M4_Logic_1

Wednesday, 7 January 2026 9:20 AM

M4_Logic_s...

Newton School of Technology

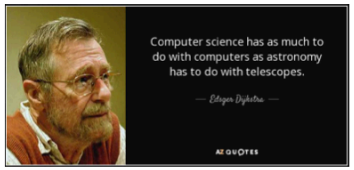
Logic - Sheet 1

Lecture Credo

> *"I see, I forget.*
> *I hear, I remember.*
> *I do, I understand."*
>
> — Confucius

In this lecture we will learn about:

- Introduction to Logic
- Propositional Logic
- Predicate Logic
- Quantifiers



Student Name: _____

Student ID: _____

Semester: _____

1 Introduction to Logic

Welcome to the first lecture of Mathematics-4, Discrete mathematics and we will be begin with logic.

Self Reflection

Do you believe you already know logic? Write your thoughts below,
Try to define logic.

Let's try to understand the importance of logic first from history and language.
Question: What do you mean by *trivial* ?

Why Logic Matters

The word *trivial* has an interesting etymology. It is composed of *tri* (meaning "3") and *via* (meaning "ways"). It originally referred to the *trivium*, the three fundamental curriculae: grammar, rhetoric, and logic. Mastery of these subjects was considered essential before study could continue with the *quadrivium*, which consisted of arithmetic, geometry, music, and astronomy.

Why was logic considered to be fundamental to one's education? To answer this question, it is necessary to explain what we mean by the term *logic*.

Lewis Carroll, Through the Looking-Glass
"Contrariwise," continued Tweedledee, "if it was so, it might be; and if it were so, it would be; but as it isn't, it ain't. That's logic."

"अगर ऐसा होता, तो शायद ऐसा होता; और अगर ऐसा ही होता, तो यह होता; लेकिन क्योंकि ऐसा नहीं है, तो यह नहीं है। यही तर्क है।"

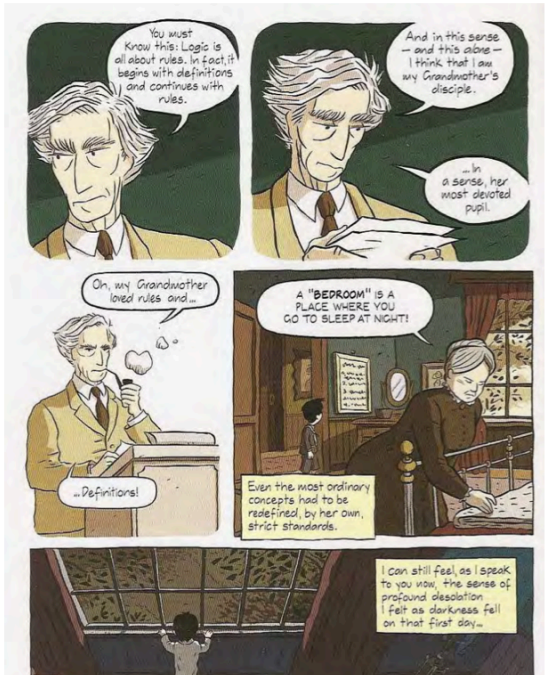
Table 1: **From debate → algebra → foundations → computation**
Logic evolves by progressively removing ambiguity.

Age	Period	Key Figures	Focus & Impact
1. Symbolic (Philosophical) Logic	500 B.C. – 19th Century	Sophists, **Socrates**, **Francis Bacon**	Logic dealt with arguments in **natural language**.
2. Algebraic Logic	Mid–Late 19th Century	**George Boole**, **Lewis Carroll**, **Ernst Schröder**	Logic reformulated using **mathematical symbols**.
3. Mathematical Logic	Late 19th – Mid 20th Century	**Gottlob Frege**, **Georg Cantor**, **David Hilbert**, **Bertrand Russell**, **Kurt Gödel**, **Alan Turing**	Logic became the **language of mathematics**.

4. Logic in Computer Science	Modern Day	(Try to fill this box)	Logic functions as the **"calculus of computer science."** Core to Boolean circuits, databases (SQL ≈ first-order logic), AI reasoning, program semantics, and formal verification.

