LECTURE NOTES IN

DISCRETE MATHEMATICS

Rey Joseph Nieto

Instructor

Bulacan State University



Updated: 25 November 2026

Introduction

Tell someone in the Philippines that you are studying Information Technology, and you may be asked whether you can fix a computer, install software or get a printer working. Hardware maintenance and technical support are valuable parts of IT, and you may eventually choose to specialise in them. The difficulty arises when this becomes our entire picture of the field.

Information Technology concerns how people and organisations use computing systems to work with information. The University of Cincinnati's overview describes a field involving hardware, software, networks, data management and information security, and includes artificial intelligence and machine learning among its areas.[1] Maintaining equipment is one way to contribute. Designing an application, organising a database, connecting services and deciding who should have access to a record are other ways. These tasks require us to understand what a system is meant to do and how its parts fit together.

Consider a university enrolment system. The computers and network must work, but the system also needs a way to identify each student, record completed subjects, check prerequisites and allocate places. If Alice has completed the required subjects, should she be allowed to enrol? What if a class is full? Can two students be assigned the same student number? A fast computer will faithfully execute a badly specified rule. Before choosing how to implement a system, we need to say clearly what its rules mean.

Discrete mathematics

The word **discrete** refers to distinct, separately identifiable objects. A student record, a file, a network connection and a step in a procedure can each be considered individually. Discrete mathematics studies structures built from such objects and the relationships between them. Its collections need not be small or even finite: the integers, for example, form an infinite discrete collection.

These notes introduce discrete mathematics at an early undergraduate level for Information Technology and Computer Science majors. They support IT 104, *Discrete Mathematics*, at Bulacan State University.

For comparison, a model of temperature may allow every real value in an interval. A model of enrolment may instead ask which students belong to a class and which pairs of subjects stand in a prerequisite relationship. The questions call for different mathematical descriptions. In this course, we concentrate on the descriptions useful for objects, connections, rules and successive steps.

The renewed prominence of **artificial intelligence (AI)**[2] makes this broader understanding of IT especially relevant. AI concerns computing methods for tasks such as recognising patterns, reasoning and generating language. Its growing use in organisations brings questions about data, software and system behaviour into more areas of work.[3]

[1] University of Cincinnati Online, *What is information technology? An in-depth guide*.

[2] Earlier waves included symbolic AI, or "Good Old-Fashioned AI" (GOFAI), based on explicit symbols and rules. Today's resurgence centres largely on deep neural networks, especially large language models (LLMs). Neural-network research is itself decades old. See Stanford's *A Short History of AI* (2016) and Bommasani et al., *On the Opportunities and Risks of Foundation Models* (2021).

[3] Stanford Institute for Human-Centered Artificial Intelligence, *The 2026 AI Index Report*, Economy chapter, on the increase in organisational AI use during 2025.

Machine learning is an approach within AI in which models are developed using data.[4] For example, a system might use previously labelled messages to learn to classify new messages as spam or legitimate mail. People still have to decide which messages to collect, what the labels mean, how to test the model and what should happen when it makes a mistake. Integrating such a model into a useful service also requires storage, permissions and a clear account of its inputs and outputs.

Mathematical reasoning helps us ask precise questions about these systems. If a spam classifier labels twenty test messages correctly, what does that establish about messages it has never seen? It provides evidence about those twenty cases; it does not prove that every future classification will be correct. Statistical methods help us assess performance beyond those test cases. Here we begin with the logical distinction between checking some examples and establishing a claim about every permitted case.

Discrete mathematics helps us describe objects and relationships, express conditions, analyse choices and justify conclusions.

AI-assisted proofs and code

AI systems already produce proofs that can be checked by formal tools. AlphaProof, for example, produced formal solutions to three problems from the 2024 International Mathematical Olympiad using Lean, software that checks proofs written in a precise formal language.[a]

There is also the claimed solution to the Navier–Stokes Millennium Prize Problem, announced by OpenAI in September 2026 with a Lean formalisation. An October paper challenged whether that formalisation corresponds to the mathematical proof being claimed.[b] A machine may check an argument for a precisely stated claim, while we still need to establish that this is the claim we intended to prove.

The same care is needed in AI-assisted coding, including what is often called "vibe coding". Generated code can contain logical errors even when it runs and produces plausible results.[c] You need enough understanding to examine the reasoning behind its steps, choose tests that could expose a mistake and judge whether it does what was intended. This course will develop that judgment through proofs that you can read, write and discuss yourself.

[a] Hubert et al., *Olympiad-Level Formal Mathematical Reasoning with Reinforcement Learning*, *Nature* (2025).

[b] OpenAI, *On the Navier–Stokes Millennium Prize Problem* (2026); Bastounis, Circelli and Hansen, *Navier–Stokes lost in translation: Why Lean verification of AI autoformalisation does not guarantee correct natural language proofs* (2026), a preprint challenging the correspondence between the formalisation and the announced proof.

[c] GitHub, *Review AI-generated code*, especially "Verify context and intent" and "Spot AI-specific pitfalls".

Using the notes

Our primary textbook is Oscar Levin's *Discrete Mathematics: An Open Introduction*, fourth edition. These lecture notes are the authoritative reference for the course's coverage, notation and conventions. Use Levin for further explanations, examples and practice.

I may make minor revisions as the course progresses to reflect the needs of the discipline. Substantive changes are unlikely.

The chapters may look long because I have been generous with worked examples. Much of that space explains and applies the same ideas in different situations, showing the steps of the reasoning rather than adding more topics for you to learn.

[4] The US National Institute of Standards and Technology describes machine learning in terms of systems that learn and adapt from data: *Machine Learning*, CSRC Glossary.

I have included definitions, explanations and worked steps so that you can follow the arguments on your own. You are welcome to study further ahead of the class and email me when a step is unclear. We will still develop the material together in lectures and problem classes.

The notes contain more examples and exercises than you are likely to need. Beyond the work I assign, they are available for enrichment, revision and further practice; I do not expect you to work through everything each week.

The short Python illustrations are optional and **non-examinable**. You will not be assessed on writing, reading or running Python code. All Python-specific explanations, code and output are contained in boxes headed *Python (non-examinable)*; the label is repeated if a box continues onto another page. You can skip these boxes without missing any steps in the mathematical discussion. The mathematics they illustrate remains examinable unless separately marked otherwise. Each illustration includes its input, output and the syntax needed to follow it, so no prior knowledge of Python is assumed. You can trace the steps on paper or run them independently using Python 3.

Working through varied problems and understanding why your solutions work help you recognise familiar patterns and decide how to proceed with less hesitation. This can improve your speed in time-limited examinations and other situations where you need to work efficiently. With this kind of practice, your mathematical reasoning can become increasingly instinctive, rather than depending only on memorised steps.

Course outline

The course has twelve chapters.

Chapters 1–4 develop mathematical language and proof. We learn to say exactly which objects a claim concerns, what its conditions mean and why a conclusion follows. An enrolment rule may require a completed prerequisite *and* an available place; changing "and" to "or" changes who qualifies. Sets and functions help us describe the objects and assignments; logic and proof help us interpret and check the rules.

Chapter 5 completes this initial toolkit with weak and strong induction. These methods let us prove suitable claims across all natural-number indices by establishing starting cases and showing how later cases follow from earlier ones.

Chapters 6–7 study graphs, trees and relations. A network can be described through its devices and connections, while a folder hierarchy can be represented by a tree. A relation records which objects are linked in a specified way, such as which students are enrolled in which subjects.

Chapter 8 follows equivalence classes with divisibility, remainders and modular arithmetic. We group integers by the remainder left on division by a fixed positive integer, then calculate with the resulting classes. For example, a system with twelve positions might cycle from 0 through 11 and then return to 0.

Chapters 9–10 develop counting. If a system offers several choices, how many possible outcomes are there? When do two descriptions count the same outcome twice? Such questions help us organise possibilities and understand why listing every case can quickly become impractical.

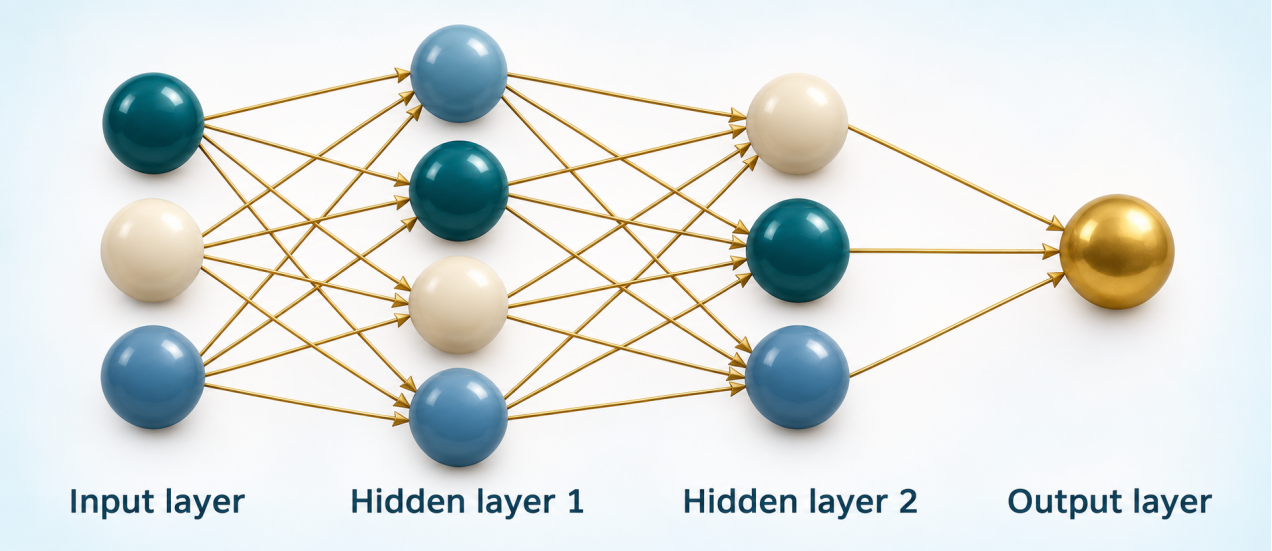
Chapter 11 develops our work on sequences and recursive rules. A rule may describe a later stage using earlier stages; we can use induction from Chapter 5 to verify formulas for these sequences.

Chapter 12 brings together functions, recursive structures and structural induction, building on the earlier work on functions, trees and recursion. We specify what a procedure takes as input, what it returns and how its base cases and recursive rules determine its behaviour. Structural induction then helps us prove properties of objects built by repeated construction rules. We will develop one family of full binary trees and the functions that count their leaves and internal vertices, then prove a relationship between those counts.

Abstraction and computing

The most important skill I want you to gain from proofwriting is **abstraction**: identifying the objects, rules and relationships that matter, and organising them clearly in your mind. Instead of reasoning only about one example, you learn to work with a description that applies to many cases.

Suppose the spam classifier uses an **artificial neural network**, one of the models used in AI. Its interconnected computational units transform numerical inputs into outputs. In a simple feedforward network, information moves through successive layers without feeding back. Numbers called *parameters* control the transformations and are adjusted during training.[5]



A feedforward neural network with three inputs, two hidden layers and one output.

Read the illustration from left to right. The three inputs could record a message's length, number of links and number of occurrences of a chosen word. Units in the hidden layers perform intermediate calculations; each combines numbers from the preceding layer and transforms the result. The output is a single score. A connection's *weight*, one of the parameters adjusted during training, controls its contribution to the receiving unit's calculation.

The spheres are vertices and the arrows are directed edges: together they form a **graph**. Sets describe the permitted inputs and outputs. With the parameters fixed, each hidden layer and the output layer define a **function**; applying these in turn composes the functions. The calculation has a recursive description: each list of numbers is computed from the list at the preceding layer.

Counting tells us that the illustrated network has $3 \cdot 4 + 4 \cdot 3 + 3 \cdot 1 = 27$ connections: every unit in one column is connected to every unit in the next.

We could label a message as spam when its output score is at least a chosen cutoff, and as legitimate otherwise. **Logic** lets us state that condition precisely. With the network and cutoff fixed, messages receiving the same predicted label form **equivalence classes**.

During training, an update rule uses the current parameter settings and training data to obtain the next settings. Tracking one parameter across these updates gives a numerical **sequence**. A routine could cycle

[5] Ian Goodfellow, Yoshua Bengio and Aaron Courville, *Deep Learning* (2016), Chapter 6: Deep Feedforward Networks.

through ten examples numbered 0 to 9. Adding 1 to the current index and taking the remainder on division by 10 returns it to 0 after 9, an application of **modular arithmetic**.

Induction can help justify that a layer-by-layer evaluation follows the specified composition of functions. Such a proof concerns the computation; it does not establish that every new message will be classified correctly.

A fuller mathematical study of neural networks usually assumes **linear algebra**, **calculus** and **probability**, alongside statistics. Linear algebra organises lists of numbers and their weighted transformations. Calculus describes how outputs change as parameters change, guiding many training methods. Probability and statistics help us model uncertainty and assess predictions using data.[6]

Our work supplies some of the foundations used in those subjects. Sets, functions and proofs provide a language for their definitions and arguments. Linear algebra studies particular kinds of functions; probability assigns numbers to sets of outcomes; and sequences help calculus describe how values approach a limit. Counting supports probability models with finitely many outcomes.

I have designed this module as a foundation for later AI study, particularly for computing students whose programmes do not include the extensive sequence of mathematics modules taken by mathematics majors.

In a **graph neural network**, graph structure also describes the data being processed.[7] For example, we might represent devices as vertices carrying measurements and their connections as edges. A model can then combine a device's measurements with information from its neighbours, using graph structure alongside the numerical methods of neural networks.[8]

The same ideas prepare you for other BS IT modules, whether or not you choose to study AI.[9] In *Data Structures and Algorithms*, trees organise data, recursion describes procedures, and proofs help us check that they work for every permitted input. *Fundamentals of Database Systems* uses sets and relations to describe data, and logic to formulate query conditions. In *Networking 1*, graphs help us model connections and trace possible routes. *Information Assurance and Security 1* needs precise rules about identity and access; logic helps us express and check those conditions.

Modelling a network, specifying a database query and checking a recursive calculation are exercises in abstraction. Stating assumptions, tracking definitions and justifying each step also help us write correct, efficient code. Drawing a model, trying a case, finding a counterexample and revising an explanation are all part of that practice. In Chapter 1, we begin by identifying objects and stating claims about them precisely.

[6] See Goodfellow, Bengio and Courville, *Deep Learning* (2016), Chapter 1 and Part I: Applied Math and Machine Learning Basics.

[7] William L. Hamilton, Rex Ying and Jure Leskovec, *Inductive Representation Learning on Large Graphs* (2017), on learning from node features and neighbouring nodes.

[8] My undergraduate thesis on topology-informed machine learning involved counting blobs and holes in retinal images. Selecting pixels at or below a brightness cutoff and including their edges and vertices gives a cubical complex; selected pixels touching at a corner count as connected in this model. Persistent cubical homology tracks connected regions and holes as the cutoff changes, giving numerical summaries for machine learning. Much of the difficulty for me was combinatorial bookkeeping: tracking pixels, edges and vertices, and organising the associated sequences of vector spaces and maps. Graph theory helped me read category-theoretic diagrams, with vertices for vector spaces, arrows for maps and paths for compositions. This helped me understand the problem in a principled way, including the objects involved and the conditions a solution had to satisfy. I knew what needed to be solved; knowing exactly what it meant to solve it was another matter. For the diagrams, see Vidit Nanda, *Computational Algebraic Topology*, Chapter 6: Persistence, §6.1.

[9] The course titles here are drawn from CHED's model BS IT curriculum: CMO No. 25, series of 2015, pp. 17–18.

Contents

Introduction ii

1 Statements and quantifiers 1
- 1.1 Sets and notation . . . 2
- 1.2 Functions . . . 3
- 1.3 Expressions and statements . . . 5
- 1.4 Compound statements . . . 6
- 1.5 Predicates and domains . . . 9
- 1.6 Quantifiers . . . 10
- 1.7 Witnesses and counterexamples . . . 11
- 1.8 Quantifier order . . . 13
- 1.9 Exercises . . . 15

2 Propositional logic 21
- 2.1 Conditional rules . . . 22
- 2.2 Converse, inverse and contrapositive . . . 23
- 2.3 Logical equivalence . . . 25
- 2.4 Negation and rewriting . . . 26
- 2.5 Quantifier negation . . . 27
- 2.6 Valid arguments . . . 29
- 2.7 Invalid arguments . . . 32
- 2.8 Exercises . . . 34

3 Proof methods 40
- 3.1 Mathematical proof . . . 41
- 3.2 Parity and divisibility . . . 42
- 3.3 Direct proof . . . 43
- 3.4 Proof by contrapositive . . . 45
- 3.5 Proof by contradiction . . . 46
- 3.6 Proof construction and checking . . . 47
- 3.7 Exercises . . . 50

4 Sets and mappings 55
- 4.1 Subsets and set operations . . . 55
- 4.2 Set inclusion . . . 58
- 4.3 Set equality . . . 59
- 4.4 Injectivity . . . 60
- 4.5 Images of subsets . . . 61
- 4.6 Relations and transitivity . . . 62
- 4.7 Proof checking . . . 64
- 4.8 Exercises . . . 65

5 Induction **70**
5.1 Indexed claims . . . 71
5.2 Ordinary (weak) induction . . . 71
5.3 Sums, divisibility and inequalities . . . 73
5.4 Strong induction . . . 75
5.5 Recursive rules . . . 77
5.6 Choice of hypothesis . . . 78
5.7 Errors in induction . . . 79
5.8 Exercises . . . 80

6 Graphs **85**
6.1 Graph representations . . . 86
6.2 Vertex degrees . . . 87
6.3 Paths and connectedness . . . 89
6.4 Trees and unique paths . . . 90
6.5 Leaves and induction . . . 91
6.6 Spanning trees . . . 93
6.7 Constructing spanning trees . . . 94
6.8 Rooted trees . . . 95
6.9 Exercises . . . 97

7 Equivalence relations **102**
7.1 Relations . . . 103
7.2 Inverse relations and composition . . . 104
7.3 Reflexivity, symmetry and antisymmetry . . . 106
7.4 Transitivity . . . 107
7.5 Equivalence relations . . . 108
7.6 Equivalence classes . . . 109
7.7 Partitions . . . 111
7.8 Exercises . . . 113

8 Modular arithmetic **118**
8.1 Divisibility . . . 119
8.2 Quotients and remainders . . . 120
8.3 Congruence . . . 121
8.4 Residue classes . . . 123
8.5 Arithmetic with congruences . . . 124
8.6 Modular systems . . . 125
8.7 Exercises . . . 128

9 Counting principles **133**
9.1 Counting outcomes . . . 134
9.2 Sum rule . . . 135
9.3 Product rule . . . 136
9.4 Counting by cases . . . 138
9.5 Counting complements . . . 139
9.6 Two overlapping sets . . . 140
9.7 Three overlapping sets . . . 141

9.8 Exercises . . . 142

10 Permutations and combinations **148**
10.1 Factorials and arrangements . . . 149
10.2 Permutations . . . 150
10.3 Combinations . . . 151
10.4 Counting via subsets . . . 153
10.5 Combinatorial proofs . . . 154
10.6 Row sums by induction . . . 156
10.7 Binomial expansions . . . 156
10.8 Exercises . . . 158

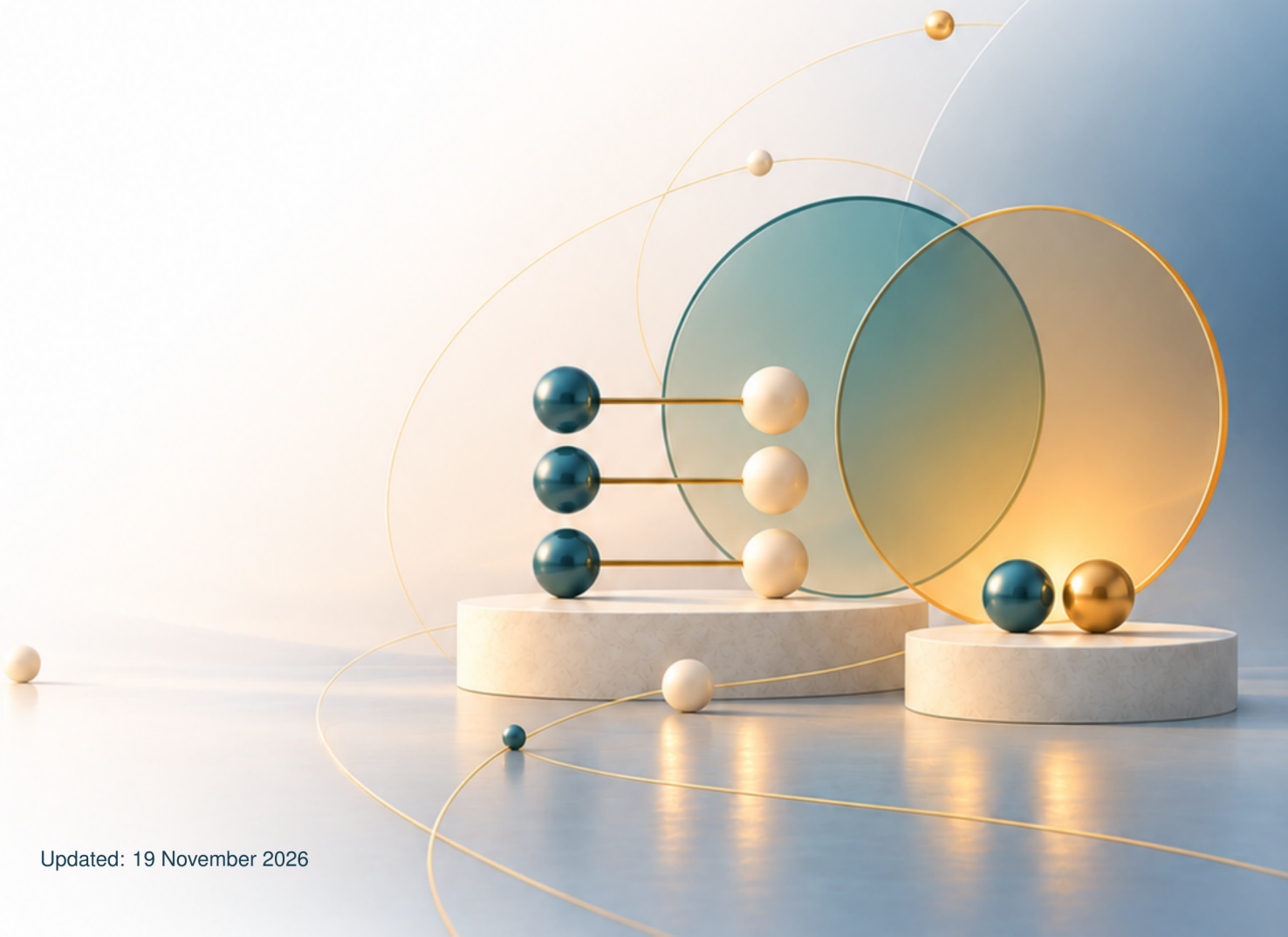
11 Sequences **163**
11.1 Sequences and indices . . . 164
11.2 Explicit and recursive descriptions . . . 165
11.3 Arithmetic sequences . . . 168
11.4 Geometric sequences . . . 169
11.5 Further recurrence relations . . . 170
11.6 Verification and comparison . . . 171
11.7 Exercises . . . 172

12 Recursion **177**
12.1 Injectivity and surjectivity . . . 178
12.2 Images and inverses . . . 179
12.3 Full binary trees . . . 181
12.4 Recursive counting functions . . . 183
12.5 Structural induction . . . 185
12.6 Exercises . . . 187

LECTURE NOTES
DISCRETE MATHEMATICS

CHAPTER 1

Statements and quantifiers



Updated: 19 November 2026

1 Statements and quantifiers

Common sense is a prized commodity, and it seems to be getting rarer by the day. You may reasonably ask what mathematical language adds when a claim can already be understood through common sense. Consider the following calculations:

$$0.9 < 1,$$
$$0.99 < 1,$$
$$0.999 < 1.$$

The same inequality holds after any finite number of nines. Now suppose the nines repeat without end. We write this as $0.\overline{99}$, where the bar means that the block 99 repeats forever. Would you expect this number to be smaller than 1, equal to 1, or greater than 1? Make a guess before reading on.

We interpret an infinite decimal as the real number approached by its finite decimal approximations; it has no final digit. Let

$$x = 0.\overline{99}.$$

Multiplying by 10 shifts the decimal point one place to the right. The digits after the decimal point are still all nines, so

$$10x = 9.\overline{99} = 9 + x.$$

Subtracting x from both sides gives

$$10x - x = 9.\overline{99} - 0.\overline{99} = 9.$$

Thus $9x = 9$, so $x = 1$. Therefore $0.\overline{99} = 1$.

The three finite inequalities remain correct. The mistake is the inference from "every finite approximation is less than 1" to "the infinite decimal is less than 1". The infinite decimal and 1 are two representations of the same real number.

A common language. Before we develop more advanced results, we need a common mathematical language in which to reason together. We must agree on what our objects, symbols and statements mean, so that we can follow one another's arguments and check whether a conclusion follows.

A rule such as "Every user can access a file" may sound clear until we try to apply it. Does it mean that each user has some accessible file, possibly a different one for each person? Or must there be one particular file that every user can access? These are different requirements. A system can satisfy the first while failing the second.

Mathematical language helps us make that difference explicit. This chapter introduces the objects we talk about, the statements we make about them and the notation used to say exactly what is being claimed. We begin with sets and functions because they give us a language for specifying the users, files, numbers and other objects under discussion.

What you should be able to do

After working through this chapter, you should be able to:

- read basic set and function notation, including domains and codomains;
- distinguish expressions, statements and open sentences;
- interpret the five logical connectives and construct short truth tables;
- translate claims using predicates, explicit domains and quantifiers;
- test claims using witnesses, counterexamples and complete finite checks;
- interpret the scope of quantifiers and explain why their order matters.

Using this self-study chapter

This chapter assumes familiarity with the topics covered in *Mathematics in the Modern World* (MMW). The discussion and exercises marked non-examinable are optional enrichment. All Python examples are non-examinable; no coding is required. The only numerical prerequisites are familiar number systems and elementary algebra; the mathematical language needed here is developed in the chapter.

1.1 Sets and notation

Before writing a rule about "every user", we must say which users count. One way to do this is to collect the relevant objects into a set.

❄ Definition 1.1 — Set and membership

A **set** is a well-defined collection of distinct objects, called its **elements**. The notation $x \in A$ means that x is an element of the set A; $x \notin A$ means that it is not.

"Well-defined" means that membership is determined by the description of the set. "The users registered in the practice system at noon today" can specify a set once the system and date are fixed.

By contrast, "the tall students in this class" and "the pretty people in this class" are **ill-defined descriptions** as they stand. What counts as tall or pretty? Two readers could disagree about whether Alice belongs because no precise criterion has been specified.

We could make the height example precise by saying "the students in this class whose recorded height is at least 170 cm", using a fixed height record. A recorded height of 170 cm meets the rule; 169 cm does not. A membership criterion need not be numerical: the fixed registration list above also determines who belongs.

