor" gives "not staff AND not visitor." You can verify both transformations by checking all 4 combinations of truth values.

The points to check in this example are:

- Negate each component of the compound statement.
- Negating a conjunction changes AND to OR.
- Negating a disjunction changes OR to AND.

Check Your Understanding

Try the questions before reading the answers.

Question 1. Negate "x is positive and even."

Answer. "x is not positive or x is not even."

Question 2. Why does a set complement depend on U?

Answer. It contains elements outside the set but still inside U.

Practice Questions

Work each question before reading its solution. The set runs from direct recall and substitution to the applied questions that exams actually use to separate grades. The downloadable PDF includes these questions and their solutions.

Question 1. State De Morgan's laws for sets and for propositional logic, side by side.

Solution. Sets: $(A \cup B)^c = A^c \cap B^c$ and $(A \cap B)^c = A^c \cup B^c$. Logic: $\neg(P \vee Q) = \neg P \wedge \neg Q$ and $\neg(P \wedge Q) = \neg P \vee \neg Q$. Same law, 2 vocabularies: union mirrors OR, intersection mirrors AND, complement mirrors NOT.

Question 2. With $U = \{1, \dots, 10\}$, $A = \{1, 2, 3, 4\}$, $B = \{3, 4, 5, 6\}$, verify $(A \cup B)^c = A^c \cap B^c$ by direct computation.

Solution. $A \cup B = \{1, 2, 3, 4, 5, 6\}$, so $(A \cup B)^c = \{7, 8, 9, 10\}$. Separately: $A^c = \{5, 6, 7, 8, 9, 10\}$, $B^c = \{1, 2, 7, 8, 9, 10\}$, and $A^c \cap B^c = \{7, 8, 9, 10\}$. Both routes land on the identical set.

Question 3. Negate the statement "It is raining AND I brought an umbrella" using De Morgan's law, in plain English.

Solution. "It is NOT raining OR I did NOT bring an umbrella." The negation of an AND statement is an OR of negations, and this exact move is why "not (raining and umbrella)" cannot be casually restated as "not raining and not umbrella," a very common error.

Question 4. Negate "The switch is on OR the light is broken."

Solution. "The switch is NOT on AND the light is NOT broken." Negating an OR flips it to an AND of negations: both original disjuncts must fail for the negation to hold.

Question 5. Simplify $\neg(\neg P \vee Q)$ step by step using De Morgan and double negation.

Solution. $\neg(\neg P \vee Q) = \neg(\neg P) \wedge \neg Q = P \wedge \neg Q$. Apply the law once to split the negation across the OR, then simplify the double negative. Nested negations resolve by repeated, careful application, never by guessing.

Question 6. A database query needs records where NOT (city = "Delhi" OR city = "Mumbai"). Rewrite it as an AND of simple conditions.

Solution. city $\neq$ "Delhi" AND city $\neq$ "Mumbai". This is De Morgan applied directly to a WHERE clause, and query optimizers perform exactly this transformation internally to decide which index strategy to use.

Question 7. In digital logic, express a NOR gate (NOT-OR) using only AND and NOT gates, via De Morgan.

Solution. $\text{NOR}(A, B) = \neg(A \vee B) = \neg A \wedge \neg B$: invert both inputs, then AND them. This identity is why NAND and NOR gates are called "universal": either one alone, wired cleverly with De Morgan's law as the blueprint, can build every other logic gate, which is why real chips are built almost entirely from just one gate type.

Question 8. Prove $(A \cap B)^c = A^c \cup B^c$ using an element argument (not a Venn diagram).

Solution. $x \in (A \cap B)^c \iff x \notin (A \cap B) \iff \text{not } (x \in A \text{ and } x \in B) \iff (x \notin A) \text{ or } (x \notin B)$ [propositional De Morgan] $\iff x \in A^c \cup B^c$. Each $\iff$ is reversible, so the sets contain exactly the same elements. This is the rigorous proof that the Venn picture only illustrates.

Question 9. A security policy denies access unless (has_badge AND is_authorized). Using De Morgan, state the exact condition under which access is denied.

Solution. Denied when NOT(has_badge AND is_authorized) = NOT has_badge OR NOT is_authorized: missing the badge, OR lacking authorization, OR both. Writing access-control logic without this translation is a classic source of security bugs, where a programmer negates only 1 of the 2 conditions and accidentally grants access that should be denied.

Question 10. Why do De Morgan's laws hold in Boolean algebra, set theory, AND digital circuits despite looking like 3 unrelated subjects?

Solution. All 3 are instances of the same abstract structure, a Boolean algebra: a set with 2 operations (join/meet, or OR/AND, or union/intersection) and a complement operation obeying identical axioms. Sets, propositions, and voltage-level circuits are simply 3 different concrete realizations of one abstract algebraic skeleton, so any law provable from the axioms, De Morgan included, automatically holds in every realization.

Frequently Asked Questions

What are De Morgan's laws?

Two logical equivalences: (1) $\text{NOT} (P \text{ AND } Q) \equiv \text{NOT } P \text{ OR } \text{NOT } Q$; (2) $\text{NOT} (P \text{ OR } Q) \equiv \text{NOT } P \text{ AND } \text{NOT } Q$. In set theory: the complement of an intersection equals the union of complements; the complement of a union equals the intersection of complements. They tell you how to push a negation across a compound statement.

Who was De Morgan?

Augustus De Morgan (1806-1871) was a British mathematician who formalized these laws around 1847. He was a contemporary of George Boole and helped shape the modern field of mathematical logic. The laws were known earlier to medieval logicians and to George Boole, but De Morgan's name stuck because of his 1847 systematic treatment.

How do De Morgan's laws apply to programming?

The boolean expression `!(a && b)` is logically equivalent to `!a || !b`. Refactoring with De Morgan's laws often makes conditional logic clearer, especially when you can flip a condition for an early-return or to avoid double negation. Useful in code review when simplifying compound boolean tests.

How do De Morgan's laws relate to logic gates?

A NAND gate is functionally identical to an OR gate with inverted inputs; a NOR gate is identical to an AND gate with inverted inputs. This identity comes directly from De Morgan's laws. Chip manufacturers often build everything from NAND (or NOR) alone, because that single gate plus De Morgan's laws lets them implement any logical function.

Do De Morgan's laws apply to quantifiers?

Yes, in predicate logic. The quantifier versions are: $\text{NOT} (\text{for all } x: P(x)) \equiv \text{there exists } x \text{ with } \text{NOT } P(x)$; and $\text{NOT} (\text{there exists } x: P(x)) \equiv \text{for all } x: \text{NOT } P(x)$. These are very useful when proving statements by contradiction or contraposition.

Are De Morgan's laws used in SQL?

Yes. The WHERE clause condition `NOT (a = 1 AND b = 2)` is equivalent to `(a <> 1 OR b <> 2)`. Rewriting via De Morgan's laws often makes complex WHERE clauses easier to read and can help the query optimizer choose better execution plans, particularly when conditions interact with indexes.