LOGIC

Why logic is necessary for everything above - it is the core foundation: Everything else is resting on this !



The above image is one of the pages from the book, *Logicomix* is a graphic novel about **Bertrand Russell**, focusing on his and other's work on the foundations of mathematics which is essentially logic. Its structure is a storyline within a frame within a frame. The link of the pdf is available on google search.

2 Propositonal Logic

Propositional Logic

Step 1: What is a Proposition?

(In words)

A proposition is a sentence that can be **True** or **False**.

Write two examples of propositions:

Write two examples that are *not* propositions:

Step 2: Naming Propositions

(In words)

We name propositions using symbols $p, q, r, \ldots$

Example:
p: "It is raining"
q: "The ground is wet"

Your turn: Define p and q from your daily life.

p:

q:

Programming Structure

Step 1: Boolean Expressions

(In code)

In programs, many expressions evaluate to:

`True` or `False`.

Write 2 Boolean expressions in Python:

Step 2: Boolean Variables

(In code)

We store truth values in variables:

`loggedIn`, `isAdmin`, `hasPaid`, ...

Your turn: pick two Boolean variables used in real code (or imagine).

Mini note: In logic and in programming, we first build *atomic truths* and then combine them.

Propositional Logic

Step 3: Building new propositions by combining atomic propositions

(In symbols)

We combine propositions using operators:

$\neg p \equiv$ (NOT), $\quad p \wedge q \equiv$ (AND)

$p \vee q \equiv$ (OR)

Your turn (complete the truth tables):

(i) NOT p

p	$\neg p$
T	F
F	T

(ii) p AND q

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

(iii) p OR q

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

Programming Structure

Step 3: Logical Operators → Python Code

Logical operators in propositional logic translate directly into Python. This makes programming a natural extension of logical reasoning.

Logical Operators in Python

$\neg p \longleftrightarrow$ `not p` $\quad p \wedge q \longleftrightarrow$ `p and q`

$p \vee q \longleftrightarrow$ `p or q`

Your turn: Complete the Python functions below.

(i) Negation

```
loggedIn = True
def is_not_logged_in(loggedIn):
    return
```

(ii) Conjunction

```
def can_access_admin(loggedIn,
isAdmin):
    return
```

(iii) Disjunction

```
def can_access_content(loggedIn,
hasPaid):
    return
```

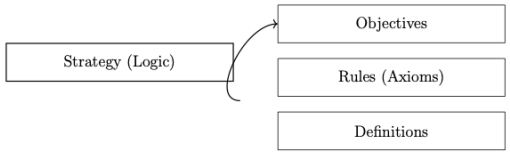

3 The "Game" of mathematics

Idea. Mathematics is not a collection of formulas. It is a **game** played according to clearly stated rules.
A game consists of:

- **Definitions** — the objects we are allowed to talk about,
- **Rules (Axioms)** — the facts we agree to use,
- **Objectives** — what we want to establish,
- **Strategy (Logic)** — how we reach conclusions.

Warm-up (Reflection)

In your own words, what makes chess, programming, or mathematics a *game*?



Let's analyze various game components

1. Definitions Some concepts are so basic that they cannot be defined without circular reasoning. These are called **primitives**.

Examples of primitives you already know:

- In arithmetic: _____
- In geometry: _____

2. Rules (Axioms) Rules tell us how the defined objects are allowed to behave.

Why do we *assume* rules instead of proving everything?

3. Objectives An objective is not just a guess — it must follow ***inescapably*** from the rules.

Before proof, a statement is called a: _____ After proof, it becomes a: _____

Key Insight

A **proof** is the *strategy* of the game.
It shows that a conclusion is not merely true, but an **inescapable consequence** of the definitions and rules.

Bridge to Programming

Complete the analogy:

Definitions	↔	_____
Rules (Axioms)	↔	_____
Proof (Strategy)	↔	_____

Why is testing a program on a few inputs *not* the same as a proof of correctness?

Exercise: Unary Logical Operators

A **unary logical operator** takes one proposition p and produces a new proposition.