Throughout several examples, we will use

$$U = \{\text{Alice}, \text{Bob}, \text{Charlie}\}, \qquad F = \{\text{guide}, \text{report}, \text{data}\}.$$

Here U is a set of users and F is a set of files. The braces delimit a set, while commas separate its listed elements. Thus Bob $\in U$ and report $\in F$. If David is not one of these three users, then David $\notin U$.

Set descriptions. We can describe a set in words, by listing its elements or by stating a condition that determines membership. Using $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ for the natural numbers, the set of natural numbers from 1 through 4 can be written as

$$D = \{1, 2, 3, 4\} = \{n \in \mathbb{N} : 1 \leq n \leq 4\}.$$

The second description is **set-builder notation**. Read it as "the set of natural numbers n such that $1 \leq n \leq 4$". The colon means "such that". It separates the kind of object being considered from the condition it must satisfy.

Set equality. A set records membership, so changing the order in which its elements are listed does not change the set. Repeating a listed element does not add a new one. For instance,

$$\{2, 5, 8\} = \{8, 2, 5\} = \{2, 2, 5, 8\}.$$

Two sets are equal when they contain exactly the same elements. The symbol $=$ expresses equality; $\in$ expresses membership. In particular, $2 \in \{2, 5, 8\}$ is true, but $\{2\} = \{2, 5, 8\}$ is false. The first says that one object belongs to a collection. The second says that two collections are identical, which they are not. The number 2 and the set $\{2\}$ containing that number are different objects in this notation; we must not substitute one for the other.

Example 1.1. Let $B = \{n \in \mathbb{N} : 2 < n \leq 6\}$. List its elements and decide whether $2 \in B$ and $6 \in B$.

The inequality $2 < n$ excludes 2, while $n \leq 6$ includes 6. Since n must be a natural number, the permitted values are $3, 4, 5, 6$. Therefore $B = \{3, 4, 5, 6\}$, $2 \notin B$ and $6 \in B$.

Number sets. We use $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ for the **natural numbers**. The **integers** include negative whole numbers as well:

$$\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}.$$

We write $\mathbb{R}$ for the **real numbers**, the numbers represented on the usual number line. In particular, negative integers belong to $\mathbb{Z}$ but not $\mathbb{N}$. Some books start the natural numbers at 1. In these notes, as in Levin, $0 \in \mathbb{N}$. When we need the positive integers, we will say so explicitly.

1.2 Functions

A set can identify the objects in a system. A function describes a precise assignment from one set of objects to another. For example, a practice system might assign each registered user one default printer.

❄ Definition 1.2 — Function, domain and codomain

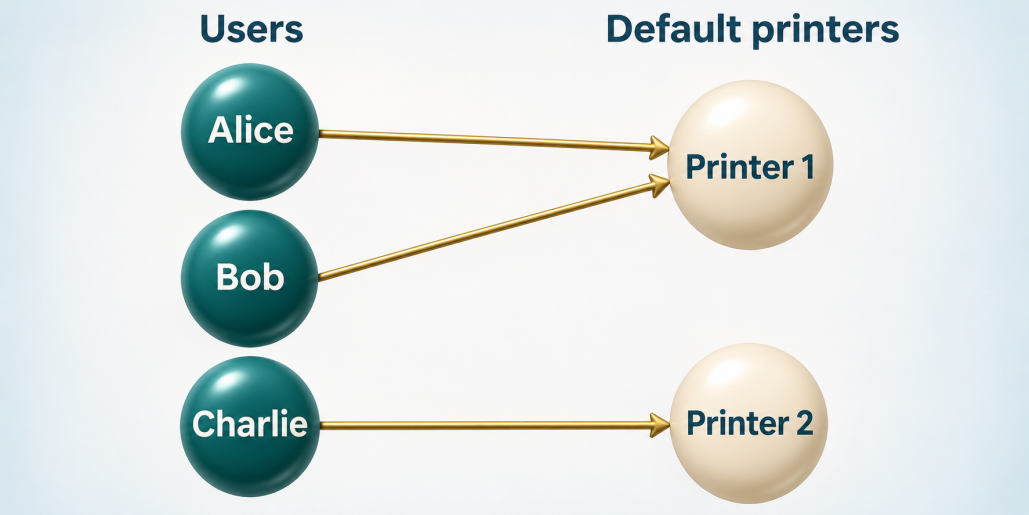
A **function** $f : A \to B$ assigns to **every** element of A **exactly one** element of B. The set A is its **domain**, and the set B is its **codomain**. The notation $f(x)$ denotes the output assigned to an input $x \in A$.

The expression $f : A \to B$ tells us the function's name and its permitted inputs and outputs. It does not, by itself, tell us which output belongs to each input. We also need a rule, a table or some other unambiguous description of the assignment.

Example 1.2. Let $L = \{\text{Printer 1}, \text{Printer 2}\}$. Define $d : U \to L$ by

User u	Alice	Bob	Charlie
Default printer $d(u)$	Printer 1	Printer 1	Printer 2

This is a function. Every user in U appears, and each is assigned exactly one printer in L. It is acceptable for Alice and Bob to have the same output: the definition restricts the number of outputs for each input, not the number of inputs that may share an output.



Each user has exactly one default printer; Alice and Bob may share the same printer.

If the assignment omitted Charlie, it would not define a function on all of U. If it gave Bob two different default printers, it would also fail. An output outside the stated codomain would fail the description $d : U \to L$ as well.

Example 1.3. Define $f : \mathbb{N} \to \mathbb{N}$ by $f(n) = 2n + 1$. To evaluate the function, substitute the input into its rule:

$$f(0) = 2(0) + 1 = 1, \qquad f(4) = 2(4) + 1 = 9.$$

Every natural-number input gives exactly one natural-number output, so the rule fits its stated domain and codomain. The output $f(4)$ is 9; the notation $f(4)$ does not mean that f is multiplied by 4.

The outputs actually attained form the function's **range**. Every element of this set belongs to the codomain; we say that the range is a **subset** of the codomain. For the last example, the range consists of the positive odd integers $1, 3, 5, \ldots$. We will examine further properties of functions later; for now, focus on the domain, the permitted outputs and the exactly-one-output requirement.

1.3 Expressions and statements

Compare $7 + 4$ with $7 + 4 = 12$. The first represents a number. The second makes a claim about that number. Although the claim is false, it is still the kind of sentence that can be judged true or false.

❄ Definition 1.3 — Expression and statement

A **mathematical expression** denotes an object or value. A **statement**, also called a **proposition**, makes a definite claim that is either true or false under the interpretation being used. Its truth or falsity is its **truth value**: TRUE or FALSE.

Thus $7 + 4$, $f(4)$ and the set U are expressions. The sentences "$7 + 4 = 11$" and "Charlie $\in U$" are true statements, while "$7 + 4 = 12$" and "$-2 \in \mathbb{N}$" are false statements. A false statement still makes a definite claim; its truth value is FALSE.

A question such as "Which file is accessible?" does not assert an answer. A command such as "Open the guide" tells someone what to do. Neither is a statement in our mathematical sense. Similarly, $5+$ is an incomplete expression: it has not supplied a second quantity to add.

Suppose b has been defined as the fixed byte count of a particular saved file. The sentence "$b > 1000$" has a definite truth value even if we have not inspected the file. Our lack of information does not prevent it from being a statement. The file and the time at which its size is recorded must be fixed; otherwise the intended claim may change.

Open sentences. Now consider $n + 4 = 12$ without assigning n a value. It is true for $n = 8$ and false for $n = 3$. We call this an **open sentence**: its truth value depends on an unspecified variable. Substituting a value can turn it into a statement. So can specifying how the claim is to range over its possible inputs, which is the purpose of quantifiers in Section 1.6.

Example 1.4. Suppose n ranges over the integers but has not been assigned a value. First identify what each item does; then, if it makes a complete claim, decide whether that claim is true or false where the information allows.

Item	Classification and reason
$n + 4$	Expression: it denotes a value once n is specified.
$n + 4 = 12$	Open sentence if n has not been assigned or quantified.
$8 + 4 = 12$	True statement: substitution has fixed the input.
$8 + 4 = 13$	False statement: it makes a complete claim, but $8 + 4$ equals 12.
"Is $n + 4 = 12$?"	Question: it asks for a truth value rather than asserting one.

Now fix $n = 8$. In this context, $n + 4 = 12$ is a true statement even though it is written with a letter. Without that assignment, the same condition is open. Quantifiers can also turn the condition into a complete claim, as we will see in Section 1.6. We must read the complete claim and its context.

1.4 Compound statements

Statements can be combined to make other statements. Suppose P means "Alice is registered in the practice system" and Q means "Alice has a current access permission". Then P and Q are **propositional variables**: letters standing for statements. They are not numbers to be added or multiplied.

An **atomic** statement is treated as a single claim in the logical analysis at hand. A **compound** (or **molecular**) statement is built from statements using logical connectives. A connective specifies how the component truth values determine the truth value of the result.

❄ Definition 1.4 — The five logical connectives

Notation	Name	Reading
$\neg P$	Negation	Not P
$P \wedge Q$	Conjunction	P and Q
$P \vee Q$	Disjunction	P or Q, allowing both
$P \to Q$	Implication	If P, then Q
$P \leftrightarrow Q$	Biconditional	P if and only if Q

The negation $\neg P$ reverses the truth value of P. A conjunction is true only when both components are true. A disjunction is true when at least one component is true, including the case where both are true. This is the **inclusive** meaning of "or" used in these notes. If a rule intends "one or the other but not both", it must say so.

An implication $P \to Q$ is false exactly when P is true and Q is false. Here P is the **antecedent** and Q is the **consequent**. An implication says that the case "P TRUE but Q FALSE" does not occur. It does not, by its notation alone, say that P causes Q.

A biconditional $P \leftrightarrow Q$ is true when P and Q have the same truth value. It requires both directions: P holds if Q holds, and Q holds if P holds. It is therefore false when exactly one component is true.

Example 1.5. Negating an inequality. Fix a saved file and a recorded time, and let b be the file's size in bytes at that time. Let P be the statement $b > 1000$, meaning "The file exceeds 1000 bytes." Its negation $\neg P$ says that the recorded size is *at most* 1000 bytes, written $b \leq 1000$.

Suppose the recorded size is exactly 1000 bytes. Then P is false because $1000 > 1000$ is false. Its negation is true because $1000 \leq 1000$ is true. The equality case belongs in the negation: a file of exactly 1000 bytes does not exceed 1000 bytes.

The condition $b < 1000$ would also be false at this size, so it cannot be the negation of $b > 1000$. Negation must reverse the truth value in every case, including the boundary.

Example 1.6. Conjunction and disjunction. Let P mean "David has a laptop" and Q mean "David has a tablet", referring to his devices at one fixed time.

Suppose David has both devices. Then P and Q are both true, so $P \wedge Q$ is true: both required claims hold. The disjunction $P \vee Q$ is also true, because at least one claim holds; having both devices satisfies that requirement too.

Now consider a situation in which David has a laptop but no tablet. Then P is true and Q is false. The conjunction $P \wedge Q$ is false because one required claim fails, while $P \vee Q$ remains true because the laptop claim holds. If he has neither device, both compound statements are false.

Truth tables. A **truth table** lists the truth value of a compound statement for every possible combination of truth values of its components. Write T for TRUE and F for FALSE. Two components have four combinations:

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \to Q$	$P \leftrightarrow Q$
T	T	F	T	T	T	T
T	F	F	F	T	F	F
F	T	T	F	T	T	F
F	F	T	F	F	T	T

Python (non-examinable)

We can generate the same four rows in Python. The values `True` and `False` are Boolean truth values. The square brackets form a list of choices. A `for` loop repeats its indented lines for each choice; the two loops below consider every pair of truth values.

```
for p in [True, False]:
    for q in [True, False]:
        print(p, q, not p, p and q, p or q,
              (not p) or q, p == q)
```

Output.

```
True True False True True True True
True False False False True False False
False True True False True True False
False False True False False True True
```

The seven columns are P, Q, $\neg P$, $P \wedge Q$, $P \vee Q$, $P \to Q$ and $P \leftrightarrow Q$, in that order. Here `not`, `and` and `or` act on Boolean values. The expression `(not p) or q` represents $P \to Q$, and `p == q` compares their truth values for the biconditional. Python uses `==` for comparison; a single `=` assigns a value to a name. Each comma in `print` separates one output column.

Example 1.7. Let P mean "Alice is registered" and Q mean "Alice has permission". The claim $P \to Q$ says, "If Alice is registered, then Alice has permission."

If Alice is registered but lacks permission, the implication is false. If Alice is registered and has permission, it is true. If Alice is not registered, neither truth value of Q produces the forbidden combination, so the implication is true in either case. This last conclusion does not tell us whether Alice has permission: it says that Alice's case does not violate the particular implication being checked.

To assert a system-wide rule, we would need a claim about all relevant users, not just Alice. Quantifiers will let us express that later in the chapter.

Example 1.8. Consider one completed integrity check and its status indicator at a fixed recorded time. Let P mean "The indicator is green" and Q mean "The check passed". We want to evaluate the claim "The indicator is green if and only if the check passed", written $P \leftrightarrow Q$.

If the check passed and the indicator is green, both components are true, so the biconditional is true. If the check did not pass and the indicator is not green, both components are false, and the biconditional is still true: the two truth values match. This second case does not assert that the check passed or that the indicator is green.

If the check passed but the indicator is not green, P is false and Q is true, so the biconditional is false. It is also false if the indicator is green although the check did not pass. In each of these cases, the truth values disagree.

Parentheses and scope. Parentheses indicate which statements a connective combines. For example, $\neg(P \vee Q)$ negates the whole disjunction, while $(\neg P) \vee Q$ negates P alone. The **scope** of a connective is the part of the expression it acts on. We use parentheses whenever grouping could be unclear.

To see the effect, take P FALSE and Q TRUE. For $\neg(P \vee Q)$, first evaluate the expression inside parentheses. The disjunction $P \vee Q$ is true because Q is true. Negating that result makes $\neg(P \vee Q)$ false.

For $(\neg P) \vee Q$, first negate P. Since P is false, $\neg P$ is true. The outer disjunction then has two true components, $\neg P$ and Q, so $(\neg P) \vee Q$ is true. The same values of P and Q give different results because the negation acts on different parts.

Example 1.9. Fix one report and a recorded time. Let P mean "Bob uploaded the report" and Q mean "Judy approved the report", referring to what has happened by that time. Translate and evaluate the sentence "Bob uploaded the report, but Judy has not approved it."

The first part asserts P. The second part denies Q, so it is $\neg Q$. Here "but" joins the two claims with the logical meaning of "and", while suggesting a contrast in ordinary English. The compound statement is therefore $P \wedge \neg Q$.

Suppose the recorded information shows that Bob's upload is complete and Judy's approval is still pending. Then P is true and Q is false. First negate Q: $\neg Q$ is true. Both components of the conjunction $P \wedge \neg Q$ are now true, so the whole statement is true.

Example 1.10. Construct the truth table for $(P \vee Q) \wedge \neg Q$.

P	Q	$P \vee Q$	$\neg Q$	$(P \vee Q) \wedge \neg Q$
T	T	T	F	F
T	F	T	T	T
F	T	T	F	F
F	F	F	T	F

The intermediate columns make the reasoning checkable. For example, in the third row, $P \vee Q$ is true because Q is true, but $\neg Q$ is false. Their conjunction is therefore false. Only the second row makes both components of the final conjunction true.

For a further grouping check, take P TRUE and Q, R FALSE. Then $P \vee (Q \wedge R)$ is true, while $(P \vee Q) \wedge R$ is false. The same component truth values have produced different results because the grouping changed.

Detailed implication analysis and logical equivalence belong to Chapter 2; here we are learning to read and evaluate the expressions.

1.5 Predicates and domains

An open sentence can be treated as a condition applied to an input. This lets us use one concise description for many related statements.

❄ Definition 1.5 — Predicate and domain of discourse

A **predicate** is a condition involving one or more variables that becomes a statement when suitable values are assigned to all its unspecified variables. Its **domain of discourse** specifies the objects that a variable is allowed to represent.

For example, let the domain be $\mathbb{Z}$ and define $E(n)$ to mean "n is even". An integer is **even** when it equals $2k$ for some integer k. Thus $E(6)$ is true because $6 = 2(3)$, and $E(5)$ is false because no integer k satisfies $5 = 2k$. Negative integers and zero can be even too: $E(-4)$ and $E(0)$ are true.

Predicate notation is a convenient name for a condition. Writing $E(n)$ does not by itself assert that every n satisfies it. Substitution supplies a particular input: $E(6)$ is one statement, and $E(5)$ is another.

Example 1.11. Functions and predicates. Let $f : \mathbb{N} \to \mathbb{N}$ have rule $f(n) = 2n + 1$, and let $H(n)$ mean "$n > 4$" for $n \in \mathbb{N}$. Then

$$f(3) = 7, \qquad H(3) \text{ is FALSE}, \qquad H(7) \text{ is TRUE}.$$

The numerical function f produces a number; the predicate H tests a condition and produces a truth value once its input is supplied. The similar notation should not lead us to confuse 7 with TRUE. Functions need not always have numerical outputs: in particular, a predicate can itself be viewed as a function with truth values as outputs.

Consider the condition $n < 0$. It is satisfied by some integers, such as -2, but by no natural number under our convention. A claim about its possible solutions therefore depends on whether n is allowed to range over $\mathbb{Z}$ or only $\mathbb{N}$. We must specify that domain before deciding what the claim means.

Likewise, solving $2x = 1$ gives $x = \frac{1}{2}$. This is an allowed input if the domain is $\mathbb{R}$, but not if it is $\mathbb{Z}$, since $\frac{1}{2}$ is not an integer. Changing the permitted inputs changes what we can conclude, even though the written equation is unchanged.

Several variables. Let $A(u, f)$ mean "user u is permitted to access file f", where $u \in U$ and $f \in F$. Fix the permissions as follows for the remainder of the worked examples. A T entry means permission is granted, and an F entry means it is not.

	guide	report	data
Alice	T	T	F
Bob	F	T	T
Charlie	T	F	T

For instance, $A(\text{Bob}, \text{report})$ is true and $A(\text{Bob}, \text{guide})$ is false. Both inputs matter. The expression $A(u, \text{report})$ still depends on the user: it is true for Alice and Bob but false for Charlie. Supplying one input has not fixed the other.

1.6 Quantifiers

Often we want a claim about a whole domain, not a list of unrelated substitutions. A **quantifier** specifies how a predicate is being asserted across that domain.

❄ Definition 1.6 — Universal and existential quantification

Let $P(x)$ be a predicate whose inputs belong to a specified domain D.

- $\forall x \in D\, P(x)$ means that $P(x)$ is true for **every** element of D. This is a **universal** claim.
- $\exists x \in D\, P(x)$ means that $P(x)$ is true for **at least one** element of D. This is an **existential** claim.

Read $\forall$ as "for every" or "for all", and $\exists$ as "there exists". If there are two or ten successful inputs, an existential claim is still true: it does not require exactly one. In the quantified examples below, the domains are nonempty. When several domains are in use, each quantifier will specify its own domain.

Example 1.12. Take $D = \{0, 1, 2, 3\}$ and let $P(n)$ mean "$n < 3$". The claim $\exists n \in D\, P(n)$ is true: $n = 0$ supplies one successful input. The claim $\forall n \in D\, P(n)$ is false, since $P(3)$ is false. The same predicate has produced different claims because the quantifier changed.

Domain restrictions. "Every user is registered" can be written as $\forall u \in U\, R(u)$ when U is already the set of users and $R(u)$ means "u is registered". If our domain instead includes objects of other kinds, we must identify which objects the claim is about.

Suppose D is a set of records, $S(x)$ means "record x is a student record", and $C(x)$ means "record x is complete". Then

$$\forall x \in D \left(S(x) \to C(x)\right)$$

says that every student record is complete. A record that is not a student record does not violate this rule. Writing $\forall x \in D\, (S(x) \wedge C(x))$ would make the stronger claim that every record is both a student record and complete.

By contrast,

$$\exists x \in D \left(S(x) \wedge C(x)\right)$$

says that some student record is complete. We need one record satisfying *both* conditions. An implication inside this existential claim would not express the same requirement: a non-student record could make $S(x) \to C(x)$ true even if no student record were complete.

For example, suppose $r \in D$ is an incomplete staff record and is not a student record. Both $S(r)$ and $C(r)$ are FALSE, so $S(r) \to C(r)$ is TRUE, while $S(r) \wedge C(r)$ is FALSE. This record does not violate the universal rule and does not provide the complete student record required by the existential claim.

Quantifier scope. The **scope** of a quantifier is the part of the expression to which it applies. Parentheses make that scope visible. In $\forall x \in D\,(S(x) \to C(x))$, the quantifier applies to the whole implication $S(x) \to C(x)$ inside the parentheses.

(Non-examinable) Free and bound variables

This optional discussion introduces formal terminology for describing variable occurrences. It will not be assessed in this course. Interpreting quantifiers, their scope and their order remains core material.

In $\forall x \in D\,P(x)$, the quantifier tells us how x is used. We call the occurrences of x that it controls **bound**; a variable occurrence not controlled by a quantifier is **free**.

Compare the following expressions, with $x, y \in \mathbb{Z}$:

Expression	What remains unspecified?
$x + y = 5$	Both x and y are free.
$\exists y \in \mathbb{Z}\,(x + y = 5)$	y is bound; x is still free.
$\forall x \in \mathbb{Z}\,\exists y \in \mathbb{Z}\,(x + y = 5)$	Both variables are bound, so this is a statement.

We have not removed the free variable from the second expression merely because the condition happens to have a solution for every integer x. As written, x is still free; its role in the claim must be specified. The third expression supplies that specification and is true: after an integer x is given, $y = 5 - x$ is an integer that works.

The words "free" and "bound" describe occurrences within an expression, not permanent properties of a letter. For example, in

$$\big(\forall x \in D\,P(x)\big) \wedge Q(x),$$

the x in $P(x)$ is bound, but the x in $Q(x)$ is free: it is outside the quantifier's parentheses. We can make the separation easier to see by renaming the bound variable to a fresh letter z:

$$\big(\forall z \in D\,P(z)\big) \wedge Q(x).$$

The quantifier and the variable it controls in P now use z; the free x in $Q(x)$ is unchanged. Renaming this bound variable does not change the claim.

Implicit universal quantification. Mathematical writing sometimes leaves a universal quantifier unstated. For example, "For real x, $x + 1 > x$" is normally read as $\forall x \in \mathbb{R}\,(x + 1 > x)$. This is a writing convention, supported by the context. It does not mean that a bare open sentence has automatically become a statement without any interpretation. In your own translations, write the quantifiers and domains explicitly until the intended claim is unambiguous.

1.7 Witnesses and counterexamples

A quantifier tells us what kind of evidence would establish or refute the claim. Checking a convenient input is useful only if we understand what that check allows us to conclude.

❄ Definition 1.7 — Witness and counterexample

A **witness** for $\exists x \in D\, P(x)$ is an element $a \in D$ for which $P(a)$ is true. A **counterexample** to $\forall x \in D\, P(x)$ is an element $b \in D$ for which $P(b)$ is false.

Example 1.13. The claim $\exists n \in \mathbb{Z}\,(n + 3 = 1)$ is true. The integer -2 is a witness because $-2 + 3 = 1$. However, $\exists n \in \mathbb{N}\,(n + 3 = 1)$ is false: the equation forces $n = -2$, and $-2 \notin \mathbb{N}$. A value satisfying the equation outside the permitted domain does not witness the stated claim.

Example 1.14. Consider $\forall n \in \mathbb{N}\,(n^2 > n)$. It holds for $n = 2$ and $n = 3$, but it fails for $n = 0$, since $0^2 = 0$ is not greater than 0. It also fails for $n = 1$. Either failed case alone disproves the universal claim; we do not have to find every failed case.

For a finite domain such as $D = \{1, 2, 3\}$, we can establish $\forall n \in D\,(n^2 \geq n)$ by checking all three inputs. They give $1 \geq 1$, $4 \geq 2$ and $9 \geq 3$. There are no unexamined elements of D left.

Python (non-examinable)

For the complete domain $D = \{1, 2, 3\}$, we can record the truth value of $n^2 \geq n$ for each input. The assignment `domain = [1, 2, 3]` stores these three inputs in a list. The empty list `[]` initially contains no results; `append` adds one result on each pass through the indented loop. Python writes multiplication as `*` and "greater than or equal to" as `>=`.

```
domain = [1, 2, 3]
checks = []
for n in domain:
    checks.append(n * n >= n)
print(checks)
print(all(checks))
print(any(checks))
```

Output.

```
[True, True, True]
True
True
```

The function `all` returns `True` when every recorded condition is true; `any` returns `True` when at least one is true. Thus the last two lines check the universal and existential claims on this stated finite domain. Every input in D has been checked.

For the infinite domain $\mathbb{N}$, those three checks are not a complete verification. Even a computer calculation checking a million inputs leaves infinitely many others unexamined. Examples may suggest a general claim, but establishing an infinite universal claim requires reasoning that covers an arbitrary permitted input. We begin the systematic study of such proofs in later chapters.

What you found	What this establishes
One successful permitted input	An existential claim is true. This fact alone does not generally establish a universal claim.
One failed permitted input	A universal claim is false. This fact alone does not generally refute an existential claim.
Every input succeeds in a complete finite check	The universal claim is true.
Every input fails in a complete finite check	The existential claim is false.

If the domain has only one element, checking that element is already a complete check. In the last two rows, "every input" means every element of the stated domain, not every element we happened to test. To explain why a claim is false, identify the relevant failed case or explain why no permitted witness can work. Formal laws for negating quantified claims follow in Chapter 2.

1.8 Quantifier order

When a predicate has more than one input, a claim may need more than one quantifier. Read the quantifiers from left to right. Their order determines which choices may depend on earlier choices.

Return to the permission table for $A(u, f)$ in Section 1.5.

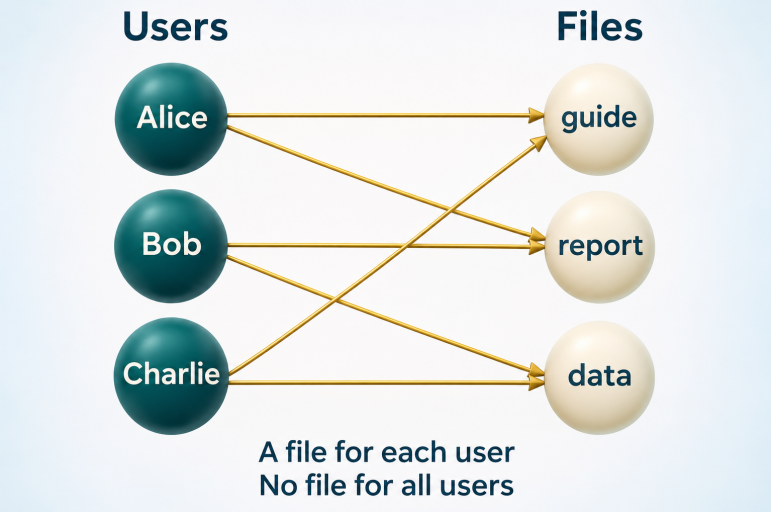
$$\forall u \in U \, \exists f \in F \, A(u, f)$$

says: "For each user, there is at least one file that the user can access." Choose a user first, then find a suitable file. For Alice we may choose guide; for Bob we may choose report; for Charlie we may choose data. We have considered every user in this finite domain, so the claim is true. There is no requirement to choose the same file for all three users.

Now reverse the quantifiers:

$$\exists f \in F \, \forall u \in U \, A(u, f).$$

This says: "There is a file that every user can access." We must choose one file that works for all users. Guide fails for Bob, report fails for Charlie, and data fails for Alice. Since those are all the files in F, no file works, and the claim is false.



Every user has access to a file, but no file can be accessed by all three users.

✍ Quantifier order

The statement $\forall u \exists f\, A(u, f)$ allows the file choice to depend on the user. The statement $\exists f \forall u\, A(u, f)$ requires one file chosen before the user is considered. Exchanging these quantifiers changes the requirement; the permission table shows that one claim can be true while the other is false.

Example 1.15. The statement $\forall x \in \mathbb{Z}\, \exists y \in \mathbb{Z}\, (y > x)$ is true. Once an integer x has been given, choose $y = x + 1$. This is an integer larger than x.

The statement $\exists y \in \mathbb{Z}\, \forall x \in \mathbb{Z}\, (y > x)$ is false. Suppose someone proposes a fixed integer y. The universal claim must include the permitted input $x = y$, but then it would require $y > y$, which is false. Therefore no proposed integer can be the required witness. The first claim offers a choice depending on x; the second asks for a single choice that works for all x.

The expression $\exists f \in F\, A(u, f)$ quantifies over files but does not specify which user u we mean or how the claim ranges over users. It describes a property of a user rather than stating how that property holds across U. Substituting a specific user, as in $\exists f \in F\, A(\text{Alice}, f)$, gives a statement. Alternatively, adding $\forall u \in U$ gives the first system-wide claim above.

The four common combinations have different readings:

Claim	Reading
$\forall u \in U\, \forall f \in F\, A(u, f)$	Every user can access every file.
$\forall u \in U\, \exists f \in F\, A(u, f)$	Each user can access at least one file; the file may depend on the user.
$\exists u \in U\, \forall f \in F\, A(u, f)$	At least one user can access every file.
$\exists u \in U\, \exists f \in F\, A(u, f)$	At least one user–file pair has access permission.

In each of these four claims, both the user and the file are quantified. If the domains and predicate are fixed, each complete claim has a truth value. The exercises ask you to make that judgment with evidence that matches the quantifiers.

1.9 Exercises

1.1. Let $B = \{2, 4, 6\}$. Decide whether each statement is true or false. Explain the distinction between an element and a set of elements where relevant.

a) $4 \in B$.

b) $5 \notin B$.

c) $\{4\} \in B$.

d) $B = \{6, 4, 2, 4\}$.

1.2.

a) List $\{n \in \mathbb{N} : n < 11 \text{ and } n \text{ is a multiple of } 3\}$. Here a multiple of 3 means a number $3k$ for an integer k.

b) Describe $\{0, 2, 4, 6, 8\}$ precisely in words.

c) Write $\{1, 2, 3, 4\}$ in set-builder notation, using $\mathbb{N}$ and inequalities.

1.3. Let $D = \{0, 1, 2, 3\}$ and $C = \{0, 1, 4, 9, 16\}$. Define $f : D \to C$ by $f(n) = n^2$.

a) State the domain and codomain.

b) Find $f(0)$ and $f(3)$.

c) List the outputs actually obtained from inputs in D.

d) No input in D has output 16. Does this prevent f from being a function $D \to C$? Explain using the definition.

1.4. In each case, the proposed domain is $\{a, b, c\}$ and the proposed codomain is $\{0, 1\}$. Decide whether the displayed assignments define a function with that domain and codomain. If not, identify the precise failure. The notation $a \mapsto 0$ means that input a is assigned output 0.

a) $a \mapsto 0$, $b \mapsto 1$, $c \mapsto 0$.

b) $a \mapsto 0$, $b \mapsto 1$, with no assignment for c.

c) $a \mapsto 0$, $a \mapsto 1$, $b \mapsto 0$, $c \mapsto 1$.

d) $a \mapsto 0$, $b \mapsto 1$, $c \mapsto 2$.

1.5. Classify each item as an expression denoting a value, a complete statement, an open sentence, a question or a command. For complete statements, give the truth value where possible. In part (c), m ranges over $\mathbb{Z}$ but has not been assigned a value.

a) $7 + 5$.

b) $7 + 5 = 13$.

c) $m + 2 = 6$.

d) Is 3 smaller than 5?

e) Delete the old file.

f) $3^2 \geq 8$.

1.6. Fix one campus computer, identified as Lab-PC-07. Consider the sentence: “The most recent successful backup of Lab-PC-07 before 11:00 a.m. on 21 November 2026 finished on 20 November 2026.” Assume such a backup exists.

a) Is this a statement even if you cannot inspect the backup records? Explain.

b) A classmate says that $2^3 = 6$ cannot be a statement because it is false. Identify the mistake.

1.7. Let P mean "The backup completed" and Q mean "The integrity check passed". Translate each formula into a clear English sentence, retaining its meaning.

a) $\neg Q$.

b) $P \wedge Q$.

c) $P \vee Q$.

d) $P \to Q$.

e) $P \leftrightarrow Q$.

1.8. Let P mean "A valid local copy exists" and Q mean "A valid remote copy exists".

a) If both copies exist, is $P \vee Q$ true or false?

b) A message says "Exactly one of the two copies exists". Is that message true in the same situation? Explain why it differs from $P \vee Q$.

1.9. Suppose P is FALSE and Q is TRUE. Find the truth value of each formula, using the truth condition for its connective.

a) $\neg P$.

b) $P \wedge Q$.

c) $P \vee Q$.

d) $P \to Q$.

e) $P \leftrightarrow Q$.

1.10. Construct the truth table for $(P \wedge Q) \vee \neg P$. Use the row order $(P, Q) = (\text{T}, \text{T}), (\text{T}, \text{F}), (\text{F}, \text{T}), (\text{F}, \text{F})$, and show intermediate columns for $P \wedge Q$ and $\neg P$ before the final column.

1.11. Let P and Q be TRUE and R FALSE.

a) Evaluate $(P \vee Q) \wedge R$, showing the value inside parentheses.

b) Evaluate $P \vee (Q \wedge R)$, again showing the inner value.

c) Can the parentheses in these two expressions be ignored? Use your answers.

1.12. Let P mean "The file has been checked", Q mean "The file has been cleared", and R mean "The file is released". Write each sentence symbolically, using parentheses where needed.

a) The file has been checked and has been cleared.

b) The file has been checked or has been cleared, or both.

c) If the file has been checked, then it is released.

d) The file is released exactly when it has been checked and cleared.

1.13. For $n \in \mathbb{Z}$, let $f(n) = n^2$ and let $P(n)$ be the predicate $n^2 \leq 10$.

a) Find $f(-3)$ and state the truth value of $P(-3)$.

b) Find $f(-4)$ and state the truth value of $P(-4)$.

c) Explain why the numerical function f and the predicate P are different, even though both definitions contain n^2.

1.14. Let $B(x, y)$ mean $x + 2 = y$, with $x, y \in \mathbb{Z}$.

a) Give the truth values of $B(3, 5)$ and $B(5, 3)$.

b) Write out $B(x, 6)$. Which variable remains unspecified?

c) Write out $B(4, y)$. Which variable remains unspecified?

d) Is $B(4, 6)$ a complete statement? If so, give its truth value.

1.15. Let $D_1 = \{0, 1, 2\}$ and $D_2 = \{-2, -1, 0, 1, 2\}$. Evaluate each claim once with $D = D_1$ and once with $D = D_2$. Give a witness or counterexample where one settles the claim.

a) $\forall x \in D\,(x \geq 0)$.

b) $\exists x \in D\,(x^2 = 4)$.

c) $\exists x \in D\,(x + 2 = 0)$.

1.16. For parts (a)–(d), write each claim with an explicit quantifier and domain, then state whether it is true. Supply a witness for each true existential claim.

a) Every integer has a non-negative square.

b) There is a negative integer.

c) Some integer has square 9.

d) In a discussion where n ranges over $\mathbb{N}$, an author writes "$n + 1 > n$" as an assertion about every allowed n. Make the implicit quantifier explicit.

For parts (e) and (f), let D be a fixed set of users, some registered and some unregistered. Let $R(u)$ mean "u is registered" and $G(u)$ mean "u has permission to access the guide". Translate each claim using D, R and G, and explain your choice of connective.

e) Every registered user has permission to access the guide.

f) At least one registered user has permission to access the guide.

1.17. *Optional; non-examinable.* All variables below range over $\mathbb{Z}$. Identify the free variables, if any, in each formula, and state which variable is bound by each quantifier. Is each entire formula a complete statement?

a) $\forall x \in \mathbb{Z}\,(x + y = 0)$.

b) $\big(\forall x \in \mathbb{Z}\,(x^2 \geq 0)\big) \wedge (y > 2)$.

c) $\exists y \in \mathbb{Z}\,(x = y + 1)$.

d) $\forall x \in \mathbb{Z}\, \exists y \in \mathbb{Z}\,(x = y + 1)$.

1.18. Let $D = \{-2, -1, 0, 1, 2\}$. Make a two-row table listing x and x^2, then decide each claim. For a true existential claim give a witness, and for a false universal claim give a counterexample. Explain what justifies the other two kinds of conclusion when the domain is finite.

a) $\exists x \in D\,(x^2 = 1)$.

b) $\forall x \in D\,(x^2 \leq 2)$.

c) $\forall x \in D\,(x^2 \geq 0)$.

d) $\exists x \in D\,(x^2 < 0)$.

1.19. Assess each conclusion. Explain what the stated checks do or do not establish.

a) Eve checks $n = 0, 1, 2$ in $\forall n \in \mathbb{N}\,(n^2 \geq n)$. All three checks succeed, so Eve says the universal claim has been proved.

b) Faith checks $n = 0, 1, 2$ in $\exists n \in \mathbb{N}\,(n^2 = 16)$. All three checks fail, so Faith says the existential claim is false.

c) Grace checks $n = 4$ in the same existential claim. Since $4^2 = 16$, Grace says the existential claim is true.

d) A classmate finds $n = 10$ when checking $\forall n \in \mathbb{N}\,(n < 10)$ and concludes that the universal claim is false.

1.20. Decide each claim. Give a witness for a true existential claim, a counterexample for a false universal claim, or a brief explanation otherwise. Listing a few failed candidates cannot rule out an existential claim over an infinite domain.

a) $\forall n \in \mathbb{N}\,(n < 100)$.

b) $\exists n \in \mathbb{N}\,(n + 7 = 3)$.

c) $\exists n \in \mathbb{Z}\,(n + 7 = 3)$.

d) $\forall n \in \mathbb{Z}\,(n^2 \geq 0)$.

1.21. Let $E(x, y)$ mean $y = x + 2$, with $x, y \in \mathbb{Z}$. Read each claim in words and decide whether it is true. For a true claim explain how to choose y; for a false claim explain why no fixed choice of y works.

a) $\forall x \in \mathbb{Z}\,\exists y \in \mathbb{Z}\,E(x, y)$.

b) $\exists y \in \mathbb{Z}\,\forall x \in \mathbb{Z}\,E(x, y)$.

1.22. Let $D = \{0, 1, 2\}$ and let $H(x, y)$ mean $y \geq x$, with $x, y \in D$.

a) Is $\forall x \in D\,\exists y \in D\,H(x, y)$ true? Give an allowed choice of y for each x.

b) Is $\exists y \in D\,\forall x \in D\,H(x, y)$ true? If so, give one fixed witness y.

c) Do these answers mean that reversing quantifiers always preserves truth? Explain using Exercise 1.21.

1.23. Use the following *new* data for this exercise and Exercise 1.24. Let $U = \{\text{Alice}, \text{Bob}, \text{Charlie}\}$ and $F = \{\text{guide}, \text{report}, \text{ledger}, \text{notes}\}$. Let $A(u, f)$ mean "user u is permitted to access file f". The table specifies all the permissions: 1 means permitted and 0 means not permitted.

	guide	report	ledger	notes
Alice	1	1	0	0
Bob	0	1	1	0
Charlie	1	0	1	0

a) Evaluate $A(\text{Alice}, \text{ledger})$, $A(\text{Bob}, \text{report})$ and $A(\text{Charlie}, \text{notes})$.

b) Evaluate $\forall u \in U\,\exists f \in F\,A(u, f)$. If true, give a suitable file for each user.

c) Evaluate $\exists f \in F\,\forall u \in U\,A(u, f)$. Explain your answer for every candidate file.

d) Evaluate $\forall f \in F\,\exists u \in U\,A(u, f)$. If false, identify a counterexample file.

e) Evaluate $\exists u \in U\,\forall f \in F\,A(u, f)$. If false, explain why every user fails.

f) *Optional; non-examinable.* In $\exists u \in U\,A(u, f)$, identify the free variable. State which allowed values make the formula true and which make it false.

g) *Optional; non-examinable.* In $\forall f \in F\,A(u, f)$, identify the free variable. Evaluate the resulting statement for each allowed value of that variable.

h) Translate "There is a file that no user is permitted to access" into symbols and decide whether it is true.

1.24. Keep the domains and predicate from Exercise 1.23. Decide whether each proposed formula faithfully expresses its English claim, and correct it if necessary. Judge the meaning: an incorrect translation may accidentally have the same truth value as the intended claim.

a) "Every user can access at least one file." Proposed formula: $\exists f \in F\, \forall u \in U\, A(u, f)$.

b) "Alice can access every file." Proposed formula: $\forall u \in U\, A(u, f)$.

c) "At least one user can access report." Proposed formula: $\forall u \in U\, A(u, \text{report})$.

d) "For every file there is a user who can access it." Proposed formula: $\forall f \in F\, \exists u \in U\, A(u, f)$.

LECTURE NOTES
DISCRETE MATHEMATICS

CHAPTER 2

Propositional logic



Updated: 19 November 2026

2 Propositional logic

In high-school algebra, a letter such as x could stand for a number. The identity $x + x = 2x$ holds for every real number x: the letter lets us state a rule once and use it for different numbers.

Here we use letters such as P and Q as placeholders for whole *statements*: ideas expressed as claims that are true or false. Using letters makes it easier to see which claims are being used and whether a conclusion follows from them.

For example, fix one data link between two devices. Let P mean "This link uses Wi-Fi" and Q mean "This link uses no data cable between the devices". The rule "If this link uses Wi-Fi, it uses no data cable" has the form $P \to Q$. If we also know P, we can conclude Q: this link uses no data cable.

The same deduction pattern can be used with statements about laboratory access or numbers. It guarantees a true conclusion whenever its starting claims are true. We must keep each letter's meaning consistent within an argument, but we can use the same reasoning rule with different statements.

Chapter 1 gave us the ingredients of this language: statements, connectives, predicates and quantifiers, together with sets and functions. Chapter 2 develops the recipes for working with them: rewriting conditions, forming negations and drawing conclusions. We will use these tools when writing proofs in Chapter 3.

This may feel abstract or even bizarre at first. With practice, familiar patterns become easier to recognise, and structuring your reasoning this way becomes increasingly instinctive.

What you should be able to do

After working through this chapter, you should be able to:

- translate conditional rules, including "if", "only if", and necessary and sufficient conditions;
- form the converse, inverse and contrapositive of an implication;
- establish logical equivalence by a truth table or justified rewriting;
- negate compound and quantified claims without changing their domains;
- justify elementary deductions from stated premises;
- demonstrate an invalid argument by making its premises true and its conclusion false.

2.1 Conditional rules

Suppose a practice laboratory displays this rule:

> You may use the laboratory only if you have completed its safety briefing.

Alice has completed the briefing. Does the rule guarantee that she may use the laboratory? It does not. The rule makes the briefing a requirement; there could still be another requirement, such as a booking. Someone who has completed the briefing but has no booking would not contradict the displayed rule.

Now replace "only if" with "if":

> You may use the laboratory if you have completed its safety briefing.

Under this second rule, completing the briefing is enough to establish permission. The two sentences use almost the same words but commit the laboratory to different rules.

Recall that $P \to Q$ is false exactly when P is true and Q is false. Here P is the antecedent and Q is the consequent. Thus the first laboratory rule, with P meaning "Alice may use the laboratory" and Q meaning "Alice has completed the briefing", says $P \to Q$. It rules out permission without the briefing. It does not rule out a completed briefing without permission.

Example 2.1. In a practice access system, suppose the rule is "If a user has a staff pass, then the user may enter." For each user let P mean "the user has a staff pass" and Q mean "the user may enter". Consider these records:

User	P	Q	$P \to Q$
Alice	T	T	T
Bob	F	T	T
Charlie	F	F	T
David	T	F	F

David's record violates the rule: he has a staff pass, but entry is denied. Bob's record does not violate it. The rule has not said that a staff pass is the only route to permission; Bob might have a visitor pass. Charlie's record does not violate it either, because the rule has made no promise of entry for someone without a staff pass.

Bob's and Charlie's records illustrate the two cases with a false antecedent. We are checking whether this conditional rule is violated, not claiming that the consequent has happened. In particular, $P \to Q$ does not by itself assert P or Q, and it does not assert that P causes Q.

"If" and "only if". The part after "if" gives a sufficient starting condition. The part after "only if" gives a requirement on what precedes it. Compare:

Wording	Logical form
If P, then Q.	$P \to Q$
Q if P.	$P \to Q$
P only if Q.	$P \to Q$
P if Q.	$Q \to P$
Q only if P.	$Q \to P$

For example, "A file is released only if it has been checked" says released → checked. A released but unchecked file would violate that rule. A checked file that remains a draft would not.

❄ Definition 2.1 — Necessary and sufficient conditions

For the implication $P \to Q$, we say that P is a **sufficient condition for** Q, and that Q is a **necessary condition for** P. The condition P is enough to ensure Q; the condition Q is required whenever P holds.

Consequently, "P is necessary for Q" means $Q \to P$, whereas "P is sufficient for Q" means $P \to Q$. The direction depends on which condition is being described as necessary or sufficient, rather than its position at the beginning of the sentence.

Example 2.2. Suppose a practice portal grants access exactly when an account is verified *and* its payment is recorded. For one account, let V mean "the account is verified", C mean "its payment is recorded" and A mean "access is granted". The complete rule is

$$A \leftrightarrow (V \wedge C).$$

Verification is necessary for access: $A \to V$. It is not sufficient on its own. If V is TRUE but C is FALSE, then $V \wedge C$ is false, so access is not granted. The same reasoning shows that payment is necessary but is not sufficient on its own.

Together, verification and payment are sufficient: $(V \wedge C) \to A$. They are also necessary: $A \to (V \wedge C)$. The biconditional records both directions, which is what "exactly when" expresses here.

2.2 Converse, inverse and contrapositive

Reversing a rule or negating its components produces a new statement. We must specify which operation has been performed.

❄ Definition 2.2 — Related implications

Starting from $P \to Q$, its

- **converse** is $Q \to P$;
- **inverse** is $\neg P \to \neg Q$;
- **contrapositive** is $\neg Q \to \neg P$.

The contrapositive both reverses the direction and negates the two components.

Their truth values can be compared in one table:

P	Q	$P \to Q$	$Q \to P$	$\neg P \to \neg Q$	$\neg Q \to \neg P$
T	T	T	T	T	T
T	F	F	T	T	F
F	T	T	F	F	T
F	F	T	T	T	T

The original implication and its contrapositive agree in every row. The converse and inverse also agree with each other, but their columns do not match the original implication. We may therefore replace an implication by its contrapositive without changing its truth value. Reversing it alone does not have this guarantee.

Example 2.3. Let $x \in \mathbb{R}$. Consider the true claim "If $x > 5$, then $x > 2$." Its related statements are:

Name	Statement
Converse	If $x > 2$, then $x > 5$.
Inverse	If $x \leq 5$, then $x \leq 2$.
Contrapositive	If $x \leq 2$, then $x \leq 5$.

The value $x = 3$ refutes both the converse and the inverse: it is greater than 2 without being greater than 5, and it is at most 5 without being at most 2. The contrapositive is true for every real x, just as the original implication is.

Here "if $x > 5$, then $x > 2$" is understood as a claim about every $x \in \mathbb{R}$. Forming its contrapositive retains that domain and the universal quantifier. We have not negated the whole quantified claim.

A statement and its converse can both be true. For instance, for an integer n, "$n = 0$" and "$n^2 = 0$" imply each other. The table says that a true implication does not *guarantee* a true converse; it does not say that the converse must be false.

Biconditionals. The claim $P \leftrightarrow Q$ requires $P \to Q$ and $Q \to P$ together. It can therefore be written as

$$(P \to Q) \wedge (Q \to P).$$

If P is both necessary and sufficient for Q, the relationship is $P \leftrightarrow Q$. Establishing only one direction leaves half of this requirement unchecked.

2.3 Logical equivalence

Two descriptions may use different connectives yet impose exactly the same condition. The matching columns for an implication and its contrapositive give one example.

❄ Definition 2.3 — Logical equivalence

Two propositional expressions are **logically equivalent** when they have the same truth value for every assignment of truth values to their component propositional variables. We use $\equiv$ to record this relationship. For example,

$$P \to Q \;\equiv\; \neg Q \to \neg P.$$

In a truth table, equivalence means that the two final columns agree in *every* row. One differing row disproves equivalence. The letters P, Q, R below may stand for compound statements as well as atomic ones, provided we substitute the same statement consistently wherever its letter occurs.

Example 2.4. Compare $P \to Q$ with $\neg P \vee Q$:

P	Q	$\neg P$	$P \to Q$	$\neg P \vee Q$
T	T	F	T	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

The final columns agree throughout, so $P \to Q \equiv \neg P \vee Q$. Saying "Either the requirement is not met, or the promised outcome holds" excludes exactly the failed case P TRUE and Q FALSE. The "or" remains inclusive.

The symbol $\leftrightarrow$ is a connective within an expression. We can evaluate $P \leftrightarrow Q$ in one particular situation. By contrast, $P \equiv Q$ asserts agreement across all truth assignments. Two expressions are logically equivalent exactly when the biconditional between them is true in every row.

Example 2.5. Suppose P is TRUE and Q is FALSE. Then $P \vee Q$ and P are both true, so $(P \vee Q) \leftrightarrow P$ is true in this situation. However, if P is FALSE and Q is TRUE, then $P \vee Q$ is true while P is false. Their values now differ. Consequently, $P \vee Q$ and P are not logically equivalent.

❄ Definition 2.4 — Tautology, contradiction and contingency

A propositional expression is a **tautology** if it is true for every truth assignment, a **contradiction** if it is false for every truth assignment, and **contingent** if it is true for some assignments and false for others.

For example, $P \vee \neg P$ is a tautology: one of P and $\neg P$ is always true. The expression $P \wedge \neg P$ is a contradiction, because they cannot both be true. The expression $P \to Q$ is contingent: its truth table contains both truth values. One false row does not make a form a contradiction, just as one true row does not make it a tautology; the classification depends on all permitted assignments.

Equivalence laws. The following laws let us replace an expression by an equivalent one. Parentheses specify the part being replaced; they may not be moved or discarded arbitrarily.

Law	Equivalent expressions
Double negation	$\neg\neg P \equiv P$
Commutativity	$P \wedge Q \equiv Q \wedge P$; $P \vee Q \equiv Q \vee P$
Associativity	$(P \wedge Q) \wedge R \equiv P \wedge (Q \wedge R)$; $(P \vee Q) \vee R \equiv P \vee (Q \vee R)$
Idempotence	$P \wedge P \equiv P$; $P \vee P \equiv P$
Distributivity	$P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$; $P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$
De Morgan's laws	$\neg(P \wedge Q) \equiv \neg P \vee \neg Q$; $\neg(P \vee Q) \equiv \neg P \wedge \neg Q$
Implication	$P \to Q \equiv \neg P \vee Q$
Biconditional	$P \leftrightarrow Q \equiv (P \to Q) \wedge (Q \to P)$

These are laws of classical two-valued logic. They can be checked by complete truth tables, as we did for the implication law. Commutativity changes the order of two components; associativity changes grouping among repeated uses of the same connective. Neither permits changing "and" into "or". Distributivity explains when changing the outer connective is justified.

2.4 Negation and rewriting

Negation must reverse the truth value of the *whole* claim. Adding "not" to one convenient word does not necessarily achieve that.

Example 2.6. Fix one account. Let P mean "the account is verified" and Q mean "the account has a recorded payment". The claim "It is not the case that the account is both verified and paid" is $\neg(P \wedge Q)$. De Morgan's law gives

$$\neg(P \wedge Q) \equiv \neg P \vee \neg Q.$$

At least one condition fails; both might fail. Replacing the claim by $\neg P \wedge \neg Q$ would wrongly insist that both fail. For a verified account without a recorded payment, P is true and Q is false. The original negation is true, but $\neg P \wedge \neg Q$ is false.

By comparison, "Neither verified nor paid" means $\neg(P \vee Q) \equiv \neg P \wedge \neg Q$. Here both conditions must fail.

The two De Morgan laws can also be read directly. An "and" claim fails when at least one component fails. An inclusive "or" claim fails only when both components fail. This reasoning explains why the connective changes when the negation is moved inside.

Example 2.7. Let V mean "the account is verified", S mean "it has a staff pass" and G mean "it has a guest pass". The eligibility condition is $V \wedge (S \vee G)$: verification is required together with at least one kind of pass. Distributivity gives

$$V \wedge (S \vee G) \equiv (V \wedge S) \vee (V \wedge G).$$

The right-hand form lists the two qualifying alternatives: verified with a staff pass, or verified with a guest pass. Verification must appear in both alternatives. The shorter-looking expression $(V \wedge S) \vee G$ would allow an unverified guest account to qualify.

To express failure of the original eligibility condition, we can write

$$\begin{aligned} \neg\big(V \wedge (S \vee G)\big) &\equiv \neg V \vee \neg(S \vee G) && \text{(De Morgan)} \\ &\equiv \neg V \vee (\neg S \wedge \neg G) && \text{(De Morgan).} \end{aligned}$$

Eligibility fails if verification is missing, or if both kinds of pass are missing. Each line replaces the previous condition by one with the same truth value under every assignment.

Example 2.8. Let P mean "Faith submitted the required file" and Q mean "Faith received an acknowledgement". The promise is $P \to Q$. Its failure is

$$\begin{aligned} \neg(P \to Q) &\equiv \neg(\neg P \vee Q) && \text{(implication law)} \\ &\equiv \neg\neg P \wedge \neg Q && \text{(De Morgan)} \\ &\equiv P \wedge \neg Q && \text{(double negation).} \end{aligned}$$

Faith submitted the file *and* did not receive an acknowledgement. This is exactly the failed row of the implication truth table. The inverse $\neg P \to \neg Q$ is a different claim about what happens when Faith does not submit; it is not the negation of the promise.

For a biconditional, failure means that the components have different truth values: either P holds without Q, or Q holds without P. Thus

$$\neg(P \leftrightarrow Q) \equiv (P \wedge \neg Q) \vee (\neg P \wedge Q).$$

For example, "The indicator is on if and only if the check passed" fails either when the indicator is on after a failed check, or when it is off after a successful check. Both kinds of mismatch must be included.

2.5 Quantifier negation

A universal claim fails when it has a counterexample. An existential claim fails when it has no witness. This gives the formal negation laws corresponding to the checks in Chapter 1.