That means it assigns a truth value to the output for:

$p = T$ and $p = F$

Question 1. How many *different* unary logical operators are possible on a single proposition p?

To answer this, complete the table below for all possible operators.

p	$\neg p$	$p \wedge \neg p$	$p \vee \neg p$	Identity (p)

Contradict

Tautology

T	f	f	T	T
F	T	F	T	f

Question 2. How many *distinct* truth tables are possible for single proposition?

Your answer: _____

Question 3 (Think carefully). Why are there no other unary operators beyond these?

Terminology Alert

- **Tautology** : A proposition is a tautology if it is **true** under every truth value assignment.
- **Contradiction** : A proposition is a ~~tautology~~ if it is **false** under every truth value assignment.

Question: If there are 4 *unary* logical operators are possible then how many *binary* logical operators are possible ?

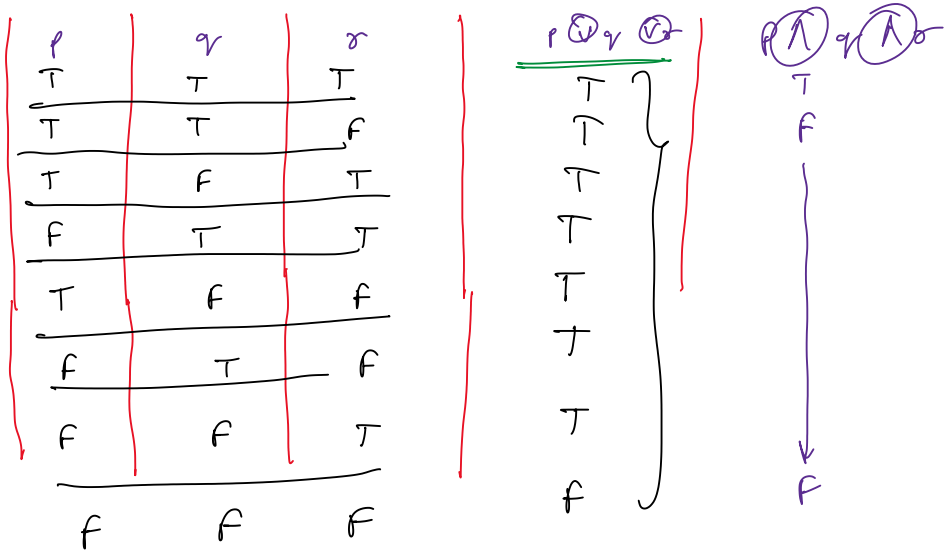
List some of known binary logical operators.

Exercise: Binary Logical Operators

Given below is some of the binary logical operators, task is to recognize the operators. In the table below, the operators are numbered and second table is provided to fill in the answers. All these operators were learned in mathematics and have been used in various scenarios in web-development and subjects like **DSA** and **ADA**.

p	q	①	②	③	④	⑤	⑥
T	T	T	T	T	T	F	F
T	F	F	T	F	F	T	T
F	T	F	T	T	F	T	T
F	F	F	F	T	T	F	T

Operator	Description in words	Name	Symbol
①			
②			
③			
④			
⑤			
⑥			



If I am elected, then achche din aayenge

The Great Confusion -Conditional Statements: $p \to q$

Instead of memorizing a truth table, let us *simulate* what

$p \to q$ ("if p, then q")

actually means.

Dress Code Rule (The Game Rule) *If today is Wednesday, then wear Pink.*

Your task: Decide whether the rule is **violated** in each situation.

Day	p	Color	q	Rule violated?		$p \to q$
Wednesday	T	Pink	T	Yes	~~No~~	T
Wednesday	T	Black	F	~~Yes~~	No	F
Tuesday	F	Pink	T	Yes	~~No~~	T
Monday	F	Red	F	Yes	~~No~~	T

Reflection (one line): When does an "if–then" rule actually fail?

Rearranging a Conditional

Given a conditional statement:

$p \to q$

Fill in the missing parts:

Name	Statement
Original	$p \to q$
Converse	~~$q \to p$~~
Inverse	~~$\neg p \to \neg q$~~
Contrapositive	~~$\neg q \to \neg p$~~

T F F T

If it is raining, then ground is wet. T
If the ground is wet, then it is raining T or F
If it not raining, ground not wet. T or F
If the ground is not wet, then it is not raining T

Question: Which of these feels logically closest to the original? Why?