❄ Definition 2.5 — Negation laws for quantifiers

For a predicate $P(x)$ on a fixed domain D,

$$\neg\big(\forall x \in D\, P(x)\big) \equiv \exists x \in D\, \neg P(x),$$

and

$$\neg\big(\exists x \in D\, P(x)\big) \equiv \forall x \in D\, \neg P(x).$$

Negate the condition and switch the quantifier, retaining the domain.

The first equivalence says "not every object has the property" exactly when "at least one object lacks it". The second says "no object has the property" exactly when "every object lacks it". This reasoning applies to all permitted elements, rather than treating a quantifier as an extra propositional variable in a truth table. The examples below use nonempty domains.

Example 2.9. Let $D = \{\text{Alice}, \text{Bob}, \text{Charlie}\}$ and let $C(x)$ mean "x has completed the practice task". Suppose Alice and Bob have completed it, but Charlie has not. Then

$$\neg\big(\forall x \in D\, C(x)\big) \equiv \exists x \in D\, \neg C(x)$$

is true, with Charlie as the witness on the right. However, $\forall x \in D\, \neg C(x)$ is false, because Alice and Bob have completed the task. One incomplete task refutes "everyone completed"; it does not establish "nobody completed".

The negation of "someone completed" would instead be $\forall x \in D\, \neg C(x)$. These are different starting claims, so their negations should be different too.

The negation of $\forall n \in \mathbb{N}\,(n \geq 4)$ is $\exists n \in \mathbb{N}\,(n < 4)$. We still choose a natural number, and the negation of $n \geq 4$ is $n < 4$, not $n \leq 4$. Here $n = 0$ is a witness for the negation. The negation of $\exists x \in \mathbb{R}\,(x^2 < 0)$ is $\forall x \in \mathbb{R}\,(x^2 \geq 0)$. Forming the negation and deciding whether it is true are separate tasks; in this second example the existential claim is false and its negation is true.

Example 2.10. Let D be a set of records. Write $S(x)$ for "x is a student record" and $C(x)$ for "x is complete". The rule that every student record is complete is

$$\forall x \in D\,\big(S(x) \to C(x)\big).$$

Its negation is obtained in two stages:

$$\begin{aligned}\neg\Big(\forall x \in D\,(S(x) \to C(x))\Big) &\equiv \exists x \in D\,\neg(S(x) \to C(x)) \\ &\equiv \exists x \in D\,\big(S(x) \wedge \neg C(x)\big).\end{aligned}$$

First the universal quantifier becomes existential. Then we negate the implication using the law from Section 2.4. The result says that some student record is incomplete. An incomplete staff record that is not a student record does not witness this failure, because it does not satisfy $S(x)$.

The contrapositive rule would be $\forall x \in D\,(\neg C(x) \to \neg S(x))$. It retains the universal quantifier and has the same meaning as the original rule. It must not be confused with the existential statement that refutes the rule.

Multiple quantifiers. Negate a statement from the outside in, applying one law at a time. For sets of users U and files F, and the predicate $A(u, f)$ meaning "user u may access file f",

$$\begin{aligned}\neg\Big(\forall u \in U\, \exists f \in F\, A(u,f)\Big) &\equiv \exists u \in U\, \neg\Big(\exists f \in F\, A(u,f)\Big)\\ &\equiv \exists u \in U\, \forall f \in F\, \neg A(u,f).\end{aligned}$$

The original says that each user can access at least one file. Its negation says that there is a user who cannot access *any* file. Each quantifier changes type, but the order of the variables remains user first, file second.

Example 2.11. Use the following new data: $U = \{\text{Alice}, \text{Bob}\}$, $F = \{\text{guide}, \text{report}\}$, and these permissions:

	guide	report
Alice	T	F
Bob	F	T

The claim $\forall u \in U\, \exists f \in F\, A(u, f)$ is true: Alice can access guide and Bob can access report. Its negation $\exists u \in U\, \forall f \in F\, \neg A(u, f)$ is false, because neither user is denied every file.

The tempting replacement $\forall u \in U\, \exists f \in F\, \neg A(u, f)$ is true in this same table: each user has an inaccessible file. It therefore cannot be the negation of the original, which is also true. Negating only the predicate while leaving both quantifiers unchanged has produced a different condition.

For comparison, the negation of "There is one file that every user may access" is

$$\neg\Big(\exists f \in F\, \forall u \in U\, A(u,f)\Big) \equiv \forall f \in F\, \exists u \in U\, \neg A(u,f).$$

Every file has at least one user who cannot access it; the user may depend on the file. In the last table, Bob supplies that failure for guide, and Alice supplies it for report. The two quantified negations answer different questions, just as the original claims do.

2.6 Valid arguments

An argument starts with claims that are being assumed and offers a conclusion. We need to check whether those assumptions are enough for that conclusion.

❄ Definition 2.6 — Premises, conclusion and validity

The **premises** are an argument's stated assumptions. Its **conclusion** is the claim it asks us to infer. An argument form is **valid** if there is no truth assignment in which all its premises are true and its conclusion is false. Such an assignment is a **countervaluation**; finding one establishes invalidity.

Validity concerns the connection between the premises and conclusion. It does not assert that the premises describe the actual situation. An argument is called **sound** when it is valid and its premises are in fact true.

Example 2.12. Truth-table verification. Consider the argument with premises $P \to Q$ and P, and conclusion Q. This pattern is called **modus ponens**. The extra column below records whether both premises are true together.

P	Q	$P \to Q$	$(P \to Q) \wedge P$
T	T	T	T
T	F	F	F
F	T	T	F
F	F	T	F

Only the first row makes both premises true, and in that row the conclusion Q is true. No countervaluation exists, so the argument is valid. The second row has a false conclusion, but it does not refute the argument: one premise, $P \to Q$, is false there.

If the premises are $A_1, \dots, A_k$ and the conclusion is C, this test amounts to asking whether

$$(A_1 \wedge \cdots \wedge A_k) \to C$$

is a tautology. It fails exactly in a row where every premise is true and the conclusion is false. An argument with inconsistent premises has no all-true premise row and is valid under this definition, but it cannot be sound: its premises cannot all be true together.

Deduction rules. Each row below is a valid pattern. All listed premises must be available before the conclusion can be used.

Pattern	Premises	Conclusion
Modus ponens	$P \to Q$, P	Q
Modus tollens	$P \to Q$, $\neg Q$	$\neg P$
Chaining implications	$P \to Q$, $Q \to R$	$P \to R$
Disjunctive syllogism	$P \vee Q$, $\neg P$	Q
Conjunction introduction	P, Q	$P \wedge Q$
Conjunction elimination	$P \wedge Q$	P (also Q)

For the next two examples, fix one data link between two devices. Let P mean “This link uses Wi-Fi” and Q mean “This link uses no data cable between the devices”. Our recurring implication is

$$P \to Q : \quad \text{If this link uses Wi-Fi, it uses no data cable.}$$

The claim concerns this particular link. Other parts of the network may still contain cables.

Example 2.13. Modus ponens. Suppose we know both of the following:

> If this link uses Wi-Fi, it uses no data cable.
> This link uses Wi-Fi.

We can conclude that this link uses no data cable. In symbols,

$$P \to Q, \qquad P, \qquad \therefore Q.$$

Here $\therefore$ means "therefore". The second premise establishes the condition required by the first. Modus ponens lets us infer its stated consequence.

Example 2.14. Modus tollens. Keep the same rule, but suppose we now know that this link uses a data cable:

> If this link uses Wi-Fi, it uses no data cable.
> This link uses a data cable.

The second premise is $\neg Q$. We conclude that this link does not use Wi-Fi, which is $\neg P$:

$$P \to Q, \qquad \neg Q, \qquad \therefore \neg P.$$

If P were true, the first premise would require Q to be true as well. That is incompatible with the given $\neg Q$. This is modus tollens.

Example 2.15. Chaining implications. Let $x \in \mathbb{R}$. We have the two rules

> If $x > 10$, then $x > 5$.
> If $x > 5$, then $x > 0$.

Combining them gives "If $x > 10$, then $x > 0$." The intermediate condition $x > 5$ connects the first rule to the second. With A meaning $x > 10$, B meaning $x > 5$ and C meaning $x > 0$, the pattern is

$$A \to B, \qquad B \to C, \qquad \therefore A \to C.$$

The conclusion is still conditional. We have not been told that $x > 10$, so we cannot conclude that $x > 0$ without a further premise.

Example 2.16. Disjunctive syllogism. Suppose a laptop has a usable Wi-Fi connection or a usable Ethernet connection, possibly both. It has no usable Wi-Fi connection. It must therefore have a usable Ethernet connection.

Let W mean "The laptop has usable Wi-Fi" and E mean "The laptop has usable Ethernet". The argument is

$$W \vee E, \qquad \neg W, \qquad \therefore E.$$

At least one alternative must hold, and the Wi-Fi alternative has been ruled out. This leaves Ethernet. If instead we knew W, we could not infer $\neg E$: the inclusive "or" also permits both connections to be usable.

Example 2.17. Conjunction introduction. Alice has a laptop. Alice also has a charger. We can combine these two facts into "Alice has a laptop and a charger." Let L mean "Alice has a laptop" and C mean "Alice has a charger". Then

$$L, \qquad C, \qquad \therefore L \wedge C.$$

Both components have already been established separately. Conjunction introduction allows us to state them together; either fact on its own would not establish the whole conjunction.

Example 2.18. Conjunction elimination. Now suppose the given premise is "Alice has a laptop and a charger." With the same meanings of L and C, we have $L \wedge C$. We may conclude that Alice has a laptop, and we may also conclude that she has a charger:

$$L \wedge C \therefore L, \qquad \text{and} \qquad L \wedge C \therefore C.$$

An "and" claim requires both components to be true. Each component is therefore available for use on its own in a later step.

Example 2.19. For one application, let P mean "its identity check is complete", Q mean "its payment is recorded", R mean "it is ready for review" and S mean "it is sent to the reviewer". Suppose the premises are

$$P \wedge Q, \qquad (P \wedge Q) \to R, \qquad R \to S.$$

The first two premises give R by modus ponens. Combining this result with the third gives S by modus ponens again. Alternatively, chaining the two implications first gives $(P \wedge Q) \to S$, to which the first premise can be applied.

If the recorded fact had been $P \vee Q$ instead of $P \wedge Q$, the first application would not be justified. A completed identity check without a recorded payment makes the "or" true but does not supply the required "and". A rule must be applied to the condition it actually requires.

Universal premises. A universal premise may supply the conditional needed in a particular case. If $\forall x \in D\,(V(x) \to A(x))$ is a premise and Alice $\in D$, it gives $V(\text{Alice}) \to A(\text{Alice})$. Together with $V(\text{Alice})$, modus ponens yields $A(\text{Alice})$. Membership in the stated domain is part of this justification. Applying the rule to an object outside D would not be licensed by the universal premise.

2.7 Invalid arguments

A conclusion can happen to be true even when an argument for it is invalid. To assess the argument, we examine what its premises require across all possible assignments, rather than checking only the situation we already know. A countervaluation must make every premise true in the same assignment and the conclusion false. Making different premises true in different assignments, or choosing a row with a false premise, does not refute the form.

Example 2.20. Consider this argument:

> If $8 > 3$, then $8 > 1$. Also, $8 > 1$. Therefore $8 > 3$.

All three numerical statements are true. Its proposed argument form, however, has premises $P \to Q$ and Q, with conclusion P. Take P FALSE and Q TRUE. Both premises are then true, but the conclusion is false. The form does not guarantee its conclusion, even though this particular numerical conclusion happens to be correct.

Example 2.21. Affirming the consequent. Return to the link between two devices: P means "This link uses Wi-Fi" and Q means "This link uses no data cable between the devices". The attempted argument is

$$P \to Q, \qquad Q, \qquad \text{therefore } P.$$

This failed pattern is called **affirming the consequent**. A countervaluation is P FALSE, Q TRUE:

P	Q	$P \to Q$	Conclusion P
F	T	T	F

The premise Q is true in this row, as is the conditional. The devices could be communicating using Bluetooth: the link uses no data cable, but it does not use Wi-Fi. The absence of a cable therefore does not establish Wi-Fi. Replacing the original premise by a biconditional would supply the missing converse, but that would be an additional assumption, not a deduction from the given rule.

Python (non-examinable)

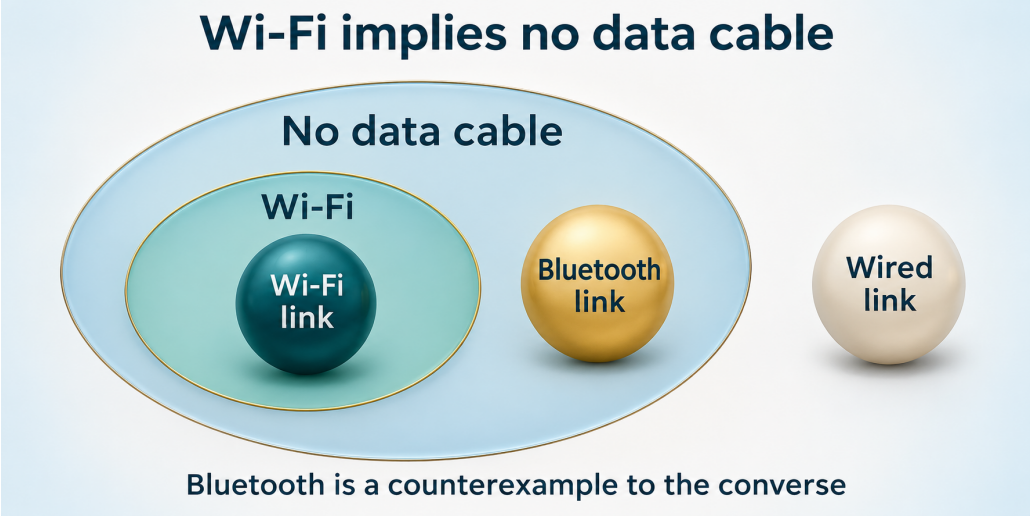
To test the argument $P \to Q,\ Q,\ \therefore P$, let p and q represent the truth values of P and Q. The two `for` loops consider `True` and `False` for each name, listing every assignment. The assignment to `premise` stores the truth value of $P \to Q$. An `if` statement runs its indented line only when its condition is true: here, when both premises hold and the conclusion P fails.

```
for p in [True, False]:
    for q in [True, False]:
        premise = (not p) or q
        if premise and q and not p:
            print(p, q)
```

Output.

```
False True
```

The output gives P false and Q true. Both premises hold while the conclusion P fails, so this countervaluation establishes invalidity.



Wi-Fi links lie within the no-data-cable case; a Bluetooth link satisfies the consequent without satisfying the antecedent.

Example 2.22. Denying the antecedent. Using the same meanings, suppose someone argues

$$P \to Q, \qquad \neg P, \qquad \text{therefore } \neg Q.$$

This is **denying the antecedent**. Again set P FALSE and Q TRUE. The conditional is true and $\neg P$ is true, while the conclusion $\neg Q$ is false. A link that does not use Wi-Fi need not use a data cable: a Bluetooth link again gives a counterexample.

Compare this with modus tollens: that valid pattern starts from $\neg Q$ and concludes $\neg P$. Negating both letters without reversing their roles does not reproduce the same argument.

2.8 Exercises

2.1. Fix one request and a recorded time. Let P mean "The request is verified" and Q mean "The request is authorised". Translate each claim into symbols.

a) If the request is verified, then it is authorised.

b) The request is verified only if it is authorised.

c) The request is verified if it is authorised.

d) Verification is sufficient for authorisation.

e) Verification is necessary for authorisation.

f) The request is authorised if and only if it is verified.

2.2. For each saved file, let C mean "The file has been checked" and R mean "The file has been released". The following are the recorded values.

File	C	R
alpha	T	T
beta	T	F
gamma	F	T
delta	F	F

a) Write "A file is released only if it has been checked" symbolically.

b) Which file violates that rule? Explain the failed condition.

c) Does the rule require every checked file to be released? Use one row of the table to explain your answer.

2.3. Let $D = \{0, 1, 2, 3, 4, 5\}$. Consider the claim $\forall n \in D\,(n > 4 \to n > 2)$.

a) Write its converse, inverse and contrapositive, retaining the domain.

b) Decide which of the original claim and the three claims in part (a) are true. Give a counterexample to each false claim.

c) Negate the original claim, leaving no negation outside a quantifier or implication. Is this negation the same claim as the contrapositive?

2.4. Let R mean "Bob is registered" and L mean "Bob may enter the laboratory". Suppose the stated rule is $R \to L$.

a) Which condition is sufficient for the other, and which is necessary?

b) If L is true, must R be true? Give truth values showing what the stated rule allows.

c) What additional conditional would turn the rule into $R \leftrightarrow L$?

d) If both R and L are false, does the biconditional in part (c) hold? Explain using its two conditional directions.

2.5. Negate each formula. Give an equivalent formula in which every negation applies directly to a propositional variable.

a) $P \wedge Q$.

b) $P \vee \neg Q$.

c) $P \to Q$.

d) $(P \wedge Q) \to R$.

2.6. Classify each formula as a tautology, a contradiction or contingent. For each contingent formula, give one assignment making it true and one making it false.

a) $P \vee \neg P$.

b) $P \wedge \neg P$.

c) $(P \to Q) \wedge P$.

d) $(P \wedge Q) \to P$.

2.7. For each pair, construct a truth table with a column for each complete formula, then decide whether the formulas are logically equivalent.

a) $P \to Q$ and $\neg Q \to \neg P$.

b) $P \to Q$ and $Q \to P$.

2.8. For each pair, give one assignment of truth values making the formulas different, and evaluate both formulas on that assignment.

a) $\neg(P \wedge Q)$ and $\neg P \wedge \neg Q$.

b) $P \vee (Q \wedge R)$ and $(P \vee Q) \wedge R$.

c) $P \to Q$ and $\neg P \to \neg Q$.

2.9. Show each equivalence by a short chain of equivalent formulas. Give the reason for each substantive step.

a) $\neg(P \to \neg Q) \equiv P \wedge Q$.

b) $\neg(\neg P \vee Q) \vee (P \wedge R) \equiv P \wedge (\neg Q \vee R)$.

2.10. Supply a justification for each of the three equivalence steps:

$$\begin{aligned}\neg\big(P \wedge (Q \vee R)\big) &\equiv \neg P \vee \neg(Q \vee R) \\ &\equiv \neg P \vee (\neg Q \wedge \neg R) \\ &\equiv (\neg P \vee \neg Q) \wedge (\neg P \vee \neg R).\end{aligned}$$

2.11. Write the negation of each claim, with explicit domains and no negation outside a quantifier. Simplify the numerical conditions as well.

a) $\forall n \in \mathbb{Z}\,(n \geq -2)$.

b) $\exists n \in \mathbb{Z}\,(n^2 = 3)$.

c) $\forall x \in \mathbb{R}\,(x < 1 \vee x > 4)$.

d) $\exists n \in \mathbb{N}\,(n > 2 \wedge n < 5)$.

2.12. Let D be a set of records. Let $S(r)$ mean "r is a student record" and $C(r)$ mean "r is complete".

a) Negate $\forall r \in D\,(S(r) \to C(r))$ and express the result in English.

b) Negate $\exists r \in D\,(S(r) \wedge C(r))$ and express the result in English.

c) Hannah proposes $\forall r \in D\,(S(r) \to \neg C(r))$ as the answer to part (a). Give a two-record example in which your answer to part (a) is true but Hannah's proposed answer is false.

2.13. For this exercise, use the following new permission data. Let $U = \{\text{Alice}, \text{Bob}, \text{Grace}\}$ and $F = \{\text{guide}, \text{report}, \text{ledger}\}$. Let $A(u, f)$ mean "user u may access file f". The table lists all permissions: 1 means permitted and 0 means not permitted.

	guide	report	ledger
Alice	1	1	0
Bob	0	1	0
Grace	1	0	0

a) Negate $\forall u \in U\, \exists f \in F\, A(u, f)$, with no negation outside a quantifier. Express your answer in English.

b) Do the same for $\exists f \in F\, \forall u \in U\, A(u, f)$.

c) Use the table to decide the truth value of each negated claim. Explain which users or files establish your answers.

2.14. For each argument, identify its propositional form and decide whether that form is valid. Separately, state the actual truth values of its premises and conclusion. In judging the form, treat its component statements as propositional variables.

a) If 3 is even, then 4 is even. The number 3 is even. Therefore 4 is even.

b) If 4 is even, then 4^2 is even. The number 4^2 is even. Therefore 4 is even.

2.15. Decide whether each argument form is valid. Name a rule that justifies a valid form. For an invalid form, give truth values making every premise true and the conclusion false.

a) Premises: $P \to Q$, P. Conclusion: Q.

b) Premises: $P \to Q$, $\neg Q$. Conclusion: $\neg P$.

c) Premises: $P \to Q$, Q. Conclusion: P.

d) Premises: $P \to Q$, $\neg P$. Conclusion: $\neg Q$.

2.16. For one fixed file, let C mean "The file has been checked", L mean "The file has been logged", and A mean "The file has been archived". Assume $C \to L$ and $L \to A$.

a) If $\neg A$ is also known, derive $\neg C$ through an intermediate conclusion. Justify each step.

b) If instead $\neg C$ is known, may we conclude $\neg A$? Either justify the inference or give truth values satisfying the premises and refuting the conclusion.

2.17. Let P mean "A usable primary backup exists" and M mean "A usable mirror backup exists". The only general requirement is $P \vee M$.

a) If $\neg P$ is known, what follows about M? Name the inference rule.

b) If P is known instead, can we infer $\neg M$? Explain with a truth assignment that respects the general requirement.

2.18. Let C mean "The request was checked" and L mean "The request was logged". A classmate argues: "$C \to L$; L; therefore C."

a) Identify the invalid form and give a countervaluation.

b) Suppose the intended policy actually says that a request is logged if and only if it was checked. Write the two conditional directions of that policy, and explain how they now justify C from L.

2.19. Let $U = \{\text{Alice}, \text{Bob}, \text{Grace}\}$. For $u \in U$, let $R(u)$ mean "u is registered", $C(u)$ mean "u completed the practical", and $A(u)$ mean "u is permitted to enter". The policy is

$$\forall u \in U \, \big(A(u) \leftrightarrow (R(u) \wedge C(u))\big).$$

The known facts are $A(\text{Alice})$, $\neg C(\text{Bob})$ and $R(\text{Grace})$.

a) What follows about Alice's registration and completion of the practical?

b) What follows about Bob's permission to enter? Identify the necessary condition that determines your answer.

c) Do the known facts settle whether Grace may enter? If not, describe two possibilities for her record, each consistent with the policy and the facts.

d) Negate "Every registered user is permitted to enter", first writing the claim and then its negation symbolically with domain U.

e) Do the known facts settle whether "Every registered user is permitted to enter" is true? If not, complete the unspecified facts in two ways: one in which that claim is true and one in which it is false, while retaining the policy and every known fact.

LECTURE NOTES
DISCRETE MATHEMATICS

CHAPTER 3

Proof methods



Updated: 19 November 2026

3 Proof methods

Alice claims that the square of every odd integer is odd. She tries $3^2 = 9$, $5^2 = 25$ and $(-7)^2 = 49$. Bob checks another thousand odd integers, and every result agrees. Have they proved the claim?

Their calculations give examples, but infinitely many integers remain unchecked. A proof must explain why an odd integer cannot produce an even square. We will return to Alice's claim and give an argument that works for every odd integer, including negative ones.

Chapter 2 supplied rules for drawing conclusions from premises. We now use those rules together with mathematical definitions to construct complete arguments. Instead of being handed both the premises and the next step, we must decide how to begin, what information a definition provides and what would establish the conclusion.

Our three main methods are direct proof, proof by contrapositive and proof by contradiction. We will practise them on elementary number properties so that the reasoning remains visible. Finding a proof can take several attempts; the finished argument usually reads more smoothly than the work that led to it. With practice, choosing a useful starting point becomes easier.

What you should be able to do

After working through this chapter, you should be able to:

- identify a claim's domain, hypotheses and required conclusion;
- use the definitions of even, odd and divisibility in an argument;
- construct short direct proofs, contrapositive proofs and contradiction proofs;
- explain why an argument covers an arbitrary permitted input;
- disprove a universal claim with a permitted counterexample;
- locate and repair missing assumptions, unjustified steps and wrong proof directions.

3.1 Mathematical proof

Recall the distinction from Chapter 2: a valid argument form guarantees its conclusion when its premises are true. A mathematical proof must also supply the mathematical reasons that make its steps available. For example, knowing the form $P \to Q,\ P,\ \therefore Q$ does not by itself tell us whether a square is even. We must connect the numerical claim to definitions and established facts.

❄ Definition 3.1 — Proof

A **proof** is a mathematical argument that establishes a claim from its hypotheses, definitions and previously established facts, using valid reasoning. Each step must be justified, and the argument must cover the cases required by the claim.

A proved result may be called a **theorem** or a **proposition**. A **lemma** is a result proved for use in another argument.

Hypotheses and conclusions. Consider

$$\forall x \in D \left(P(x) \to Q(x)\right).$$

The **hypothesis** is $P(x)$ and the required conclusion is $Q(x)$, with x belonging to D. In a direct proof, we choose an arbitrary $x \in D$, assume $P(x)$ and derive $Q(x)$. We are establishing what happens *if* the hypothesis holds; we do not have to show that every element satisfies the hypothesis.

"Arbitrary" means that we have not selected an input with extra helpful properties. If a claim concerns all integers, beginning with $n = 3$ would restrict the argument to one input. Beginning with "Let n be an integer" leaves it available for every integer allowed by the claim. We may then impose exactly the hypotheses the claim gives us.

Example 3.1. Prove that for every real number x, if $x > 7$, then $2x + 1 > 15$.

The claim has domain $\mathbb{R}$, hypothesis $x > 7$ and conclusion $2x + 1 > 15$. We need no numerical value for x.

Proof. Let $x \in \mathbb{R}$ and suppose that $x > 7$. Multiplication by the positive number 2 preserves the inequality, so $2x > 14$. Adding 1 gives $2x + 1 > 15$, as required. □

The argument applies to every real x satisfying the hypothesis. The small square marks the end of the proof. The reason for multiplying by a *positive* number was stated because multiplying an inequality by a negative number would reverse its direction.

Trying values can suggest a claim, reveal a counterexample or help us find the calculation to use. The final proof must explain what covers the untested inputs. A complete check of a finite domain can do that, as in Chapter 1. For an infinite domain, a finite collection of successful tests alone does not.

You can also work backwards while searching. If the goal is to show that a number equals twice an integer, it is reasonable to ask which integer would work. When presenting the proof, establish that representation from the hypotheses. Simply assuming the desired conclusion and obtaining a true statement does not show that the conclusion follows.

3.2 Parity and divisibility

We need a small amount of arithmetic vocabulary for our proof examples. Throughout the core examples, our number variables range over the integers unless a different domain is specified. Sums, differences and products of integers are integers. A quotient of two integers need not be an integer: for instance, $5/2 \notin \mathbb{Z}$.

❄ Definition 3.2 — Even and odd integers

An integer n is **even** if $n = 2k$ for some integer k. It is **odd** if $n = 2k + 1$ for some integer k. The integer k is a witness to the stated representation.

An integer's **parity** is whether it is even or odd. The definitions turn these familiar labels into forms we can use in algebra. To use the hypothesis "n is even", we may write $n = 2k$ for some $k \in \mathbb{Z}$. To prove that a quantity is even, we must show it equals twice an *integer*. For oddness, the target is twice an integer plus 1.

Example 3.2. The number -10 is even because $-10 = 2(-5)$, with integer witness -5. The number -7 is odd because $-7 = 2(-4) + 1$, with witness -4. Zero is even because $0 = 2(0)$.

We do not require k in the definitions to be positive. Restricting it to positive integers would exclude valid negative examples and zero.

We use the familiar fact that every integer is either even or odd. Dividing an integer by 2 leaves remainder 0 or 1, giving precisely the forms $2k$ and $2k + 1$. We will study remainders systematically in Chapter 8.

The alternatives cannot both hold. If $2k = 2\ell + 1$ for integers k, ℓ, then $k - \ell = \frac{1}{2}$, but an integer difference cannot equal $\frac{1}{2}$. Thus, for an integer, "not even" means odd, and "not odd" means even. These conclusions depend on the integer domain; a real number such as $\frac{1}{2}$ is neither an even nor an odd integer.

❄ Definition 3.3 — Divisibility

For integers d, n with $d \neq 0$, write $d \mid n$ if $n = dq$ for some integer q. Read this as "d divides n" or "n is a multiple of d". Write $d \nmid n$ when no such integer q exists. The divisors used in this chapter are nonzero.

The notation $d \mid n$ makes a claim; it is not a request to carry out a division. In particular, $d \mid n$ and $n \mid d$ have different meanings. The witness q must be an integer.

Example 3.3. We have $6 \mid 42$ because $42 = 6(7)$. However, $6 \nmid 20$: the equation $20 = 6q$ forces $q = 10/3$, which is not an integer. Writing $20 = 6(10/3)$ does not establish divisibility.

Also, $6 \mid 0$ because $0 = 6(0)$. The number being divided may be zero even though we have required the divisor to be nonzero.

Evenness is divisibility by 2. More generally, the definition of divisibility tells us both how to unpack a hypothesis and what kind of witness to construct for a conclusion. We use only this elementary meaning here; the number-theory chapter develops it further.

3.3 Direct proof

In a direct proof, we begin with what is given and work towards what is required. Definitions often supply the first useful equation.

❄ Definition 3.4 — Direct proof of an implication

To prove $P \to Q$ directly, assume P and deduce Q using definitions and established results. For a universally quantified claim, first fix an arbitrary object in the stated domain and retain its hypotheses throughout the argument.

Example 3.4. Prove that the sum of two even integers is even.

The hypotheses concern two numbers, so use two representations: $a = 2r$ and $b = 2s$. The goal is to write $a + b$ as twice an integer.

Proof. Let $a, b \in \mathbb{Z}$ and suppose both are even. There are integers r, s such that $a = 2r$ and $b = 2s$. Therefore

$$a + b = 2r + 2s = 2(r + s).$$

Since $r + s$ is an integer, this represents $a + b$ as twice an integer. Hence $a + b$ is even. □

The witness for the conclusion is $r + s$. We have explained why it belongs to $\mathbb{Z}$, rather than merely pointing to a factor of 2.

✍ Separate witnesses

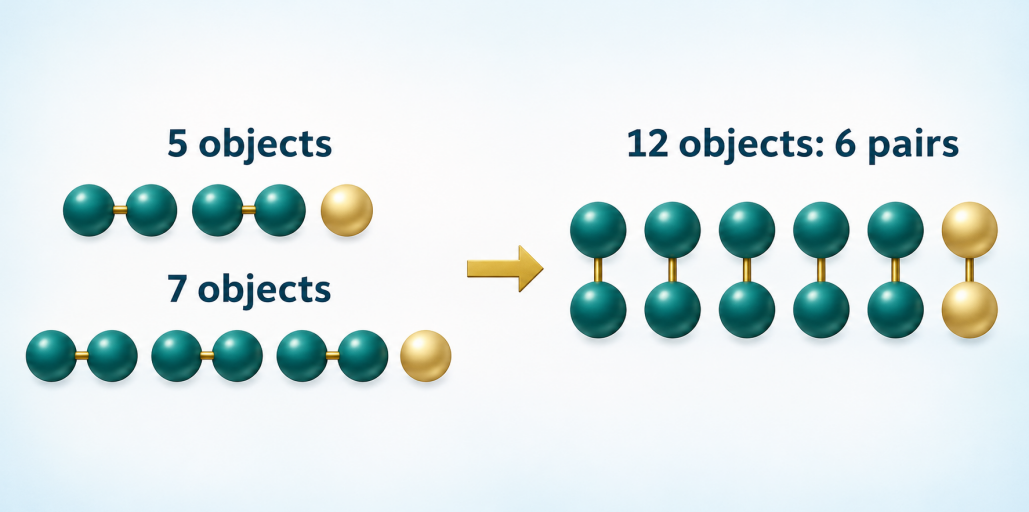
Writing $a = 2k$ and $b = 2k$ with the same k would assume $a = b$. The original claim allows different even integers, such as 6 and 10, whose witnesses are 3 and 5. Use separate letters when the hypotheses do not require the witnesses to agree. They may turn out to be equal in a particular case, but the proof must not impose that restriction.

Example 3.5. Prove that the sum of two odd integers is even.

Proof. Let $a, b \in \mathbb{Z}$ be odd. Write $a = 2r + 1$ and $b = 2s + 1$ for integers r, s. Then

$$a + b = (2r + 1) + (2s + 1) = 2r + 2s + 2 = 2(r + s + 1).$$

The number $r + s + 1$ is an integer, so $a + b$ is even. □



In the example 5 + 7, the two unpaired objects form a sixth pair.

Two odd inputs have produced an even output. The definition needed at the end is the definition of the *conclusion*, not necessarily the one used to describe the inputs.

Example 3.6. Squares of odd integers. We can now settle Alice's claim from the opening.

Proof. Let $n \in \mathbb{Z}$ be odd. Then $n = 2k + 1$ for some $k \in \mathbb{Z}$. Squaring gives

$$\begin{aligned} n^2 &= (2k+1)^2 \\ &= 4k^2 + 4k + 1 \\ &= 2(2k^2 + 2k) + 1. \end{aligned}$$

Since $2k^2 + 2k$ is an integer, the final expression has the form 2(integer) + 1. Thus n^2 is odd. □

The proof did not select a particular odd input or assume that k was positive. It covers every odd integer allowed by the claim. The numerical tests helped us notice the pattern; the representation $n = 2k + 1$ explains why it always holds.

Example 3.7. Prove that the product of an even integer and any integer is even.

Proof. Let $a, b \in \mathbb{Z}$ and suppose a is even. Write $a = 2k$ for some $k \in \mathbb{Z}$. Then

$$ab = (2k)b = 2(kb).$$

Because $kb \in \mathbb{Z}$, the product ab is even. □

We did not need b to be even. We did need it to be an integer: $2(\frac{1}{2}) = 1$ shows why an arbitrary real second factor would not support the same conclusion.

Example 3.8. Prove that if $d \mid a$ and $d \mid b$, then $d \mid (a + b)$, where $a, b, d \in \mathbb{Z}$ and $d \neq 0$.

Proof. Fix such integers and assume $d \mid a$ and $d \mid b$. By definition, $a = dr$ and $b = ds$ for some integers r, s. Hence

$$a + b = dr + ds = d(r + s).$$

Since $r + s \in \mathbb{Z}$, the definition of divisibility gives $d \mid (a + b)$. □

The conclusion is a divisibility claim, so it needs an integer multiplier. Here that multiplier is $r + s$, just as in the earlier even-sum proof.

Example 3.9. Suppose $d \mid a$ and $d \mid b$, with $a, b, d \in \mathbb{Z}$ and $d \neq 0$. Prove that $d \mid (3a - 2b)$.

Proof. There are integers j, k with $a = dj$ and $b = dk$. Substituting these expressions gives

$$3a - 2b = 3dj - 2dk = d(3j - 2k).$$

The multiplier $3j - 2k$ is an integer, so $d \mid (3a - 2b)$. □

The same reasoning works for any integer coefficients r, s:

$$ra + sb = r(dj) + s(dk) = d(rj + sk).$$

Because r, s, j, k are integers, so is $rj + sk$. Thus $d \mid (ra + sb)$.

Example 3.10. For integers a, b, c with $a, b \neq 0$, prove that $a \mid b$ and $b \mid c$ imply $a \mid c$.

Proof. Assume $a \mid b$ and $b \mid c$. Choose integers r, s such that $b = ar$ and $c = bs$. Substitution gives

$$c = bs = (ar)s = a(rs).$$

Since $rs \in \mathbb{Z}$, we have $a \mid c$. □

Compare this with chaining implications in Chapter 2. Here the definitions provide the equations that connect the two hypotheses. The direction matters: $a \mid b$ supplied $b = ar$, not $a = br$.

Each proof above begins with the domain and hypotheses, introduces what the definitions permit, and explains why the final expression meets the definition required by the conclusion. Algebra performs part of the work; the sentences identify what the algebra establishes.

3.4 Proof by contrapositive

Sometimes the hypothesis supplies information about a complicated expression, while the conclusion concerns a simpler input. It may be easier to start by assuming that the conclusion is false.

Chapter 2 established

$$(P \to Q) \equiv (\neg Q \to \neg P).$$

The second implication is the contrapositive of the first. Proving it therefore establishes the original implication.

❄ Definition 3.5 — Proof by contrapositive

To prove $P \to Q$ by contrapositive, assume $\neg Q$ and deduce $\neg P$. Keep the original domain and any background hypotheses. For a quantified implication, the argument must work for an arbitrary permitted input.

Example 3.11. Prove that for every integer n, if n^2 is even, then n is even.

A direct attempt gives $n^2 = 2k$ for an integer k, but that does not yet express n as twice an integer. The contrapositive starts with n odd and asks us to show that n^2 is odd, which Example 3.6 already established.

Proof. We prove the contrapositive. Let $n \in \mathbb{Z}$ and suppose that n is not even. Then n is odd, so Example 3.6 shows that n^2 is odd and hence not even. This proves that if n is not even, then n^2 is not even. The original implication follows by contraposition. □

We used a previously proved result instead of repeating its calculation. The proof did not assume that n^2 was even; that is the starting assumption for a direct proof of the original implication.

Example 3.12. For $n \in \mathbb{Z}$, prove that if $3n + 2$ is odd, then n is odd.

The contrapositive is: if n is not odd, then $3n + 2$ is not odd. Since these quantities are integers, we can rewrite it using evenness.

Proof. We prove the contrapositive. Let $n \in \mathbb{Z}$ be even. Then $n = 2k$ for some $k \in \mathbb{Z}$, so

$$3n + 2 = 3(2k) + 2 = 2(3k + 1).$$

The multiplier $3k + 1$ is an integer. Hence $3n + 2$ is even and is not odd. This proves the contrapositive, so the stated implication is true. □

Example 3.13. For integers a, b, prove that if $a + b$ is odd, then a is odd or b is odd.

Negating the conclusion requires De Morgan's law: "not (a odd or b odd)" means that neither is odd. Both are then even.

Proof. We prove the contrapositive. Let $a, b \in \mathbb{Z}$ and suppose neither is odd. Both are even, so Example 3.4 shows that $a + b$ is even. Thus $a + b$ is not odd, which proves the contrapositive. Therefore an odd sum has at least one odd summand. □

The "or" is inclusive. We have proved the implication stated in the claim, not its converse: 3 and 5 are both odd, but $3 + 5$ is even.

✍ Contrapositive and negation

For a claim $\forall n \in \mathbb{Z}\,(P(n) \to Q(n))$, the contrapositive retains the universal quantifier:

$$\forall n \in \mathbb{Z}\,(\neg Q(n) \to \neg P(n)).$$

The negation of the whole claim is instead $\exists n \in \mathbb{Z}\,(P(n) \wedge \neg Q(n))$. It says that a counterexample exists. These have different jobs: the contrapositive is an equivalent claim to prove; the negation describes how the original could fail.

3.5 Proof by contradiction

A nonexistence claim can be difficult to attack by listing things that do not work. A contradiction proof lets us assume that a proposed object exists and ask what that would force.

❄ Definition 3.6 — Proof by contradiction

To prove a claim S, temporarily assume $\neg S$, while retaining the stated domain and background hypotheses. Deduce incompatible conclusions, such as a statement together with its negation. These show that the temporary assumption is impossible, so S holds.

The contradiction must follow from justified steps. Being surprised by an answer, or merely failing to find an example, does not establish one. We must identify the incompatible facts.

To prove a conditional $P \to Q$ this way, its negation is $P \wedge \neg Q$. For a universal conditional, we suppose that there is a permitted counterexample satisfying those two conditions. We do not suppose that every input is a counterexample.

Example 3.14. A greatest integer would be an integer at least as large as every integer. The claim says that no such integer exists.

Proof. Suppose, for contradiction, that a greatest integer g exists. The number $g + 1$ is an integer because integers are closed under addition. It satisfies $g + 1 > g$. However, greatestness would require $g + 1 \leq g$. These inequalities cannot both hold. Therefore no greatest integer exists. □

We did not try a long list of proposed greatest integers. The assumption gave us one candidate g, and the expression $g + 1$ defeated that candidate regardless of its value.

Example 3.15. Prove that there are no integers a, b satisfying $6a + 3 = 4b$.

Proof. Suppose, for contradiction, that integers a, b satisfy the equation. Its left side can be written as

$$6a + 3 = 2(3a + 1) + 1,$$

so it is odd, because $3a + 1$ is an integer. Its right side is $4b = 2(2b)$, so it is even. The equation would make the same integer both odd and even, which is impossible. Hence no such integers exist. □

The negation of "there are no such integers" is the existence of a pair that works. That is precisely what we temporarily assumed; the parity definitions then ruled out every proposed pair.

Example 3.16. We can also prove that an integer with an even square must be even using contradiction. Compare the assumptions with the contrapositive proof in Example 3.11.

Proof. Let $n \in \mathbb{Z}$ and suppose n^2 is even. Suppose, for contradiction, that n is not even. Then n is odd, so Example 3.6 gives that n^2 is odd. This contradicts our hypothesis that n^2 is even. Therefore n is even. □

This proof retained the original hypothesis and temporarily denied the conclusion. The contrapositive proof assumed only that n was not even and established that n^2 was not even. Both methods are valid, but their stated assumptions and stopping points differ.

3.6 Proof construction and checking

The logical form gives us possible starting points, but it does not guarantee that one particular method will be shortest. For a claim $P \to Q$, compare the following:

Method	Starting point	What to establish
Direct	Assume P.	Deduce Q.
Contrapositive	Assume $\neg Q$.	Deduce $\neg P$.
Contradiction	Assume P and $\neg Q$.	Deduce incompatible facts.

In each row, keep the domain and any background hypotheses. For a general claim S that is not presented as a conditional, contradiction starts with $\neg S$. We often choose a method because its assumptions give a representation we know how to use.

For instance, knowing that n is odd immediately gives $n = 2k + 1$. Knowing only that n^2 is even gives an equation for the square, which is less convenient if we want an equation for n itself. This was the reason for trying the contrapositive in Example 3.11.

Example 3.17. Prove that an integer n is even if and only if n^2 is even.

Proof. Let $n \in \mathbb{Z}$. First suppose n is even. By Example 3.7, its product with the integer n is even, so n^2 is even.

Conversely, suppose n^2 is even. Example 3.11 gives that n is even. Both implications hold, so the biconditional is proved. □

The first direction used a direct proof result; the second used one proved by contrapositive. A biconditional does not require using the same method in both directions.

Counterexamples. Before investing in a proof, test simple allowed values, including zero or negative inputs when the domain permits them. One counterexample to a universal claim settles its falsity. In an implication, that counterexample must make the hypothesis true and the conclusion false. A false hypothesis does not supply a failed case.

Example 3.18. Consider the proposed claim: for nonzero integer d and integers a, b, if $d \mid ab$, then $d \mid a$ or $d \mid b$.

Take $d = 6$, $a = 2$ and $b = 3$. We have $ab = 6$, so $6 \mid ab$. However, $6 \nmid 2$ and $6 \nmid 3$. The hypothesis is true and the conclusion false, so this is a counterexample.

The universal claim is false. The definition of divisibility does not allow us to separate the factors in this way for an arbitrary d.

Python (non-examinable)

Consider the claim that, for integers a, b, if $6 \mid ab$, then $6 \mid a$ or $6 \mid b$. We can search for counterexamples with $d = 6$. Python's `range(1, 7)` supplies the integers 1 through 6; the upper endpoint is excluded. The operator `%` gives the remainder, so `a % d == 0` tests whether d divides a. The assignments below record the hypothesis and conclusion for each pair. The `if` line selects exactly a true hypothesis and a false conclusion.

Python (non-examinable) — continued

```
d = 6
for a in range(1, 7):
    for b in range(1, 7):
        hypothesis = (a * b) % d == 0
        conclusion = (a % d == 0) or (b % d == 0)
        if hypothesis and not conclusion:
            print(a, b)
```

Output.

```
2 3
3 2
3 4
4 3
```

The first pair recovers the counterexample verified above. Each printed pair supplies another one, and any one of them refutes the universal claim. Successful tests alone would not prove a claim about all integers.

Faulty proofs. The next examples separate the result being claimed from the argument offered for it. Check where each argument starts, what its equations actually establish and which hypotheses it uses.

Example 3.19. Charlie wants to prove that if n^2 is even, then n is even. He writes: "Let n be even. Then $n = 2k$, so $n^2 = 2(2k^2)$ is even. Therefore the claim is proved."

His calculation establishes that n even implies n^2 even. He began with the intended conclusion and proved the converse. Although both directions happen to be true here, that does not make one a proof of the other. The argument must begin with assumptions appropriate to its chosen method.

A repair is the contrapositive proof in Example 3.11, or the contradiction proof in Section 3.5.

Example 3.20. Eve proposes that every integer is even because

$$n = 2\left(\frac{n}{2}\right).$$

The equation is correct, but evenness requires an integer witness. When $n = 5$, the proposed witness is $5/2$, which is not an integer. The equation therefore does not prove evenness for an arbitrary n.

Example 3.21. Alice claims that for all integers a, b, c, the equation $ab = ac$ implies $b = c$. Her proof divides both sides by a.

The input $a = 0$, $b = 2$, $c = 5$ satisfies $ab = ac$, since both products are zero, but $b \neq c$. It is a counterexample. The division step failed because the hypotheses did not exclude $a = 0$.

With the added hypothesis $a \neq 0$, the statement is true: from $ab = ac$, division by the nonzero number a gives $b = c$. The repaired proof establishes the claim with that extra hypothesis; it does not prove the original unrestricted claim.

Look for the exact definition or earlier result that supports each step. An expression of the form $2t$ establishes evenness only after we know $t \in \mathbb{Z}$. A contradiction must identify incompatible facts. A proof about every input must avoid extra restrictions on that input.

(Non-examinable) An irrationality proof

This enrichment example combines the even-square result with familiar fraction arithmetic. The example and its terminology are non-examinable.

A real number is **rational** if it can be written as a/b for integers a, b with $b \neq 0$; it is **irrational** if no such fraction exists. A fraction is in **lowest terms** when no integer greater than 1 divides both numerator and denominator. We use the elementary fact that any rational number can be written in lowest terms by cancelling common factors. Choose a positive denominator.

The number $\sqrt{2}$ is the positive real number whose square is 2. We will prove that it is irrational.

Proof. Suppose, for contradiction, that $\sqrt{2} = a/b$, where $a, b \in \mathbb{Z}$, $b > 0$, and the fraction is in lowest terms. Squaring and multiplying by b^2 gives

$$a^2 = 2b^2.$$

Thus a^2 is even. Example 3.11 shows that a is even, so $a = 2k$ for some $k \in \mathbb{Z}$. Substitution gives

$$4k^2 = 2b^2, \qquad \text{hence} \qquad b^2 = 2k^2.$$

Now b^2 is even, so the same result shows that b is even. The integer 2 therefore divides both a and b, contradicting the choice of a fraction in lowest terms. Hence $\sqrt{2}$ is irrational. □

The contradiction is with the chosen representation being in lowest terms, not with fractions being allowed to have even numerators and denominators. A fraction such as $2/4$ is perfectly valid; it simply is not in lowest terms.

3.7 Exercises

3.1. Justify each statement directly from its definition. Write the required integer explicitly, rather than giving only the statement's truth value.

a) -8 is even.

b) -11 is odd.

c) $7 \mid 0$.

d) $-3 \mid 12$.

3.2. For each claim, write the opening assumptions and say what remains to be shown using the indicated method. Do not complete the proof yet.

a) For all integers a, b, if a is odd and b is even, then $a + b$ is odd. Use a direct proof.

b) For every integer n, if $5n + 4$ is odd, then n is odd. Use a proof by contrapositive.

c) There are no integers a, b for which $4a + 1 = 2b$. Use a proof by contradiction.

3.3. Fill each gap and explain why the integer used in the conclusion really is an integer.

Let a, b be arbitrary integers, with a odd and b even. There are integers r, s such that $a =$ _____ and $b =$ _____. Therefore

$$a + b = _____ = 2(r + s) + 1.$$

Since _____ $\in \mathbb{Z}$, the definition of _____ shows that $a + b$ is odd.

3.4. Consider this argument.

Let a, b be arbitrary odd integers. Write $a = 2r + 1$ and $b = 2s + 1$ for some integers r, s. Then

$$ab = (2r + 1)(2s + 1) = 2(2rs + r + s) + 1.$$

The number $2rs + r + s$ is an integer, so ab is odd.

a) Which definition justifies the first representations of a, b?

b) Why is $2rs + r + s$ an integer?

c) Which definition justifies the conclusion?

d) Why does the proof apply to all odd integers, including negative ones?

3.5. Prove each claim. State the domain of your arbitrary input and justify the final step.

a) For every real number x, if $x > -3$, then $4x + 5 > -7$.

b) For every integer n, if n is even, then $n^2 + 3$ is odd.

c) For every integer n, $(n + 1)^2 - n^2$ is odd.

3.6. Prove that for every integer n there is an integer m such that $m > n$ and $3 \mid (m - n)$. Give a formula for m in terms of n, then check both required properties.

3.7. Let d, a, b be integers with $d \neq 0$. Complete this proof of the claim that if $d \mid a$ and $d \mid b$, then $d^2 \mid ab$.

Suppose $d \mid a$ and $d \mid b$. There are integers r, s with $a =$ _____ and $b =$ _____. Hence

$$ab = _____ = d^2(rs).$$

Here $d^2 \neq 0$ because _____, and $rs \in \mathbb{Z}$ because _____. Thus $d^2 \mid ab$.

3.8. Prove each claim directly from the definition of divisibility.

a) For integers d, a with $d \neq 0$, if $d \mid a$, then $d \mid (a^2 - a)$.

b) For integers a, b with $a \neq 0$, if $a \mid b$, then $a \mid (b + 3a)$.

3.9. a) Prove that for every integer n, if $8 \mid n$, then $4 \mid n$.

b) State the converse and give a counterexample to it. Check explicitly that your example satisfies the converse's hypothesis and fails its conclusion.

3.10. A student writes:

> For $n = -1, 0, 2$, the expression $10n + 5$ gives $-5, 5, 25$. All three are divisible by 5. Therefore $5 \mid (10n + 5)$ for every integer n.

a) Are the calculations correct? Do they prove the universal claim? Explain the gap in the reasoning.

b) Replace the argument with a direct proof of the universal claim.

3.11. A proposed proof says:

> Let a, b be odd integers. Then $a = 2k+1$ and $b = 2k+1$ for some integer k. Hence $a - b = 0$, which is even.

a) Explain what extra assumption is introduced by using the same k. Give two odd integers for which those two representations cannot hold with the same k.

b) Prove the intended claim: the difference of any two odd integers is even.

3.12. A student proposes:

> If $3 \mid n$, write $n = 3k$ for some integer k. Then $n = 6(k/2)$, so $6 \mid n$.

a) Identify the missing requirement in the last step.

b) Give a counterexample to the proposed claim. In your example, what is $k/2$?

c) Prove the repaired claim: for every integer n, if $3 \mid n$, then $6 \mid 2n$.

3.13. A student offers the following as a proof of the even-square claim, without using that claim as an earlier result:

> Let n be an integer and suppose n^2 is even. Since n is even, there is an integer k with $n = 2k$. Therefore n is even.

a) Identify the unsupported assertion. Explain why the final sentence has not been established by this argument.

b) Write a valid proof by contrapositive. You may use the previously proved fact that the square of an odd integer is odd.

3.14. Prove that for every integer n, if $5n + 4$ is odd, then n is odd.

a) State the contrapositive in words, using the fact that every integer has exactly one parity.

b) Complete the calculation for an arbitrary even integer $n = 2k$:

$$5n + 4 = _____ = 2(_____).$$

c) Explain why the calculation proves the contrapositive and hence the original claim.

3.15. Prove by contrapositive that for all integers a, b, if ab is odd, then a and b are both odd. State the contrapositive before proving it. Remember that the negation of "both are odd" is "at least one is even" for integer inputs.

3.16. Prove by contrapositive that for every integer n, if $n^2 + 2n$ is odd, then n is odd. Start with an arbitrary integer satisfying the contrapositive's hypothesis.

3.17. Prove by contradiction that there are no integers a, b satisfying $4a + 1 = 2b$. Write the assumption that begins the contradiction proof, identify the incompatible properties it forces on one integer, and state the conclusion.

3.18. Prove by contradiction that there is no smallest integer divisible by 4. If M were such a smallest integer, construct another integer divisible by 4 that is smaller than M. Your argument must also work when M is negative.

3.19. Prove by contradiction that there are no integers a, b for which $a + b$ is odd and $a - b$ is even. You may use previously proved parity facts. Identify the fact used at each step.

3.20. For each claim, choose direct proof, contrapositive or contradiction. Give a reason for your choice, the opening assumption or assumptions, and one useful next step. More than one method may work; a full proof is not required here.

a) For integers d, a, b with $d \neq 0$, if $d \mid a$ and $d \mid b$, then $d \mid (2a - 5b)$.

b) For every integer n, if $n^2 + 1$ is even, then n is odd.

c) There is no greatest integer divisible by 7.

3.21. Each claim below is false. Give a counterexample and verify both that its hypotheses hold and that its conclusion fails.

a) For all integers a, b, if ab is even, then a and b are both even.

b) For all integers d, n with $d \neq 0$, if $d \mid n^2$, then $d \mid n$.

c) For all nonzero integers a, b, if $a \mid b$ and $b \mid a$, then $a = b$.

Repair part (a) by replacing "both" with "at least one", and prove the repaired claim. You may use Exercise 3.4.

3.22. Prove that for all integers a, b, the sum $a + b$ is odd if and only if a and b have different parity: one is even and the other is odd. Write both directions of the proof. You may use earlier parity results, but name the result whenever you use it.

LECTURE NOTES
DISCRETE MATHEMATICS

CHAPTER 4

Sets and mappings



Updated: 19 November 2026

4 Sets and mappings

Suppose a system replaces its old list of users permitted to enter a laboratory with a new list. What would establish that nobody loses access? Every user on the old list would have to appear on the new one. What would establish that access is unchanged? We would also have to check that the new list admits nobody who was absent from the old one.

These requirements already have the logical forms studied in Chapters 1 and 2: a claim about every user, and two directions when we want an exact match. Chapter 3 supplied methods for proving such claims. We now apply those methods to sets, functions and relations.