YOU CAN LOGIN IF AND ONLY IF YOU HAVE THE CORRECT PASSWORD

p q

Biconditional: $p \leftrightarrow q$

A biconditional means:

"p if and only if q"

Interpretation: Both directions must hold.

$p \to q$ and $q \to p$

Your example (real life):

A Strange Fact (Hold This Thought)

From a false statement, any conclusion can follow.
Why does logic allow this?
We will return to this when we study Rules of Inference.

In natural language, biconditionals are often *implicit*.

People rarely say "if and only if." Instead, they say things like:

"If you finish your meal, then you can have dessert."

What is usually *meant* is:

"You can have dessert **if and only if** you finish your meal."

That single English sentence silently includes **two** logical statements:

$$p \to q \quad \text{and} \quad q \to p$$

Key Point: English often hides the converse. Logic never does.

Because precision is essential in mathematics and computer science, we will **always distinguish** between:

$$p \to q \quad \text{(conditional)} \qquad \text{and} \qquad p \leftrightarrow q \quad \text{(biconditional)}$$

Follow-Up Exercise: "If" vs "If and Only If"

Think of a number between 10 and 1,000,000.

Define the following propositions:

p : "Your number is prime."
q : "Your number is even."
r : "Your number is evenly divisible by a positive integer other than 1 and itself."

Translate the following English sentences into logical notation.

(1) *If the number you're thinking of is even, then it isn't prime.*

Logical form: $q \to \neg p$

(2) *The number you're thinking of isn't prime if it's evenly divisible by an integer other than 1 and itself.*

Logical form: $q \to \neg p$

Reflection (important):
Which English word caused the logical meaning to change? Why does English hide this difference so easily?

Remember, in the **game** of mathematics, we setting the definitions and rules of the

game, for Logic, we have to respect certain rules. One of the rules is setting the precedence of logical operators.

negation	$\neg p$	"not p"	*highest precedence (binds tightest)*
conjunction	$p \wedge q$	"p and q"	
disjunction	$p \vee q$	"p or q"	
exclusive or	$p \oplus q$	"p xor q"	
implication	$p \Rightarrow q$	"if p, then q" or "p implies q"	
if and only if	$p \Leftrightarrow q$	"p if and only if q"	*lowest precedence (binds loosest)*

[1]

[1]The word *precedence* (*pre* meaning "before," *cede* meaning "to go") means "what comes first," so the precedence rules tell us the order in which the operators "get to go."

Application in Computer Science: System Specifications

In computer systems, **specifications must be consistent**. If requirements contradict each other, then *no system can satisfy them all.*

Consider the following system specifications:

1. The diagnostic message is stored in the buffer **or** it is retransmitted.
2. The diagnostic message is **not** stored in the buffer.
3. If the diagnostic message is stored in the buffer, then it is retransmitted.

Step 1: Define propositions
Let

p : "The diagnostic message is stored in the buffer."
q : "The diagnostic message is retransmitted."

Step 2: Translate into logic

(1) $p \vee q$
(2) $\neg p$
(3) $p \rightarrow q$

Step 3: Consistency Check
Is there an assignment of truth values to p and q that makes *all* specifications true?

Your conclusion: Consistent Inconsistent

Justify your answer (briefly):

Consider the following two logical expressions:

$$\neg(p \vee q) \quad \text{and} \quad \neg p \wedge \neg q$$

Step 1: Build the truth table.

p	q	$p \vee q$	$\neg(p \vee q)$	$\neg p$	$\neg p \wedge \neg q$
T	T				
T	F				
F	T				
F	F				

Question: What do you observe about the last two columns?

Explain your reasoning (one sentence):

Compound propositions that have the same truth values in all possible cases are called **logically equivalent**. We can define this notion as follows.

Terminology Alert: Logical Equivalence

Two compound propositions p and q are called **logically equivalent** if the biconditional

$$p \leftrightarrow q$$

is a **tautology**.