The objects change, but the starting question remains useful: what does the definition require us to show? A statement about one set containing another becomes an argument about an arbitrary element. A statement about distinct inputs becomes an argument about a function's outputs. A statement about a relation becomes an argument about the pairs it contains. We will develop each translation before using it in a proof.

What you should be able to do

After working through this chapter, you should be able to:

- interpret subsets, set equality, union, intersection and difference;
- prove a set inclusion by following an arbitrary element;
- establish set equality by proving both inclusions;
- prove or refute injectivity using the stated function domain;
- calculate images of subsets and justify claims about their elements;
- prove transitivity or give a complete counterexample to it;
- identify and repair arguments that use the wrong direction or an unjustified assumption.

4.1 Subsets and set operations

Chapter 1 introduced $x \in A$: one object belongs to a set. We now need notation for a claim about every element of one set.

❄ Definition 4.1 — Subsets and the empty set

A set A is a **subset** of a set B, written $A \subseteq B$, if every element of A is also an element of B. Thus

$$A \subseteq B \quad \text{means} \quad \text{for every } x, \text{ if } x \in A, \text{ then } x \in B.$$

Write $A \not\subseteq B$ if this fails. A **proper subset**, written $A \subsetneq B$, satisfies $A \subseteq B$ and $A \neq B$. The **empty set**, written $\varnothing$, has no elements.

The symbol $\subseteq$ allows equality: $A \subseteq A$ because an element of A is certainly an element of A. To refute $A \subseteq B$, we need one object that belongs to A and does not belong to B. An object outside both sets does not refute this claim.

Example 4.1. Suppose Alice may read exactly the files in $A = \{\text{guide}, \text{report}\}$, and Bob may read exactly the files in $B = \{\text{guide}, \text{report}, \text{data}\}$. Every file in A is in B, so $A \subseteq B$. The file data belongs to B but not A, so $B \not\subseteq A$ and $A \neq B$. Consequently, $A \subsetneq B$.

Here $\text{report} \in A$ says that one file belongs to Alice's permission set. The statement $\{\text{report}\} \subseteq A$ says that every element of the one-file set belongs to A. These statements are both true, but their left-hand objects are different: a file in the first, a set of files in the second.

Recall that two sets are equal when they have exactly the same elements. Therefore

$$A = B \quad \text{if and only if} \quad A \subseteq B \text{ and } B \subseteq A.$$

The first inclusion prevents A from having an element missing from B. The second prevents B from having an element missing from A. Equal numbers of elements alone do not establish either inclusion.

❄ Definition 4.2 — Union, intersection and difference

For sets A and B:

- $A \cup B$, their **union**, contains the objects in A or B, including those in both;
- $A \cap B$, their **intersection**, contains the objects in both A and B;
- $A \setminus B$, their **difference**, contains the objects in A that are not in B.

Equivalently,

$$\begin{aligned} x \in A \cup B &\iff x \in A \text{ or } x \in B, \\ x \in A \cap B &\iff x \in A \text{ and } x \in B, \\ x \in A \setminus B &\iff x \in A \text{ and } x \notin B. \end{aligned}$$

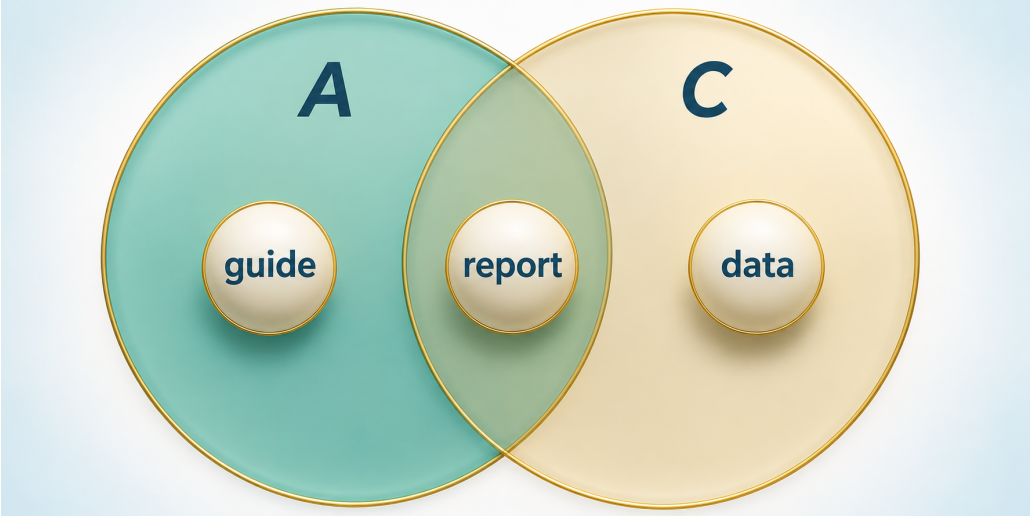
Each $\iff$ reads "if and only if".

These definitions connect set notation to the connectives of Chapter 2. Membership in a union uses inclusive "or"; membership in an intersection requires both conditions. In $A \setminus B$, the order of the sets specifies which elements to retain and which to remove.

Example 4.2. Let $A = \{\text{guide}, \text{report}\}$ and $C = \{\text{report}, \text{data}\}$. Then

$$A \cup C = \{\text{guide}, \text{report}, \text{data}\},$$
$$A \cap C = \{\text{report}\},$$
$$A \setminus C = \{\text{guide}\}, \qquad C \setminus A = \{\text{data}\}.$$

For instance, guide belongs to $A \setminus C$ because it belongs to A and not to C. Report is excluded from that difference because it is in C, even though it is also in A. These two differences are not equal. Also, $A \setminus A = \emptyset$: no file can both belong and fail to belong to the same set.



Report belongs to both sets; guide belongs only to A, and data only to C.

Python (non-examinable)

Python can carry out these operations on the same finite sets. Braces enclose the sets, and quotation marks enclose the file names. The operators `|`, `&` and `-` give union, intersection and difference, respectively.

```
A = {"guide", "report"}
C = {"report", "data"}
print(sorted(A | C))
print(sorted(A & C))
print(sorted(A - C))
print(sorted(C - A))
```

Output.

```
['data', 'guide', 'report']
['report']
['guide']
['data']
```

Python sets have no required display order. The function `sorted` makes an alphabetical list for a fixed display; its square brackets denote that list. The four lines match the union, intersection and two differences above.

4.2 Set inclusion

To prove $A \subseteq B$, begin with an arbitrary element of A and show that it belongs to B. The definition of A supplies the hypothesis; the definition of B tells us what the conclusion needs. This is the direct-proof method of Chapter 3 applied to membership. Such an argument is often called **element chasing**.

Choosing one convenient element, such as guide, would show only what happens to that element. Writing "Let $x \in A$" leaves x arbitrary: the argument must use only its stated membership and the hypotheses.

Example 4.3. Prove that $A \cap B \subseteq A$ for any sets A, B.

Proof. Let $x \in A \cap B$. By the definition of intersection, $x \in A$ and $x \in B$. In particular, $x \in A$. Thus every element of $A \cap B$ belongs to A, so $A \cap B \subseteq A$. □

The same argument gives $A \cap B \subseteq B$ by retaining the other membership statement. We do not have to show that every element of A lies in the intersection; that would be the reverse inclusion.

The inclusion $\emptyset \subseteq B$ holds for every set B. A counterexample would have to belong to $\emptyset$, which is impossible. In the implication "if $x \in \emptyset$, then $x \in B$", the antecedent is false for every x, so there is no failed case. Likewise, the proof above does not require $A \cap B$ to be nonempty. It explains what follows if an element belongs to that intersection.

Example 4.4. Suppose $A \subseteq B$ and $B \subseteq C$. Prove that $A \subseteq C$.

Proof. Take an arbitrary $x \in A$. The inclusion $A \subseteq B$ gives $x \in B$. The inclusion $B \subseteq C$ then gives $x \in C$. Hence $A \subseteq C$. □

Both hypotheses concern the same element x. We have followed it through the sets rather than selected different elements at each step. The argument has the chaining form from Chapter 2.

Example 4.5. Suppose $A \subseteq B$. Prove that $A \cup C \subseteq B \cup C$.

Proof. Let $x \in A \cup C$. Then $x \in A$ or $x \in C$. If $x \in A$, the hypothesis gives $x \in B$, so $x \in B \cup C$. If $x \in C$, we have $x \in B \cup C$ directly. The alternatives cover every element of $A \cup C$, including an element that belongs to both sets. Therefore $A \cup C \subseteq B \cup C$. □

We have used a short **proof by cases**: each permitted alternative leads to the required conclusion. The alternatives need not be mutually exclusive. It would be wrong to begin by assuming that an element of the union belongs to both sets.

Example 4.6. Define

$$A = \{n \in \mathbb{Z} : 6 \mid n\}, \qquad B = \{n \in \mathbb{Z} : 3 \mid n\}.$$

Prove that $A \subseteq B$.

Proof. Let $n \in A$. Then $n \in \mathbb{Z}$ and $6 \mid n$. By the definition of divisibility, $n = 6k$ for some $k \in \mathbb{Z}$. Consequently $n = 3(2k)$. Since $2k$ is an integer, $3 \mid n$, so $n \in B$. Hence $A \subseteq B$. □

The set notation has packaged the claim "every multiple of 6 is a multiple of 3". The algebra and integer witness are the same tools used in Chapter 3.

4.3 Set equality

For $A = B$, one inclusion is only half the task. The other inclusion may use different hypotheses or need a different argument. State the direction being proved so that a reader can check both.

Example 4.7. Suppose $A \subseteq B$. Prove that $A \cap B = A$.

Proof. First, $A \cap B \subseteq A$ by Example 4.3.

For the other inclusion, let $x \in A$. Since $A \subseteq B$, we also have $x \in B$. Thus x belongs to both sets, giving $x \in A \cap B$. Therefore $A \subseteq A \cap B$. Both inclusions hold, so $A \cap B = A$. □

The first direction needs no assumption about $A \subseteq B$. The second does: without it, an element of A might be missing from B.

Example 4.8. Prove that for any sets A, B, C,

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C).$$

An object belongs to the left side when it belongs to A and to at least one of B, C. This suggests the distributive law for "and" and "or" from Chapter 2.

Proof. Let x be arbitrary. Unpacking the definitions gives

$$\begin{aligned} x \in A \cap (B \cup C) &\iff (x \in A) \text{ and } (x \in B \text{ or } x \in C) \\ &\iff (x \in A \text{ and } x \in B) \text{ or } (x \in A \text{ and } x \in C) \\ &\iff x \in (A \cap B) \cup (A \cap C). \end{aligned}$$

The middle step is logical distributivity. Every step is reversible, so an element belongs to either side exactly when it belongs to the other. The two sets are equal. □

This chain establishes both inclusions at once. Reading it forwards gives the left-to-right inclusion; reading it backwards gives the right-to-left inclusion. A chain of one-way implications would require separately checking the missing direction.

Example 4.9. A proposed identity is

$$(A \cup B) \setminus C = (A \setminus C) \cup B.$$

Take $A = \{\text{guide}\}$ and $B = C = \{\text{report}\}$. Then

$$(A \cup B) \setminus C = \{\text{guide}\}, \qquad (A \setminus C) \cup B = \{\text{guide}, \text{report}\}.$$

Report is a counterexample to equality: it belongs to the right side and not to the left. On the left, removing C affects elements from both alternatives. On the right, the untouched set B puts report back into the result. A general identity must hold for every permitted choice of sets; this one choice refutes it.

4.4 Injectivity

Recall that a function $f : X \to Y$ assigns each input in its domain X exactly one output in its codomain Y. Several inputs may share an output without violating this definition. Injectivity imposes an additional requirement.

❄ Definition 4.3 — Injectivity

A function $f : X \to Y$ is **injective**, or **one-to-one**, if distinct inputs always have distinct outputs:

$$\text{for all } a, b \in X, \quad a \neq b \implies f(a) \neq f(b).$$

Equivalently, using the contrapositive,

$$\text{for all } a, b \in X, \quad f(a) = f(b) \implies a = b.$$

To refute injectivity, exhibit two distinct elements of X with the same output.

Example 4.10. Identifiers and printers. Let $U = \{\text{Alice}, \text{Bob}, \text{Charlie}\}$, $I = \{104, 108, 112, 116\}$ and $P = \{\text{Printer 1}, \text{Printer 2}, \text{Printer 3}\}$. Define an identifier function $i : U \to I$ and a default-printer function $d : U \to P$ by the following complete table.

User u	Alice	Bob	Charlie
Identifier $i(u)$	104	108	112
Default printer $d(u)$	Printer 1	Printer 1	Printer 2

Both assignments are functions: each user has exactly one permitted output. The identifier function is injective because its three outputs are distinct. The unused identifier 116 does not prevent injectivity. The printer function is not injective because Alice and Bob are distinct users but $d(\text{Alice}) = d(\text{Bob}) = \text{Printer 1}$.

For a function described by a formula on an infinite domain, a short list of distinct outputs does not settle injectivity. A useful opening is "Let a, b belong to the domain and suppose $f(a) = f(b)$". Use the function rule to show that this forces $a = b$. We do not assume that a, b are distinct in this version of the proof.

Example 4.11. Define $f : \mathbb{Z} \to \mathbb{Z}$ by $f(n) = 4n - 3$. The output is an integer for every integer input, so this is a function with the stated domain and codomain. Prove that it is injective.

Proof. Let $a, b \in \mathbb{Z}$ and suppose $f(a) = f(b)$. Then $4a - 3 = 4b - 3$. Adding 3 and dividing by the nonzero number 4 gives $a = b$. Thus equal outputs require equal inputs, so f is injective. □

The equality $f(a) = f(b)$ was the starting assumption; $a = b$ was the conclusion. Reversing those roles would not prove injectivity.

Example 4.12. The function $q : \mathbb{Z} \to \mathbb{Z}$ given by $q(n) = n^2$ is not injective: $-3 \neq 3$, but $q(-3) = q(3) = 9$.

Now keep the same squaring rule but restrict its domain to $D = \{0, 1, 2, 3\}$. The outputs are $0, 1, 4, 9$, respectively. All are distinct, and every input in D has been considered. Thus the function $q_D : D \to \mathbb{Z}$, $q_D(n) = n^2$, is injective. The earlier pair $-3, 3$ cannot refute this statement because $-3 \notin D$. This finite check establishes injectivity on D; it does not check every input in $\mathbb{N}$ or $\mathbb{Z}$.

4.5 Images of subsets

A function sends individual inputs to outputs, but we can also collect the outputs obtained from a whole set of inputs. This produces a new set to which our inclusion and equality methods apply.

❄ Definition 4.4 — Image of a subset

Let $f : X \to Y$ and $A \subseteq X$. The **image** of A under f is

$$f(A) = \{f(a) : a \in A\}.$$

Thus $f(A) \subseteq Y$, and

$$y \in f(A) \iff \text{there is an } a \in A \text{ such that } f(a) = y.$$

Here a is an element of X, while A is a subset of X. Thus $f(a)$ is one output and $f(A)$ is the set of outputs obtained from A.

Example 4.13. Use the printer function d from Example 4.10. For $A = \{\text{Alice}, \text{Bob}\}$,

$$d(A) = \{d(\text{Alice}), d(\text{Bob})\} = \{\text{Printer 1}\}.$$

The image is a set, so the shared output is listed only once. For $B = \{\text{Alice}, \text{Charlie}\}$, the image is $d(B) = \{\text{Printer 1}, \text{Printer 2}\}$. No input is sent to Printer 3, although it is in the codomain P. The image of the full domain, $d(U)$, is the range recalled in Chapter 1.

For any function, $f(\emptyset) = \emptyset$: there is no input in the empty set that could supply an output for its image.

Example 4.14. Let $f : X \to Y$ and $A, B \subseteq X$. Prove that if $A \subseteq B$, then $f(A) \subseteq f(B)$.

An element of $f(A)$ is an output. To use the hypothesis $A \subseteq B$, we must identify an input in A that produced it.

Proof. Assume $A \subseteq B$ and take $y \in f(A)$. The definition of image supplies an $a \in A$ with $y = f(a)$. Since $A \subseteq B$, this same input a belongs to B. It therefore supplies $y = f(a) \in f(B)$. We have proved $f(A) \subseteq f(B)$. □

We did not need f to be injective. The output only needed one input witness in the larger set.

Example 4.15. For $f : X \to Y$ and $A, B \subseteq X$, prove that

$$f(A \cup B) = f(A) \cup f(B).$$

Proof. First let $y \in f(A \cup B)$. There is an $x \in A \cup B$ with $f(x) = y$. If $x \in A$, then $y \in f(A)$; if $x \in B$, then $y \in f(B)$. In either case, $y \in f(A) \cup f(B)$, proving the first inclusion.

Conversely, let $y \in f(A) \cup f(B)$. If $y \in f(A)$, choose $a \in A$ with $f(a) = y$. Since $a \in A \cup B$, we have $y \in f(A \cup B)$. If $y \in f(B)$, an input $b \in B$ with $f(b) = y$ gives the same conclusion. This proves the reverse inclusion, hence the equality. □

Each inclusion starts with an output and finds an appropriate input. The "or" in the union lets either input-set alternative do the work.

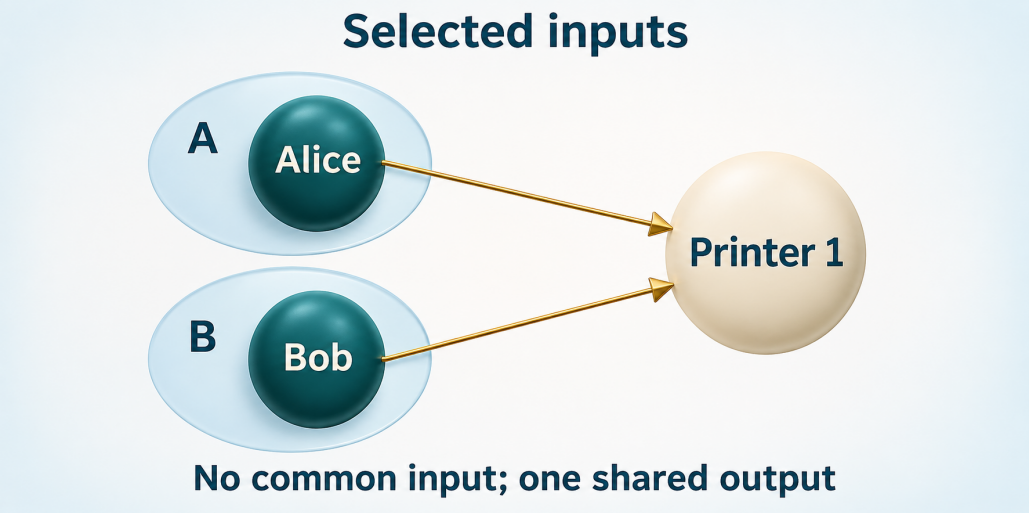
Example 4.16. It is tempting to replace union by intersection in the last equality. For an arbitrary function, the resulting equality can fail. Using the printer function again, take $A = \{\text{Alice}\}$ and $B = \{\text{Bob}\}$. Then

$$A \cap B = \emptyset, \qquad d(A \cap B) = \emptyset,$$

but

$$d(A) \cap d(B) = \{\text{Printer 1}\}.$$

The output is shared, but the two input sets have no shared user.



A and *B* are disjoint, but their images share Printer 1.

There is always an inclusion in one direction:

$$f(A \cap B) \subseteq f(A) \cap f(B).$$

Proof. If $y \in f(A \cap B)$, choose $x \in A \cap B$ with $f(x) = y$. Then $x \in A$ and $x \in B$, so this same x witnesses both $y \in f(A)$ and $y \in f(B)$. Therefore $y \in f(A) \cap f(B)$, as required. □

For the reverse direction, $y \in f(A)$ and $y \in f(B)$ may have different input witnesses. We cannot assume they are one shared input. Alice and Bob demonstrate the obstruction.

4.6 Relations and transitivity

Chapter 1 used predicates with two inputs, such as "user u may access file f". We now consider relationships between objects from one specified set. Recording which pairs satisfy a predicate gives a set of ordered pairs.

An **ordered pair** (a, b) records a first entry and a second entry. Two ordered pairs are equal exactly when their first entries agree and their second entries agree. In particular, (Alice, Bob) and (Bob, Alice) are different. Braces would instead describe the unordered set {Alice, Bob}.

❄ Definition 4.5 — Relation on a set

A **relation** R on a set S is a set of ordered pairs (a, b) with $a, b \in S$. We write

$$aRb \quad \text{or} \quad (a, b) \in R$$

to say that the pair belongs to the relation. A condition on two inputs defines a relation by specifying exactly which pairs it contains.

For instance, on $\mathbb{Z}$ we could define aRb to mean $a < b$. Then $2R5$ holds and $5R2$ does not. A relation need not assign exactly one second entry to each first entry: one integer is less than many others. The exactly-one-output requirement belongs to functions, not to arbitrary relations.

❄ Definition 4.6 — Transitivity

A relation R on S is **transitive** if, for all $x, y, z \in S$,

$$(xRy \text{ and } yRz) \implies xRz.$$

The elements x, y, z need not be distinct.

To prove transitivity, take arbitrary $x, y, z \in S$, assume both xRy and yRz, and deduce xRz using the relation's definition. The middle element must be the same in the two premises. To refute transitivity, find allowed x, y, z for which xRy and yRz hold but xRz fails.

Example 4.17. Let $S = \{\text{Alice}, \text{Bob}, \text{Charlie}\}$ and

$$R = \{(\text{Alice}, \text{Bob}), (\text{Bob}, \text{Charlie})\}.$$

These are all the pairs in R. The relation is not transitive: Alice R Bob and Bob R Charlie hold, but Alice R Charlie does not. The ordered pair (Alice, Charlie) is absent. The triple Alice, Bob, Charlie therefore makes both premises true and the conclusion false.

We did not refute transitivity merely by pointing to an absent pair. The two present pairs provide the premises that would require it.

Example 4.18. Let $d : U \to P$ be any fixed default-printer function. Define a relation R on U by

$$uRv \quad \iff \quad d(u) = d(v).$$

Prove that R is transitive.

Proof. Let $u, v, w \in U$ and suppose uRv and vRw. Unpacking the definition gives $d(u) = d(v)$ and $d(v) = d(w)$. Equality then gives $d(u) = d(w)$, which is precisely uRw. Thus R is transitive. □

The function is fixed throughout the argument, so the two occurrences of $d(v)$ refer to the same output. No table of particular printer assignments was needed.

Example 4.19. On $\mathbb{Z}$, define aRb to mean $3 \mid (a-b)$. Prove that this relation is transitive.

Proof. Let $a, b, c \in \mathbb{Z}$ and assume aRb and bRc. There are integers r, s such that $a-b = 3r$ and $b-c = 3s$. Adding these equations gives

$$a - c = (a-b) + (b-c) = 3r + 3s = 3(r+s).$$

Since $r+s \in \mathbb{Z}$, we have $3 \mid (a-c)$, so aRc. Therefore R is transitive. □

The definition of the relation tells us which arithmetic claim to prove. Chapter 3's integer-witness method supplies the proof.

Example 4.20. On $\mathbb{Z}$, define aTb to mean $b = a+1$. We have $0T1$ and $1T2$, since $1 = 0+1$ and $2 = 1+1$. However, $0T2$ is false: $2 \neq 0+1$. Thus T is not transitive.

Substituting the two premises into the rule gives $c = b+1 = (a+1)+1 = a+2$. That calculation does not give the required conclusion $c = a+1$. Following a two-step chain is insufficient unless it yields exactly what the definition requires.

4.7 Proof checking

Before checking individual algebraic steps, identify the kind of claim being proved. Its definition determines the opening assumptions and the required endpoint.

Claim	Starting point	Required conclusion
$A \subseteq B$	Take $x \in A$.	Show $x \in B$.
$A = B$	Prove an inclusion in each direction.	Account for every element of both sets.
f is injective	Take a, b in the domain with $f(a) = f(b)$.	Show $a = b$.
R is transitive	Take $x, y, z \in S$ with xRy and yRz.	Show xRz.

For an image claim, also track which symbols name inputs and which name outputs. Membership $y \in f(A)$ supplies an input witness $a \in A$ with $f(a) = y$. Two uses of that definition need not supply the same input.

Example 4.21. Faith argues that $A \cap B = A$ for all sets A, B because an element of $A \cap B$ belongs to A. Her argument correctly proves $A \cap B \subseteq A$. It does not prove $A \subseteq A \cap B$.

For $A = \{1, 2\}$ and $B = \{2\}$, the intersection is $\{2\}$, which is not A. The element $1 \in A$ is missing from the intersection. If the hypothesis $A \subseteq B$ were added, the reverse inclusion would follow as in Section 4.3. Adding that hypothesis would change the claim, so it must be stated.

Example 4.22. Grace proposes that $q : \mathbb{Z} \to \mathbb{Z}$, $q(n) = n^2$, is injective because $a = b$ implies $a^2 = b^2$. Her implication is true, but it starts with equal inputs. The injectivity criterion starts with equal outputs and asks whether the inputs must agree.

The distinct inputs $-3, 3$ already refute the claim for this domain. Indeed, every function assigns the same output to two occurrences of the same input. That fact alone cannot establish injectivity.

Example 4.23. Let $S = \{\text{Alice}, \text{Bob}, \text{Charlie}\}$ and $R = \{(\text{Alice}, \text{Bob})\}$. Mike says R is not transitive because $(\text{Alice}, \text{Charlie}) \notin R$.

His proposed counterexample lacks the second premise: Bob R Charlie is false. In fact, this R is transitive. For xRy to hold, we must have $x = \text{Alice}$ and $y = \text{Bob}$. But no pair in R begins with Bob, so yRz cannot then hold. There is no triple with both premises true, and hence no triple violating the transitivity implication.

4.8 Exercises

4.1. Let $A = \{1, 3, 5\}$ and $B = \{1, 2, 3, 5\}$. Decide each statement. For a failed inclusion, give an element that refutes it.

a) $3 \in A$.

b) $\{3\} \subseteq A$.

c) $A \subseteq B$.

d) $B \subseteq A$.

e) $A \subsetneq B$.

f) $\emptyset \subseteq A$.

4.2. Let $A = \{0, 2, 4\}$ and $B = \{2, 3, 4\}$. List the elements of each set.

a) $A \cup B$.

b) $A \cap B$.

c) $A \setminus B$.

d) $B \setminus A$.

e) $A \cap \emptyset$.

f) $A \cup \emptyset$.

4.3. Let $A, B \subseteq S$. Write each claim using quantifiers over S and membership statements. For equality, write both required directions.

a) $A \subseteq B$.

b) $A = B$.

c) $A \nsubseteq B$.

4.4. Fill the gaps to prove $A \cap B \subseteq B$ for arbitrary sets A, B.

> Let $x \in$ _____. By the definition of intersection, $x \in$ _____ and $x \in$ _____. In particular, $x \in$ _____. Since x was arbitrary, _____.

Explain why selecting only one particular element would not establish the claim for arbitrary sets.

4.5. Prove each claim for arbitrary sets A, B.

a) $A \subseteq A \cup B$.

b) $A \setminus B \subseteq A$.

4.6. Let $A = \{n \in \mathbb{Z} : 10 \mid n\}$ and $B = \{n \in \mathbb{Z} : 5 \mid n\}$.

a) Prove $A \subseteq B$ using an integer witness.

b) Prove that the inclusion is proper by finding an element of B that is not in A.

4.7. Prove that if $A \subseteq B$, then $A \cap C \subseteq B \cap C$. Identify where you use the hypothesis $A \subseteq B$.

4.8. Suppose $A \subseteq B$. Prove $A \cup B = B$. State separately the two inclusions you establish.

4.9. Prove that

$$A \setminus (B \cup C) = (A \setminus B) \cap (A \setminus C)$$

for arbitrary sets A, B, C. Either prove both inclusions or use a chain of reversible membership conditions. Explain the logical law used when a union is excluded.

4.10. A student proves $A \cap B \subseteq A$ and $A \cap B \subseteq B$, then concludes $A = B$.

a) Explain why those are not the two inclusions needed for $A = B$.

b) Give explicit sets for which both proved inclusions are true but $A \neq B$.

4.11. Let $D = \{\text{Eve}, \text{Faith}, \text{Grace}\}$, $C = \{10, 20, 30, 40\}$ and $P = \{1, 2, 3\}$. Define $f : D \to C$ and $g : D \to P$ by the complete table.

Input	Eve	Faith	Grace
f output	10	30	20
g output	2	2	1

a) Which functions are injective? Justify each answer.

b) Find $f(\{\text{Eve}, \text{Grace}\})$ and $g(\{\text{Eve}, \text{Faith}\})$.

c) Does the unused output 40 prevent f from being injective? Explain from the definition.

4.12. Prove that each function is injective. State the domain of your arbitrary inputs and justify the algebra.

a) $f : \mathbb{Z} \to \mathbb{Z}$, $f(n) = 5n + 4$.

b) $h : \mathbb{R} \to \mathbb{R}$, $h(x) = 1 - 2x$.

4.13. Let $f : \mathbb{Z} \to \mathbb{Z}$ be given by $f(n) = n(n - 1)$.

a) Find two distinct integer inputs with the same output.

b) Keep the same rule but use the domain $D = \{1, 2, 3\}$. List every output and decide whether $f_D : D \to \mathbb{Z}$ is injective.

c) Why must a counterexample to injectivity use inputs in the stated domain?

4.14. Let $q : \mathbb{Z} \to \mathbb{Z}$, $q(n) = n^2$, and take $A = \{-2, 0, 1\}$ and $B = \{0, 2\}$. Find each set.

a) $q(A)$.

b) $q(B)$.

c) $q(A \cap B)$.

d) $q(A) \cap q(B)$.

e) $q(A \cup B)$.

f) $q(A) \cup q(B)$.

Compare the sets in parts (c) and (d), then those in parts (e) and (f). For each comparison, state whether the sets are equal or one is a proper subset of the other.

4.15. For $q : \mathbb{Z} \to \mathbb{Z}$, $q(n) = n^2$, construct nonempty finite sets $A, B \subseteq \mathbb{Z}$ such that $q(A) \subseteq q(B)$ but $A \not\subseteq B$. Verify both conditions explicitly.

4.16. Let $f : X \to Y$ and $A, B, C \subseteq X$. Suppose $A \subseteq C$ and $B \subseteq C$. Prove that $f(A) \cup f(B) \subseteq f(C)$. Begin with an arbitrary output in the left-hand union and account for both alternatives.

4.17. The proposed identity is $f(A \setminus B) = f(A) \setminus f(B)$ for every function and every two subsets of its domain. Refute it using $q : \mathbb{Z} \to \mathbb{Z}$, $q(n) = n^2$, and suitably chosen nonempty finite sets A, B. Calculate both sides.

4.18. On $S = \{\text{Alice}, \text{Bob}, \text{Charlie}\}$, let

$$R = \{(\text{Alice}, \text{Bob}), (\text{Bob}, \text{Charlie}), (\text{Alice}, \text{Charlie})\}.$$

These are all its pairs.

a) Decide whether Bob R Alice and Alice R Charlie hold.

b) Prove that R is transitive by identifying every possible two-step chain and checking its required conclusion.

c) Remove (Alice, Charlie) and call the new relation T. Give a complete counterexample to transitivity of T.

4.19. On $S = \{\text{Alice}, \text{Bob}\}$, let $R = \{(\text{Alice}, \text{Bob}), (\text{Bob}, \text{Alice})\}$. Show that R is not transitive. State a triple and check all three facts needed for the counterexample.

4.20. For each relation on $\mathbb{Z}$, either prove transitivity or give a triple making both premises true and the conclusion false.

a) aRb means $a \leq b$.

b) aTb means $b = a + 2$.

4.21. Let S be a set of tasks and let $p : S \to \mathbb{Z}$ assign each task a fixed numerical priority. Define xRy to mean $p(x) \leq p(y)$. Prove that R is transitive, identifying the numerical fact used after unpacking its two premises.

4.22. Let S be the positive integers, and define aRb to mean $a \mid b$. Prove that R is transitive. You may adapt the integer-witness argument from Chapter 3, but write the argument for the relation and check why its divisors meet the nonzero requirement.

4.23. For $f : \mathbb{Z} \to \mathbb{Z}$, $f(n) = 2n + 5$, a proposed injectivity proof says:

> Let $a, b \in \mathbb{Z}$ and suppose $a = b$. Then $2a + 5 = 2b + 5$, so $f(a) = f(b)$. Therefore f is injective.

a) Identify the direction proved and the direction required.

b) Replace the argument with a valid injectivity proof.

4.24. On $S = \{0, 1, 2\}$, let $R = \{(0, 0), (0, 1), (1, 1)\}$. A student says R is not transitive because $(1, 0) \notin R$.

a) Explain what additional facts a counterexample involving the missing pair $(1, 0)$ would need.

b) Decide whether this R is transitive, giving a complete argument from its listed pairs.

4.25. Prove that for any sets A, B,

$$A \subseteq B \quad \Longleftrightarrow \quad A \setminus B = \emptyset.$$

Write both directions. In the direction starting from the empty difference, explain why an element of A cannot be outside B.

LECTURE NOTES
DISCRETE MATHEMATICS

CHAPTER 5

Induction



Updated: 19 November 2026

5 Induction

Add the first few positive odd integers:

$$1 = 1^2, \qquad 1 + 3 = 2^2, \qquad 1 + 3 + 5 = 3^2.$$

The next sum is $1 + 3 + 5 + 7 = 16 = 4^2$. These calculations suggest that the sum of the first n positive odd integers is always n^2. How could we prove a claim about a sum whose number of terms changes with n?

Suppose the sum of the first k odd integers really is k^2. The next odd integer is $2k + 1$. Adding it would give $k^2 + (2k + 1) = (k + 1)^2$, exactly the next square. We have found a way to pass from one case to the next. We also know where to begin: the one-term sum is $1 = 1^2$.

This is the idea of **mathematical induction**. Chapters 3 and 4 taught us to state assumptions carefully and justify a conclusion from them. In an induction proof, the main implication explains how an established case leads to a later one. A starting case allows that implication to begin doing its work.

We will study ordinary, or weak, induction and strong induction together. They differ in which earlier cases the inductive hypothesis makes available. Both have the same proving power. Choosing a useful hypothesis and checking its range will matter more than the name.

What you should be able to do

After working through this chapter, you should be able to:

- state the claim $P(n)$ and its first permitted index;
- distinguish a base case, an inductive hypothesis and the next target;
- construct ordinary induction proofs of elementary identities, divisibility claims and inequalities;
- use strong induction when a proof needs earlier cases beyond the immediately preceding one;
- justify every earlier index used and check enough starting cases;
- explain why the two induction principles have equivalent proving power;
- diagnose missing bases, circular assumptions and steps that skip required indices.

5.1 Indexed claims

We use $P(n)$ for a statement depending on an integer index n. The goal is to prove $P(n)$ for every integer $n \geq n_0$, where $n_0 \in \mathbb{N}$ is a stated starting index. Recall that $\mathbb{N} = \{0, 1, 2, \ldots\}$ in these notes. A particular claim may begin at 1, 3 or another index without changing this convention.

For the opening claim, $P(n)$ is

$$1 + 3 + \cdots + (2n - 1) = n^2, \qquad n \geq 1.$$

The left side means the sum of the first n positive odd integers. For $n = 1$, it has just the term 1. The dots indicate the intervening odd integers when there are more terms.

Example 5.1. For this $P(n)$, the statements at k and $k + 1$ are

$$\begin{aligned} P(k) &: \quad 1 + 3 + \cdots + (2k - 1) = k^2, \\ P(k + 1) &: \quad 1 + 3 + \cdots + (2k - 1) + (2k + 1) = (k + 1)^2. \end{aligned}$$

Both sides change when the index changes. The last term at index $k + 1$ is $2(k + 1) - 1 = 2k + 1$, and the proposed value of that longer sum is $(k + 1)^2$.

Writing down $P(k + 1)$ identifies the target. It does not give us permission to assume that the displayed equation is true. In the proof, we will start with the shorter sum whose value the hypothesis supplies, then add the new term.

5.2 Ordinary (weak) induction

❄ Definition 5.1 — The principle of ordinary induction

Let $P(n)$ be a claim for integer indices $n \geq n_0$, with $n_0 \in \mathbb{N}$. To prove it for all these indices, establish:

1. **Base case:** prove $P(n_0)$.
2. **Inductive step:** for an arbitrary integer $k \geq n_0$, assume $P(k)$ and prove $P(k + 1)$.

Then $P(n)$ holds for every integer $n \geq n_0$. The assumption $P(k)$ within the step is the **inductive hypothesis**. This method is also called **weak induction**.

The step proves an implication $P(k) \to P(k + 1)$ for every permitted k. It uses the same direct-proof logic as earlier chapters: assume the antecedent within that argument and establish its consequence. The base case supplies an actual true statement to which this implication can first be applied.

For instance, if $n_0 = 1$, the base gives $P(1)$; the step with $k = 1$ gives $P(2)$; the step with $k = 2$ then gives $P(3)$. Any fixed later index is reached after finitely many such steps. The proof gives one argument for every transition, so we do not have to write each transition separately.

✍ Assume one case within the step

Assuming $P(k)$ to prove $P(k+1)$ does not assume that $P(n)$ already holds for every n. The step is a conditional argument. The base case and the proved implication together establish the universal conclusion. Assuming that conclusion at the outset would leave nothing proved.

Example 5.2. Prove that $1 + 3 + \cdots + (2n-1) = n^2$ for every integer $n \geq 1$.

Proof. Let $P(n)$ be the stated identity.

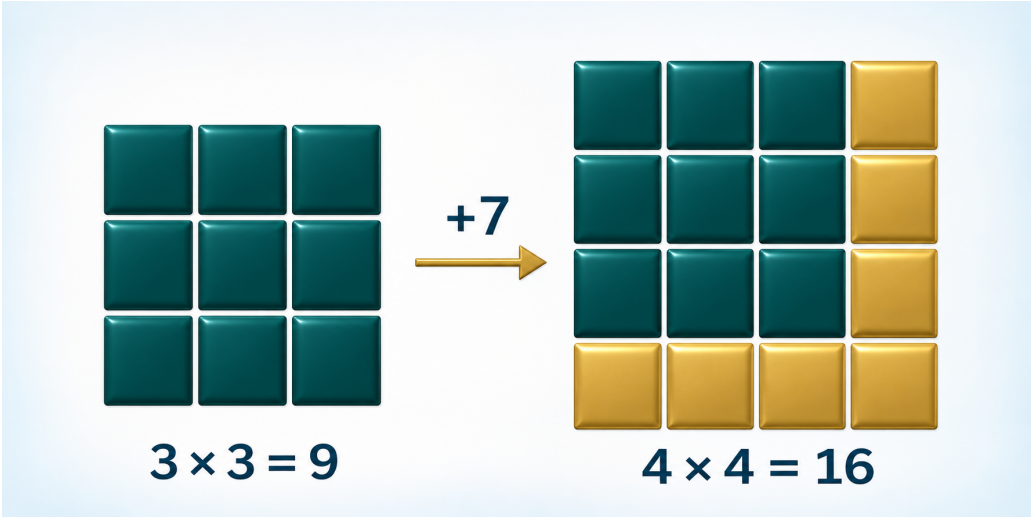
Base case. At $n = 1$, the sum is 1, which equals 1^2. Thus $P(1)$ holds.

Inductive step. Fix an arbitrary integer $k \geq 1$ and assume $P(k)$: $1 + 3 + \cdots + (2k-1) = k^2$. Adding the next odd integer gives

$$\begin{aligned} 1 + 3 + \cdots + (2k-1) + (2k+1) &= k^2 + (2k+1) \quad \text{by the inductive hypothesis} \\ &= (k+1)^2. \end{aligned}$$

This is $P(k+1)$. By ordinary induction, the identity holds for every integer $n \geq 1$. □

The hypothesis was used only to replace the shorter sum by k^2. The remaining algebra established the new target. We did not assume the value of the longer sum.



At $k = 3$, the next odd number supplies the seven tiles that enlarge a 3×3 square to 4×4.

Choosing $k = 100$ and proving one implication about that case would not establish the required step for every k. In the proof above, no property of k was used beyond $k \geq 1$ and the stated inductive hypothesis. The same step therefore works wherever it is needed.

Python (non-examinable)

We can calculate the first five odd-integer sums alongside the proposed squares. The variable `total` stores the running sum; `range(1, 6)` supplies the indices 1, 2, 3, 4, 5, stopping before 6. In Python, `**` denotes a power.

```
total = 0
for n in range(1, 6):
    total = total + (2 * n - 1)
    print(n, total, n ** 2)
```

Output.

```
1 1 1
2 4 4
3 9 9
4 16 16
5 25 25
```

The columns give n, the calculated sum and n^2. These five checks illustrate the identity. The induction proof above establishes it for every permitted n; the finite calculation does not.

5.3 Sums, divisibility and inequalities

Induction becomes useful when the expression at the next index can be related to the current one. A sum gains a term; a power gains a factor; a polynomial can be expanded to expose the previous expression.

❄ Definition 5.2 — Finite summation notation

The symbol $\sum$ means that the indicated terms are added. For integers $0 \le r \le s$, if a_j denotes the term labelled by the index j, then

$$\sum_{j=r}^{s} a_j = a_r + a_{r+1} + \cdots + a_s.$$

The lower limit gives the first index and the upper limit the last. If $r = s$, there is just one term. Throughout these examples the limits include both endpoints.

Example 5.3. We have

$$\sum_{j=1}^{4} (2j+1) = 3 + 5 + 7 + 9 = 24.$$

The first three terms give $\sum_{j=1}^{3}(2j+1) = 15$; the term added when the upper limit becomes 4 is $2(4)+1 = 9$. In general, for $k \ge 1$,

$$\sum_{j=1}^{k+1}(2j+1) = \sum_{j=1}^{k}(2j+1) + \big(2(k+1)+1\big).$$

The summation index is j, leaving k to name the current upper limit. In contrast, $\sum_{j=0}^{0} 2^j$ has one term, $2^0 = 1$; it is not an empty sum.

Example 5.4. Prove that, for every $n \in \mathbb{N}$,

$$\sum_{j=0}^{n} 2^j = 2^{n+1} - 1.$$

Proof. Let $P(n)$ be this identity.

Base case. At $n = 0$, the left side is $2^0 = 1$ and the right side is $2^1 - 1 = 1$.

Inductive step. Let $k \geq 0$ be an integer and suppose $\sum_{j=0}^{k} 2^j = 2^{k+1} - 1$. Then

$$\begin{aligned}\sum_{j=0}^{k+1} 2^j &= \left(\sum_{j=0}^{k} 2^j\right) + 2^{k+1} \\ &= (2^{k+1} - 1) + 2^{k+1} \quad \text{by the inductive hypothesis} \\ &= 2 \cdot 2^{k+1} - 1 \\ &= 2^{k+2} - 1.\end{aligned}$$

Since $k + 2 = (k + 1) + 1$, this is the right side required by $P(k + 1)$. Induction proves the claim for all $n \in \mathbb{N}$. □

At index n, this sum has $n + 1$ terms because it starts at 0. Writing the next target first helps prevent replacing its exponent by the wrong expression.

Example 5.5. Prove that $7 \mid (8^n - 1)$ for every $n \in \mathbb{N}$. Recall that divisibility requires an integer witness, and that $7 \mid 0$ is allowed.

Proof. Let $P(n)$ be the claim $7 \mid (8^n - 1)$.

Base case. For $n = 0$, $8^0 - 1 = 0 = 7(0)$.

Inductive step. Fix an integer $k \geq 0$ and assume $P(k)$. Then $8^k - 1 = 7q$ for some $q \in \mathbb{Z}$. Rewrite the next expression as

$$\begin{aligned}8^{k+1} - 1 &= 8(8^k - 1) + 7 \\ &= 8(7q) + 7 \\ &= 7(8q + 1).\end{aligned}$$

The number $8q + 1$ is an integer, so $7 \mid (8^{k+1} - 1)$. This proves $P(k + 1)$. Hence $P(n)$ holds for every $n \in \mathbb{N}$ by induction. □

The first rewrite made $8^k - 1$ appear, allowing the hypothesis to be used. The witness for the next case was then constructed from the witness for the current one.

Example 5.6. Prove that $3 \mid (n^3 - n)$ for all $n \in \mathbb{N}$.

Proof. Let $P(n)$ be the stated divisibility claim. At $n = 0$, $n^3 - n = 0$, which is divisible by 3.

For the inductive step, fix $k \geq 0$ and suppose $k^3 - k = 3q$ for an integer q. Expanding gives

$$\begin{aligned}(k+1)^3 - (k+1) &= k^3 + 3k^2 + 3k + 1 - k - 1 \\ &= (k^3 - k) + 3k(k+1) \\ &= 3\big(q + k(k+1)\big).\end{aligned}$$

Since $q + k(k+1)$ is an integer, the next expression is divisible by 3. The base case and step establish $P(n)$ for every $n \in \mathbb{N}$. □

Here the expression supplied by the hypothesis was obtained by expanding and regrouping. There is no requirement that it appear unchanged before we begin the algebra.

Example 5.7. Prove that $3^n \geq 2n + 1$ for all $n \in \mathbb{N}$.

Proof. The base case $n = 0$ gives $1 \geq 1$. For an arbitrary integer $k \geq 0$, assume $3^k \geq 2k + 1$. Then

$$\begin{aligned}3^{k+1} &= 3 \cdot 3^k \\ &\geq 3(2k+1) && \text{by the hypothesis, multiplying by } 3 > 0 \\ &= 6k + 3 \\ &\geq 2k + 3 && \text{since } 4k \geq 0 \\ &= 2(k+1) + 1.\end{aligned}$$

This is the required next inequality. By induction, the claim holds for all $n \in \mathbb{N}$. □

We did not need equality at the intermediate steps. We needed estimates in the direction that gives the target lower bound.

Example 5.8. Prove that $2^n > n + 2$ for every integer $n \geq 3$. The claim is false at $n = 2$, where it would say $4 > 4$.

Proof. Let $P(n)$ be $2^n > n + 2$, with domain $n \geq 3$. The base case is $P(3)$: $8 > 5$.

Fix an integer $k \geq 3$ and assume $2^k > k + 2$. Then

$$2^{k+1} = 2 \cdot 2^k > 2(k+2) = 2k + 4 > k + 3 = (k+1) + 2,$$

where $2k + 4 > k + 3$ follows from $k + 1 > 0$. Thus $P(k+1)$ holds. Induction proves the claim for all integers $n \geq 3$. □

The proof starts at 3 because that is the first index in its claim. It does not need $P(0)$, $P(1)$ or $P(2)$.

5.4 Strong induction

Ordinary induction makes $P(k)$ available when proving $P(k+1)$. Sometimes the useful reduction leads to a case further back. Strong induction allows all the earlier cases within the claimed range to be used.

❄ Definition 5.3 — The principle of strong induction

Let $P(n)$ be a claim for integers $n \geq n_0$, with $n_0 \in \mathbb{N}$. Choose a final base index $r \geq n_0$ and establish:

1. **Base case or cases:** prove every case $P(n_0), P(n_0+1), \ldots, P(r)$.
2. **Inductive step:** for an arbitrary integer $k \geq r$, assume $P(j)$ for every integer j with $n_0 \leq j \leq k$, and prove $P(k+1)$.

Then $P(n)$ holds for every integer $n \geq n_0$. If only one base case is needed, take $r = n_0$.

The stronger hypothesis still stops before the target. It includes $P(k)$ and any earlier case in the range; it does not include $P(k+1)$. The base cases must make the first use of the general step possible. Their number depends on which earlier indices the argument needs.

Example 5.9. Suppose the strong inductive hypothesis is that $P(j)$ holds for every integer j with $3 \leq j \leq k$, where $k \geq 5$. Then $P(3)$ and $P(k)$ are available. So is $P(k-2)$: the inequality $k \geq 5$ gives $k-2 \geq 3$, and certainly $k-2 \leq k$.

The hypothesis does not supply $P(2)$, which is below its range. It does not supply $P(k+1)$, which is the target beyond that range. These statements might have truth values for other reasons, but they are not assumptions licensed by this hypothesis.

Example 5.10. In a mathematical model of message construction, a block has length 4 or 5 units. Blocks can be joined, but not split. Prove that every integer length $n \geq 12$ can be formed. Equivalently, prove that $n = 4a + 5b$ for some $a, b \in \mathbb{N}$.

Adding a 4-unit block constructs the target from a length 4 smaller. This suggests using an earlier case and checking enough starting lengths to leave no gap.

Proof. Let $P(n)$ mean that $n = 4a + 5b$ for some $a, b \in \mathbb{N}$.

Base cases. The four lengths 12, 13, 14, 15 can be formed:

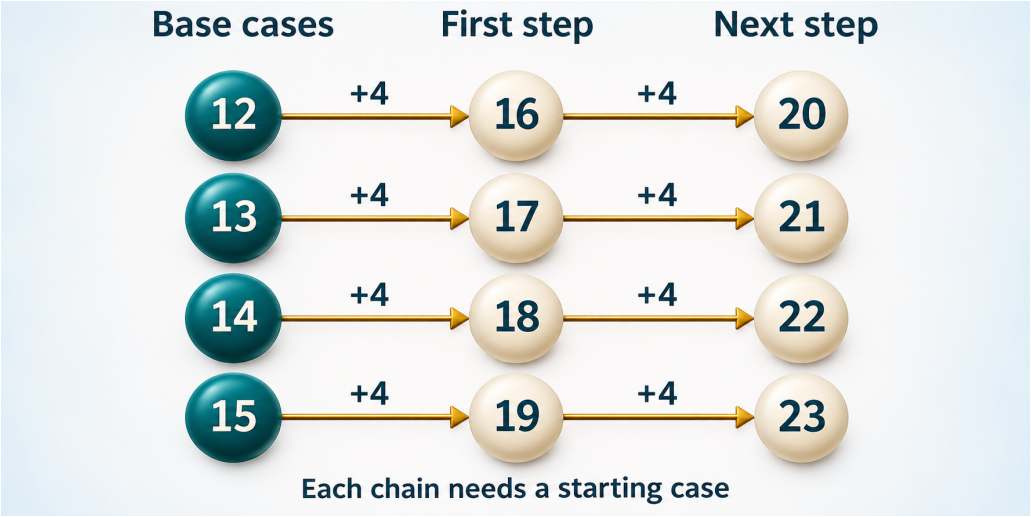
$$12 = 4(3) + 5(0), \quad 13 = 4(2) + 5(1),$$
$$14 = 4(1) + 5(2), \quad 15 = 4(0) + 5(3).$$

Inductive step. Fix an arbitrary integer $k \geq 15$ and assume $P(j)$ for every integer $12 \leq j \leq k$. To form length $k+1$, consider the earlier length $k-3$. Its index satisfies $12 \leq k-3 \leq k$, so the hypothesis gives $k-3 = 4a + 5b$ for some $a, b \in \mathbb{N}$. Adding a 4-unit block yields

$$k + 1 = (k-3) + 4 = 4(a+1) + 5b.$$

Both $a+1$ and b are natural numbers. Thus $P(k+1)$ holds. Strong induction proves the claim for every integer $n \geq 12$. □

The first new length covered by the step is 16, obtained from 12. The same step gives 17 from 13, 18 from 14 and 19 from 15. Checking only 12 would start one of these chains and leave the others without a starting case. For a later target such as 23, the proof can use the already available length 19.



Adding 4 advances each chain; the bases 12, 13, 14 and 15 start all four.

5.5 Recursive rules

A **sequence** is a list of values labelled by integer indices. Here a_n means the value at index n; it does not mean a multiplied by n. A **recursive rule** specifies a new value using earlier values. Initial values tell us where to start. We will study sequences systematically in Chapter 11; the particular rule below needs only these conventions.

Define integer values by

$$a_0 = 1, \qquad a_1 = 3, \qquad a_{n+2} = 2a_{n+1} - a_n \quad (n \geq 0).$$

The first two values are specified separately. The rule gives $a_2 = 2(3) - 1 = 5$ and then $a_3 = 2(5) - 3 = 7$. It uses two earlier values each time. Twice one earlier integer minus the other is again an integer.

Example 5.11. Prove that the values defined above satisfy $a_n = 2n + 1$ for every $n \in \mathbb{N}$.

Proof. Let $P(n)$ be $a_n = 2n + 1$.

Base cases. At $n = 0$, $a_0 = 1 = 2(0) + 1$. At $n = 1$, $a_1 = 3 = 2(1) + 1$.

Inductive step. Fix an integer $k \geq 1$ and assume $P(j)$ for every $0 \leq j \leq k$. In particular, $a_k = 2k + 1$ and $a_{k-1} = 2(k - 1) + 1$. Both uses are permitted because $0 \leq k - 1 \leq k$. Substituting $n = k - 1 \geq 0$ into the recursive rule gives

$$\begin{aligned} a_{k+1} &= 2a_k - a_{k-1} \\ &= 2(2k + 1) - \big(2(k - 1) + 1\big) \\ &= 4k + 2 - (2k - 1) \\ &= 2k + 3 \\ &= 2(k + 1) + 1. \end{aligned}$$

Thus $P(k + 1)$ holds. Strong induction proves $a_n = 2n + 1$ for all $n \in \mathbb{N}$. □

The step uses $P(k)$ and $P(k-1)$, although the hypothesis makes every earlier case available. It need not use all of them. At its first permitted value $k = 1$, the step establishes $P(2)$ from the two checked bases. No value at index -1 is needed.

5.6 Choice of hypothesis

The names "weak" and "strong" describe the hypotheses used in the step. Strong induction gives more earlier cases at once, but does not prove a larger collection of statements about the natural numbers. Both principles can be used to obtain the other.

Ordinary induction is available within strong induction because the strong hypothesis includes $P(k)$. If that is all the step needs, the other earlier cases may simply remain unused.

For the other direction, suppose we have the bases and step of a strong induction argument, with indices beginning at n_0 and last base index r. Define $Q(k)$ to say:

$$P(j) \text{ holds for every integer } j \text{ with } n_0 \leq j \leq k.$$

The checked bases establish $Q(r)$. For an arbitrary integer $k \geq r$, assuming $Q(k)$ supplies exactly the earlier cases needed by the strong step, so that step proves $P(k+1)$. Together with $Q(k)$, the new case gives $Q(k+1)$. Ordinary induction applied to Q therefore establishes all the required P cases. This is why strengthening the claim used in ordinary induction can reproduce a strong induction proof.