We write

$$p \equiv q$$

to denote that p and q are logically equivalent.

Meaning: Propositions p and q have the *same truth value for every possible assignment* of truth values to their variables.

These logical equivalences are especially useful in programming for optimizing code, particularly when writing long lists of `if` conditions. Many such scenarios can be modeled using propositional logic, allowing the resulting code to be written more efficiently and clearly.

The following pages demonstrate this idea using an example in which a control logic is required to monitor three independent sensors.

System Control Logic (Given)

Consider the following (correct but repetitive) code:

```
if (temp_high and not pressure_high):
    enter_safe_mode()

if (temp_high and not vibration_detected):
    enter_safe_mode()

if (temp high and not pressure high and
```

```
    not vibration_detected):
    enter_safe_mode()
```

Let:

p : temperature is high
q : pressure is high
r : vibration is detected

Step 1: Rewrite each condition using logical symbols.

(1) _____
(2) _____
(3) _____

Question: What part of the condition appears in *every* line?

_____

Step 2: Propose a Replacement

Propose a *single logical condition* that you think captures exactly when `enter_safe_mode()` should run.

_____

Important: Your condition must trigger *exactly* in the same cases as before.

4 Predicate Logic

Activity: One Sentence Is Not Enough

Consider the following statements:

- "Masha loves pizza."
- "Dasha loves pizza."
- "Sasha loves pizza."

Question 1. What do all these sentences have in common?

_____

Question 2. Why does writing many such sentences feel inefficient?

_____

Instead of writing many similar statements, we write:

"_____ loves pizza."

Here, the blank can be replaced by different names.

Question 3. What role does the blank play?

_____

Question 4. Is this sentence true or false *before* filling the blank?

_____

a loves pizza

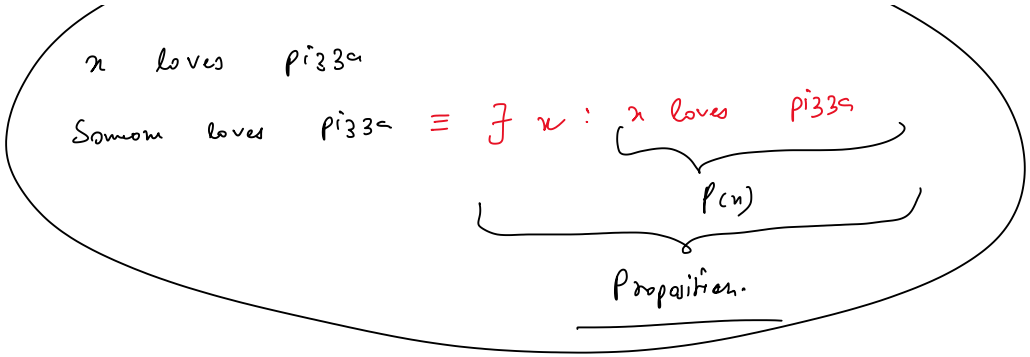
Terminology Alert

A sentence with a placeholder (like a blank or variable) that becomes true or false only after substitution is called a **predicate**.

Exercise: Predicate or Proposition?

Decide whether each item is a **predicate** or a **proposition**.

1. "Sasha loves pizza." Proposition
2. "____ loves pizza." Predicate
3. "Everyone loves pizza." Proposition
4. "Someone loves pizza." Proposition



Hidden Logical Structure

Compare the two sentences:

- "Everyone loves pizza."
- "Someone loves pizza."

Question. Why is one much stronger than the other?

Think: Which sentence becomes false more easily?

Connection to Programming

In programming, a predicate behaves like a **Boolean function**.

Example (Python-style thinking):

```
def loves_pizza(person):
"""
Predicate: loves_pizza(person)
Returns True or False
"""
response = input(f"Does {person} like pizza? (yes/no): ")
return response.lower() == "yes"
```

Question. Is `loves_pizza(x)` true or false *before* we give a value to `x`?

17

Exercise: Predicates vs Propositions

Recall: A **predicate** becomes a **proposition** only after we substitute specific values for its variables.

Consider the following predicates:

1. *isPrime*(n): "The positive integer n is a prime number."
2. *isPowerOf*(n, k): "n is an exact power of k."
3. *onlyPowersOfTwo*(S): "Every element of the set S is a power of two."
4. $Q(n, a, b)$: "$n = a + b$ and both a and b are prime."
5. *sumOfTwoPrimes*(n): "n is the sum of two prime numbers."

Part A: Evaluating Propositions

For each expression below:

- State whether it is **True** or **False**
- Briefly explain *why*

1. is Prime(261) True / False

   Reason:

2. is Prime(262) True / False

   Reason:

3. $Q(8, 3, 5)$ True / False

   Reason:

4. $Q(9, 3, 6)$ True / False

   Reason:

Someone loves pizza.

x loves pizza.

18

Part B: What Kind of Object Is This?

For each expression, decide whether it is a: **Predicate** or a **Proposition**.

1. is Prime(n) Predicate / Proposition
2. is Prime(261) Predicate / Proposition
3. sum Of TwoPrimes(n) Predicate / Proposition

4. sum Of Two Primes(10) Predicate / Proposition

Explain the rule you used to decide:

Exercise: Implementing a Predicate

Recall the predicate:

isPrime(n) : "n is a prime number."

In programming terms, this corresponds to a Boolean function.

Task. Write a Python function `is_prime(n)` that returns `True` if n is prime and `False` otherwise.

Important:

- You may assume n is a positive integer.
- Focus on *correctness*, not efficiency.

Implementation Space

```
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0
            return False
    return True
```


5 Quantifiers

We have seen that we can form a proposition from a predicate by applying that predicate to a particular argument. However, we can also form a proposition from a predicate by using *quantifiers*. Quantifiers allow us to formalize statements such as:

- *Every JavaScript program contains at least four* `for` *loops* (false!), or
- *There exists a proposition that cannot be expressed using only the connectives* $\wedge$ *and* $\vee$ (true!).

These types of statements are expressed using the two standard quantifiers: the **universal** quantifier ("every") and the **existential** quantifier ("some").

Terminology Alert: Universal Quantification

The **universal quantification** of a predicate $P(x)$ is the statement:

"$P(x)$ for all values of x in the domain."

The notation

$$\forall x\, P(x)$$

denotes the universal quantification of $P(x)$.
We read $\forall x\, P(x)$ as:

- "for all x, $P(x)$" or
- "for every x, $P(x)$"

An element for which $P(x)$ is false is called a **counterexample** to $\forall x\, P(x)$.

Terminology Alert: Existential Quantification

The **existential quantification** of a predicate $P(x)$ is the statement:

"There exists an element x in the domain such that $P(x)$."

The notation

$$\exists x\, P(x)$$

denotes the existential quantification of $P(x)$.
Here, $\exists$ is called the **existential quantifier**.

Exercise 1: Where Do We Need Quantifiers?

Consider the following predicates:

1. isPrime(n)
2. isPowerOf(n, k)
3. onlyPowersOfTwo(S)
4. $Q(n, a, b)$
5. sumOfTwoPrimes(n)

For each predicate, answer:

- Does it require a quantifier? (Yes / No)
- If yes, which one? ($\forall$ or $\exists$)

Predicate	Quantifier needed?	Which one?
isPrime(n)		
isPowerOf(n, k)		
onlyPowersOfTwo(S)		
$Q(n, a, b)$		
sumOfTwoPrimes(n)		

Exercise 2: Universal Quantifier

The predicate `onlyPowersOfTwo(S)` means:

"Every element of the set S is a power of two."

Complete the formal structure below:

onlyPowersOfTwo(S) = $\forall$ ___x___ $\in$ ___S___ : ___x is a power of 2.___

Reflection: Why does a single counterexample destroy this statement?

$S = \{ 2^k \mid k \text{ is an integer} \}$

$\forall n \in S : P(n)$ | $\forall n \in S :$ n is a power of 2

$\exists n \in \mathbb{R} : P(n)$ | $\exists n \in \mathbb{R} :$ n is a power of 2

21

Exercise 3: Existential Quantifier

The predicate `sumOfTwoPrimes(n)` means:

"n can be written as the sum of two prime numbers."

Complete the logical structure below:

sumOfTwoPrimes(n) = $\exists$ ____, ____ $\in$ _____ : _____

Think carefully: Why is *one example* sufficient to prove this predicate true?

22

Quantifiers in Logic	Quantifiers as Loops
1) Universal Quantifier ∀	**A) ∀ behaves like "all()"**

Meaning (in English):
$\forall x \in S : P(x)$ means:

"For every x in S, $P(x)$."

Counterexample idea:
If we find one x such that $P(x)$ is false, then the whole statement is false.

2) Existential Quantifier $\exists$

Meaning (in English):
$\exists x \in S : P(x)$ means:

"There exists some x in S such that $P(x)$."

Witness idea:
If we find x such that $P(x)$ is true, then the whole statement is true.

3) Finite-domain expansion

Let $S = \{x_1, x_2, \ldots, x_n\}$. Then:

$$\forall x \in S : P(x) \equiv P(x_1) \wedge P(x_2) \wedge \cdots \wedge P(x_n)$$

$$\exists x \in S : P(x) \equiv P(x_1) \vee P(x_2) \vee \cdots \vee P(x_n)$$

Quick Check

Which statement is easier to *disprove*?
(Circle one)

$\forall x \in S : P(x)$ $\exists x \in S : P(x)$

Goal: return True only if every element passes.

```
for x in S:
  if not P(x):
    return False
return True .
```

Fill: `not P(x)`, `True`, `False`

B) $\exists$ behaves like "any()"

Goal: return True as soon as one element passes.

```
for x in S:
  if P(x):
    return True
return false.
```

Fill: `P(x)`, `True`, `False`

Reflection (1–2 lines)

Why does the $\forall$ loop stop early when it finds a counterexample?

Important note (domain matters):
This loop analogy makes literal sense when S is _____.

6 Application of Logic - Database Systems - Do it yourself

As a rough approximation, we can think of a database *as a two-dimensional table, where rows correspond to individual entities, and columns correspond to fields (data about those entities). A database* query *defines a predicate $Q(x)$ that consists of tests of the values from various columns, joined by the basic logical connectives. The database system then returns a list of rows/entities for which the predicate is true. We can think of this type of database access as involving predicates: in response to query Q, the system returns the list of all rows x for which $Q(x)$ is true. An example is shown in Figure below Each of the following predicates $Q(x)$ uses tests on particular columns in x's row. For each, give a logically equivalent predicate* in which each column's name appears at most once. *You may also use the symbols* True *and* False *and the four most common logical connectives ($\neg, \wedge, \vee, \Rightarrow$) as many times as you please. Use a truth table to prove that your answer is logically equivalent to the given predicate.*

$[age(x) < 18] \vee \big([age(x) \geq 18] \wedge [gpa(x) \geq 3.0]\big)$

$takenCS(x) \Rightarrow \neg\Big([home(x) = \text{Hawaii}] \Rightarrow \big([home(x) = \text{Hawaii}] \wedge takenCS(x)\big)\Big)$

$\big(hasMajor(x) \wedge [schoolYear(x) \neq 3] \wedge onCampus(x)\big) \vee \big(hasMajor(x) \wedge [schoolYear(x) \neq 3] \wedge \neg onCampus(x)\big)$
$\quad \vee \big(hasMajor(x) \wedge [schoolYear(x) = 3] \wedge \neg onCampus(x)\big)$

name	*GPA*	*CS taken?*	*home*	*age*	*school year*	*on campus?*	*has a major?*
Alice	4.0	yes	Alberta	20	3	yes	yes
Bob	3.14	yes	Bermuda	19	3	yes	no
Charlie	3.54	no	Cornwall	18	1	no	yes
Desdemona	3.8	yes	Delaware	17	2	no	no

A sample database. Here, to find a list of all students with grade point averages over 3.4 who have taken at least one CS course if they're from Hawaii, we could query $[GPA(x) \geq 3.4] \wedge (takenCS(x) \Leftarrow [home(x) = \text{Hawaii}])$. For this database, this query would return Charlie (and not Alice, Bob, or Desdemona).

24