Example 5.12. Use the same values a_n from Example 5.11. Instead of assuming only one formula, let $Q(k)$ be the paired claim

$$a_k = 2k + 1 \quad \text{and} \quad a_{k+1} = 2k + 3.$$

We can prove $Q(k)$ by ordinary induction.

Proof. At $k = 0$, both parts hold because $a_0 = 1$ and $a_1 = 3$. For an arbitrary $k \geq 0$, suppose $Q(k)$ holds. Then

$$a_{k+2} = 2a_{k+1} - a_k = 2(2k + 3) - (2k + 1) = 2k + 5.$$

We already know $a_{k+1} = 2k + 3 = 2(k + 1) + 1$. The new equality is $a_{k+2} = 2(k + 1) + 3$. These are precisely the two parts of $Q(k + 1)$. Ordinary induction proves $Q(k)$ for every $k \in \mathbb{N}$, and hence $a_n = 2n + 1$ for every $n \in \mathbb{N}$. □

The paired claim carries forward the old second value and establishes the new second value. The arithmetic has not become more powerful; we have organised the information differently.

For a sum whose next term is added to the previous sum, ordinary induction usually provides enough information. For the two-value rule, we needed two earlier formulas. For the block construction, we needed a case four indices before the target.

Example	Earlier information used	Convenient approach
Odd-integer sum	The sum at k.	Ordinary induction.
Two-value rule	The formulas at k and $k-1$.	Strong induction, or an ordinary proof of a paired claim.
Block construction	The representation at $k-3$ for target $k+1$.	Strong induction with four starting cases.

Whichever version we choose, write the target first, identify the earlier indices needed and verify that the hypothesis actually includes them.

Example 5.13. For $n \in \mathbb{N}$, the number $n(n+1)$ is even. If n is even, its product with the integer $n+1$ is even. If n is odd, write $n = 2q+1$ with $q \in \mathbb{Z}$. Then $n+1 = 2(q+1)$ is even, so the product is again even. The parity alternatives cover every permitted n, establishing the claim directly.

Induction is useful when a transition between indices supplies a manageable proof. It is one method to consider alongside the direct and indirect methods of the earlier chapters.

5.7 Errors in induction

Check the base cases and the step as separate obligations. Within the step, check its direction, the exact hypothesis used and the index of the case it first establishes.

Example 5.14. Consider $P(n)$: $n+1 \leq n$, for $n \in \mathbb{N}$. If we temporarily assume $P(k)$, adding 1 to both sides gives $k+2 \leq k+1$, which is $P(k+1)$. Thus the transition implication is valid.

The universal claim is still false. Its proposed base $P(0)$ would be $1 \leq 0$, which is false. A proved implication does not establish its antecedent. There is no true starting case here from which the argument can begin.

Example 5.15. David claims that every natural number is even. He checks that 0 is even and observes that if k is even, then $k+2$ is even.

Both facts are correct, but the step advances from k to $k+2$, not to $k+1$. Starting from 0, it reaches $0, 2, 4, 6, \ldots$, leaving the odd indices untouched. The permitted input 1 refutes David's universal claim. An induction argument must cover all the indices in its conclusion.

Example 5.16. While trying to prove $7 \mid (8^{k+1} - 1)$, Grace writes: "Let $8^{k+1} - 1 = 7r$ for an integer r. Therefore $7 \mid (8^{k+1} - 1)$."

The supposed integer r already asserts exactly the target divisibility claim. Its existence has not been established. The available hypothesis supplies an integer q with $8^k - 1 = 7q$, not a witness for the next exponent. Example 5.5 repairs the argument by constructing $r = 8q + 1$ and checking the required equation.

Example 5.17. In Example 5.11, the step uses $P(k-1)$. Starting that step at $k = 0$ would require $P(-1)$, outside the range of the claim and of the defined values. Checking $P(0)$ alone would not fix this.

Instead, we checked $P(0)$ and $P(1)$, then began the general step at $k = 1$. Its first target is $P(2)$, and both cases it uses are available. For any proposed step, substitute its smallest permitted k before relying on the general notation.

5.8 Exercises

5.1. Let $P(n)$ be the claim $2 + 4 + \cdots + 2n = n(n+1)$, for integers $n \geq 1$.

a) State and verify its first case.

b) Write $P(k)$ and $P(k+1)$, including the last term and right side in each.

c) Which equation is assumed and which must be proved in an ordinary inductive step?

5.2. Eve checks $n = 0, 1, \ldots, 9$ in the claim $n < 10$ and concludes that it holds for every $n \in \mathbb{N}$.

a) Give a counterexample to her conclusion.

b) Has she proved a transition from an arbitrary permitted index to the next one? Explain.

5.3. Suppose $P(3)$ has been proved, together with $P(k) \to P(k+1)$ for every integer $k \geq 3$. Which statements below are guaranteed by these facts? Explain the range of the conclusion.

a) $P(2)$.

b) $P(3)$.

c) $P(8)$.

d) $P(100)$.

5.4. Prove the claim from Exercise 5.1. After stating the hypothesis for an arbitrary integer $k \geq 1$, complete the calculation

$$\begin{aligned} 2 + 4 + \cdots + 2k + 2(k+1) &= \underline{\qquad\qquad} + 2(k+1) \\ &= (k+1)\big(\underline{\qquad\qquad}\big) \\ &= (k+1)(k+2). \end{aligned}$$

Identify the line using the hypothesis, check the base case and state the final conclusion.

5.5. Use ordinary induction in each part. Write the next target before beginning the step.

a) For integers $n \geq 1$, $\displaystyle\sum_{j=1}^{n}(3j-1) = \frac{n(3n+1)}{2}$.

b) For $n \in \mathbb{N}$, $\displaystyle\sum_{j=0}^{n} 3^j = \frac{3^{n+1}-1}{2}$.

5.6. Prove by ordinary induction that $4 \mid (9^n - 1)$ for all $n \in \mathbb{N}$. In the step, construct the new witness explicitly from the integer supplied by the inductive hypothesis.

5.7. Prove by ordinary induction that $2 \mid (n^2 - n)$ for every $n \in \mathbb{N}$. Show how $(k+1)^2 - (k+1)$ can be written using $k^2 - k$ and an additional even quantity.

5.8. Prove by ordinary induction that $2^n \geq n + 1$ for every $n \in \mathbb{N}$. Justify the estimate used after applying the hypothesis.

5.9. Show that $2^n > 2n$ fails at $n = 2$ but holds at $n = 3$. Then prove by ordinary induction that it holds for every integer $n \geq 3$. State the permitted range of k in the step.

5.10. Suppose the inductive hypothesis is $\sum_{j=0}^{k} 3^j = (3^{k+1} - 1)/2$.

a) Write the exact equation required at index $k + 1$.

b) Which single term is added to the sum in the hypothesis?

c) How many terms occur in the sum at index $k + 1$?

5.11. Assume $P(j)$ for every integer $4 \le j \le k$, with $k \ge 7$. Let m be an integer satisfying $4 \le m < k$. Which statements below are guaranteed to be within the hypothesis for every permitted choice of k, m? Justify the index bounds.

a) $P(m)$.

b) $P(m + 1)$.

c) $P(k - 4)$.

d) $P(k + 1)$.

5.12. You want to prove $P(n)$ for every integer $n \ge 4$. Your proposed strong-induction step proves $P(k + 1)$ using $P(k - 2)$.

a) What is the smallest value of k for which $k - 2$ is in the claimed range?

b) Which consecutive base cases must be checked before that step begins?

c) Identify the first three new target indices and the earlier index used for each.

5.13. A model allows blocks of length 3 or 5 units, joined without splitting. Prove by strong induction that every integer length $n \ge 8$ can be formed.

a) Express $8, 9, 10$ as $3a + 5b$ with $a, b \in \mathbb{N}$.

b) For target $k + 1$ with $k \ge 10$, identify the earlier length obtained by subtracting 3, and check its index range.

c) Complete the inductive step and state why the three base cases cover the first uses of your construction.

5.14. Define

$$b_0 = 0, \qquad b_1 = 2, \qquad b_{n+2} = 3b_{n+1} - 2b_n \quad (n \ge 0).$$

Prove by strong induction that $b_n = 2^{n+1} - 2$ for all $n \in \mathbb{N}$.

a) Check the two starting cases.

b) State a hypothesis that supplies the formulas for b_k and b_{k-1}, with a valid lower bound on k.

c) Use the recursive rule to establish the formula for b_{k+1} and conclude the proof.

5.15. Let $c_0 = c_1 = 1$ and $c_{n+2} = c_{n+1} + c_n$ for $n \ge 0$. Prove by strong induction that $c_n \le 2^n$ for all $n \in \mathbb{N}$. Check both bases and show why the two earlier bounds imply the bound at $k + 1$.

5.16. Define integers by $d_0 = 1$, $d_1 = 3$ and $d_{n+2} = d_{n+1} + d_n + 1$ for $n \ge 0$. Prove by strong induction that every d_n is odd. In the step, use separate integer witnesses for the two earlier odd values and identify the new witness.

5.17. a) Compare the hypotheses available when proving $P(k+1)$ by ordinary and strong induction.

b) Explain why a stronger hypothesis does not give the principle greater proving power here. Describe the combined claim $Q(k)$ that lets ordinary induction retain all earlier P cases.

c) Does a strong-induction proof have to use every earlier case in its hypothesis? Explain.

5.18. For the values b_n in Exercise 5.14, let $Q(k)$ state both $b_k = 2^{k+1} - 2$ and $b_{k+1} = 2^{k+2} - 2$. Describe an ordinary induction proof of $Q(k)$ for $k \in \mathbb{N}$. Give its base, its hypothesis and its next target, and explain which equality is retained and which must be newly established. A full algebraic calculation is not required.

5.19. A student claims $5 \mid 2^n$ for every $n \in \mathbb{N}$. In the step, they assume $2^k = 5q$ for an integer q and derive $2^{k+1} = 5(2q)$.

a) Is the step's conditional argument valid?

b) Check the proposed base $n = 0$. Is the universal claim true?

c) Explain why the valid conditional does not establish the claim by induction.

5.20. For a family of claims $P(n)$ on $\mathbb{N}$, a student establishes $P(0)$ and $P(k) \to P(k+2)$ for all $k \in \mathbb{N}$.

a) Which indices can be reached from the proved base?

b) What additional base would be needed to reach the remaining indices using the same step?

c) Suppose $P(n)$ means "n is even". Can that additional base be verified? What conclusion is actually justified?

5.21. In a proposed step for $4 \mid (9^n - 1)$, the hypothesis is $9^k - 1 = 4q$ for an integer q. The student writes:

> Let $r = (9^{k+1} - 1)/4$. Then $9^{k+1} - 1 = 4r$, so the target is divisible by 4.

a) What property of r remains unjustified?

b) Use the hypothesis to construct an integer witness for the next case and repair the step.

5.22. Let $u_0 = 2$, $u_1 = 4$ and $u_{n+2} = u_{n+1} + 2u_n$ for $n \geq 0$. To prove $u_n = 2^{n+1}$, a proposed argument checks $n = 0, 1$, then begins its general step at $k \geq 2$:

> Assume $u_j = 2^{j+1}$ for all $0 \leq j \leq k$. Then $u_{k+1} = u_k + 2u_{k-1} = 2^{k+1} + 2 \cdot 2^k = 2^{k+2}$.

a) What is the first target index covered by the stated step? Which case between the bases and that target has not been checked?

b) Show that the general step can start earlier. Check the indices it uses at its first permitted value.

c) Complete the corrected induction proof.

5.23. For each claim, choose ordinary induction, strong induction or a short direct proof. Give a reason and the useful starting assumption or assumptions. More than one method may work.

a) $3 \mid (6n + 9)$ for every $n \in \mathbb{N}$.

b) $\sum_{j=1}^{n} 2j = n(n + 1)$ for every integer $n \geq 1$.

c) The formula for b_n in Exercise 5.14.

5.24. A model begins with total $t_0 = 3$. At step $n + 1$ it adds $2(n + 1)$, so

$$t_{n+1} = t_n + 2(n + 1) \qquad (n \geq 0).$$

Prove that $t_n = n(n + 1) + 3$ for all $n \in \mathbb{N}$. State your induction method, verify its base and show how the update rule turns the hypothesis into the next target.

LECTURE NOTES
DISCRETE MATHEMATICS

CHAPTER 6

Graphs



Updated: 25 November 2026

6 Graphs

A message can travel between two computers even when they have no direct link: other computers may pass it along. If one link fails, another route might remain. Which connections are essential, and which provide an alternative route?

To study these questions, we can represent each computer by a vertex and each direct two-way link by an edge. The resulting mathematical object is a **graph**. We record which pairs are linked; the positions of the computers on a page do not determine their connections.

This uses Chapter 4's sets and relationships in a concrete setting. It also gives Chapter 5's induction a new job. Instead of proving a formula about one number at each index, we will prove a claim about every tree with a given number of vertices. The inductive step will depend on reducing an arbitrary tree to a smaller one.

What you should be able to do

After working through this chapter, you should be able to:

- specify a simple graph and read its drawings and neighbour lists;
- calculate degrees and use the degree-sum formula;
- distinguish walks, paths and cycles, and justify connectedness;
- recognise trees and explain their unique-path property;
- use leaf deletion and induction to prove the tree edge count;
- construct and check spanning trees, including by an edge-adding algorithm;
- interpret roots, parents, children, depths and rooted leaves.

6.1 Graph representations

❄ Definition 6.1 — Finite simple undirected graph

A **graph** $G = (V, E)$ consists of a finite nonempty set V of **vertices** and a set E of **edges**, each a two-element subset of V. Two vertices are **adjacent**, or **neighbours**, when their pair is an edge. An edge is **incident** to each of its two endpoints.

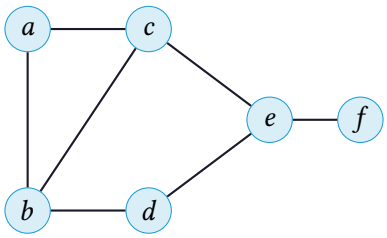
All graphs in this chapter are of this kind. An edge $\{u, v\}$ has no direction because $\{u, v\} = \{v, u\}$. Its endpoints are distinct: $\{u, u\} = \{u\}$ has only one element and is not allowed as an edge. Since E is a set, listing the same pair twice does not create a second edge.

For a finite set S, the notation $|S|$ means its number of elements. We will often write $n = |V|$ for the number of vertices and $m = |E|$ for the number of edges. The set E may be empty even though V must be nonempty.

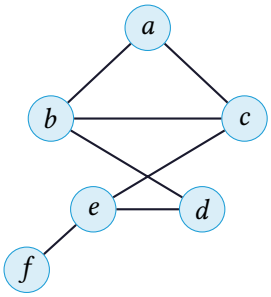
Example 6.1. Use the following fixed graph G in the worked examples unless another graph is specified:

$$V = \{a, b, c, d, e, f\},$$
$$E = \{\{a, b\}, \{a, c\}, \{b, c\}, \{b, d\}, \{c, e\}, \{d, e\}, \{e, f\}\}.$$

There are six vertices and seven edges. Each drawing below represents exactly these sets.



Drawing 1



Drawing 2

In Drawing 2, the edges $\{b, d\}$ and $\{c, e\}$ cross at a point with no vertex marked there. The crossing supplies no extra vertex and no extra connection. A message following $\{b, d\}$ cannot switch to $\{c, e\}$ at that unmarked point in this model.

Moving a labelled vertex or bending an edge does not change G as long as every edge keeps the same endpoints. Conversely, two drawings with six vertices and seven edges need not represent the same graph: their actual vertex and edge sets must agree.

6.2 Vertex degrees

❄ Definition 6.2 — Degree and isolated vertex

The **degree** $\deg(v)$ of a vertex v is the number of edges incident to it. In a simple graph, this is also the number of its neighbours. A vertex of degree 0 is **isolated**.

Example 6.2. For the graph in Example 6.1, its neighbour lists give another complete representation.

Vertex	Neighbours	Degree
a	b, c	2
b	a, c, d	3
c	a, b, e	3
d	b, e	2
e	c, d, f	3
f	e	1

For example, c has degree 3 because it is incident to $\{a, c\}, \{b, c\}, \{c, e\}$. The crossing in Drawing 2 does not increase its degree. Every neighbour entry must be reciprocated: if c lists e, then e lists c.

Adding a new vertex g with no incident edges would produce a different graph. Its neighbour list would be empty and $\deg(g) = 0$. A drawing or list must retain such a vertex even though it contributes no edges.

The degree-sum formula. Each edge has two distinct endpoints. When we add all vertex degrees, that edge is counted once at each endpoint. It therefore contributes exactly 2 to the sum. This proves the **degree-sum formula**, also called the **handshake lemma**:

$$\sum_{v \in V} \deg(v) = 2|E| = 2m.$$

Here $\sum_{v \in V}$ means that we add one degree for each vertex in the finite set V. The sum of the degrees is therefore always even.

Example 6.3. The six degrees in the table have sum $2 + 3 + 3 + 2 + 3 + 1 = 14$. Thus $2m = 14$ and $m = 7$. Adding the degrees without dividing by 2 would count each edge twice.

Python (non-examinable)

A Python **dictionary** can associate each vertex label with its list of neighbours. The entry `"a": ["b", "c"]`, for example, records the two neighbours of a. Square brackets select the list for a label, and `len` counts its entries. The loop visits the labels in the order entered and adds their degrees to `total`.

```
neighbours = {
    "a": ["b", "c"],
    "b": ["a", "c", "d"],
    "c": ["a", "b", "e"],
    "d": ["b", "e"],
    "e": ["c", "d", "f"],
    "f": ["e"]
}
total = 0
for vertex in neighbours:
    degree = len(neighbours[vertex])
    print(vertex, degree)
    total = total + degree
print("Degree sum:", total)
```

Output.

```
a 2
b 3
c 3
d 2
e 3
f 1
Degree sum: 14
```

The lists record the same six vertices and seven edges as the drawing. Every edge occurs in both endpoints' lists, giving the degree sum 14. This representation assumes the reciprocal, duplicate-free neighbour lists required by our simple undirected graph.

Example 6.4. Alice proposes a simple network with seven vertices, each having exactly three neighbours. The degree sum would be $7(3) = 21$, but it must equal twice an integer number of edges. The proposal is impossible.

This conclusion follows from the stated connections, so no choice of positions in a drawing can repair it.

Example 6.5. Could a simple graph on four vertices have degrees 3, 3, 1, 1? Their sum is 8, which passes the parity check. However, each degree-3 vertex must be adjacent to all three other vertices. In particular, each proposed degree-1 vertex would be adjacent to both degree-3 vertices, giving it degree at least 2. This is a contradiction.

The degree-sum formula rules out some proposals. Passing that check alone does not construct a graph or prove one exists.

Example 6.6. Prove that every graph has an even number of vertices of odd degree.

Proof. Suppose instead that an odd number of its vertex degrees were odd. Pair all but one of those odd degrees. Each pair has an even sum, and all the even degrees also have an even sum. The single unpaired odd degree would make the total degree sum odd. This contradicts the degree-sum formula. Therefore the number of odd-degree vertices is even. □

Zero is an allowed even number here. A graph can have no odd-degree vertices.

6.3 Paths and connectedness

Adjacency concerns one edge. To describe travel through several edges, we need a list of the vertices visited in order.

❄ Definition 6.3 — Walk, path and cycle

A **walk** in $G = (V, E)$ is a finite list $v_0, v_1, \dots, v_\ell$ of vertices in V in which each consecutive pair is an edge. Its **length** is the number ℓ of edges traversed. Vertices and edges may repeat in a walk.

A **path** is a walk with no repeated vertices. The one-vertex list v_0 is a path of length 0.

A **cycle** is a closed walk of length at least 3: its last vertex is its first, and all the vertices before that final entry are distinct.

These notes reserve "path" for the no-repetition convention. A cycle is recorded separately, with its starting vertex repeated at the end. Counting the entries in a route gives one more than its length. Choosing a different starting vertex or going around in the opposite direction describes the same cycle.

Example 6.7. In the fixed graph G, the list a, b, c, a, c, e is a walk of length 5. Each of its five steps follows an edge, but it is not a path because a and c repeat.

Remove the closed portion a, b, c, a except for one occurrence of a. The remaining list is a, c, e, a path of length 2 with the same endpoints. More generally, whenever a walk repeats a vertex, the part between two occurrences can be removed. Repeating this shortening process leaves a path between its endpoints.

Example 6.8. The list a, b, d, e, f is a path in G, of length 4. The list a, d, e is not even a walk because $\{a, d\} \notin E$. The list a, b, c, a is a cycle of length 3.

By contrast, a, b, a is a closed walk but not a cycle: it has length 2 and traverses the same edge out and back. Returning to the starting point alone does not establish the presence of a cycle.

❄ Definition 6.4 — Connected graph

A graph is **connected** if there is a path between every pair of its vertices. Otherwise it is **disconnected**. A **connected component** is one connected piece: it contains all the vertices reachable from any one of its vertices, together with the edges between them.

A single vertex with no edges is connected: its length-0 path connects it to itself. In a larger graph, an isolated vertex forms a component by itself.

Example 6.9. The fixed graph G is connected. From a, we can reach b, c, d, e, f using the paths

$$a, b; \quad a, c; \quad a, b, d; \quad a, c, e; \quad a, c, e, f.$$

Since edges have no direction, paths can be traversed backwards. To travel between any two vertices, we can first travel to a and then to the other vertex. This gives a walk, from which repetitions can be removed to obtain a path.

Now consider instead the graph on the same six vertices with edge set $\{\{a, b\}, \{c, d\}, \{e, f\}\}$. Every vertex has degree 1, but the graph has three components. There is no path from a to c. Having a neighbour does not imply being connected to every other vertex.

6.4 Trees and unique paths

❄ Definition 6.5 — Tree, forest and degree-one leaf

A graph is **acyclic** if it has no cycles. A **tree** is a connected acyclic graph. A **forest** is an acyclic graph; it need not be connected. In an unrooted tree, a **leaf** is a vertex of degree 1.

A graph with one vertex and no edges is a tree. Under the degree-one convention, it has no leaves. Any tree with at least two vertices has a different behaviour, which we will prove below.

Example 6.10. Keep all six vertices of G, but retain only the edges

$$F_1 = \{\{a, b\}, \{a, c\}, \{b, d\}, \{c, e\}, \{e, f\}\}.$$

Call this graph T_1. Its vertices occur along the path d, b, a, c, e, f, and its edges are exactly the consecutive pairs in that list. Thus it is connected and has no cycle: it is a tree. Its degree-one leaves are d and f.

Example 6.11. On $\{a, b, c, d, e, f\}$, take the edges $\{a, b\}, \{b, c\}, \{d, e\}, \{e, f\}$. There are two separate paths, a, b, c and d, e, f. The graph has no cycle, so it is a forest. It is not a tree because the two components are not connected to each other.

Example 6.12. Prove that a graph is a tree if and only if every pair of distinct vertices has exactly one path between them.

Proof. First suppose T is a tree. Connectedness provides a path between any two distinct vertices u, v. If there were two different paths, follow their common beginning until they first separate. The paths eventually meet again because they both end at v. The two branches between a separation and their next meeting give a cycle, contradicting the definition of a tree. Therefore the path is unique.

Conversely, suppose every pair of distinct vertices has exactly one path. The existence of these paths makes the graph connected. If it contained a cycle, choose two adjacent vertices on that cycle. Its edge gives one path between them, while going the other way around the cycle gives another. This contradicts uniqueness. Thus the graph has no cycles and is a tree. □

For a one-vertex graph there are no distinct vertex pairs to check, and the graph is indeed a tree. For larger graphs, “exactly one” requires both existence and uniqueness.

Example 6.13. In G, the paths a, c, e and a, b, d, e join the same endpoints. Their branches give the cycle a, c, e, d, b, a. The graph remains connected, but it cannot be a tree. An extra edge such as $\{b, c\}$ does not stop the listed cycle from being a cycle; its own steps and vertices satisfy the definition.

6.5 Leaves and induction

The edge count of a tree will follow by induction on its number of vertices. Before using an inductive hypothesis, we must justify how an arbitrary tree can be reduced to a smaller tree.

Example 6.14. Let T be a tree with at least two vertices. Prove that T has at least two degree-one leaves.

Proof. Because T is finite, there are only finitely many paths. Choose one of greatest length. It has at least one edge because T is connected and has at least two vertices.

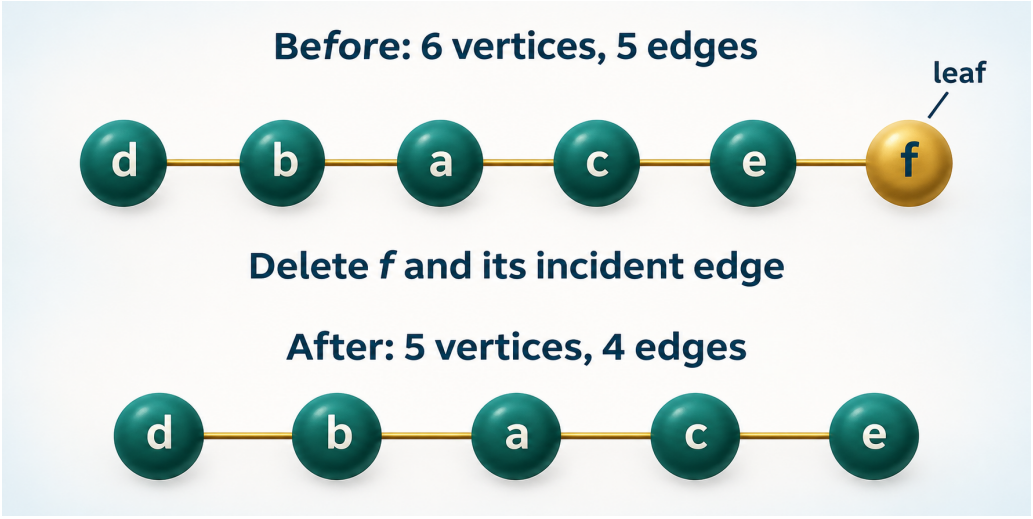
Let u be one endpoint. It has the neighbour that is next along the path. Suppose it had another neighbour w. If w were elsewhere on the path, that edge and the path back to u would form a cycle. If w were outside the path, we could extend the path through w, contradicting its greatest length. Both alternatives are impossible, so $\deg(u) = 1$. The same argument applies to the other endpoint. These are two distinct leaves. □

Absence of cycles ruled out an extra neighbour on the path; choosing a path of greatest length ruled out an extra neighbour outside it.

Example 6.15. Let ℓ be a leaf of a tree T with at least two vertices. Delete ℓ and its single incident edge, obtaining T'.

The remaining graph is nonempty and contains no cycle, because any cycle in it would already have been in T. For any two remaining vertices, their path in T cannot pass through ℓ between its endpoints: doing so would require two different incident path edges, whereas ℓ has degree 1. Thus their path survives in T'. The remaining graph is connected and acyclic, hence a tree.

This removes exactly one vertex and one edge. Deleting an arbitrary vertex of higher degree need not preserve connectedness.



Deleting leaf f from the example path removes one vertex and one edge; all remaining vertices stay connected.

Example 6.16. Prove that a tree with n vertices has $n - 1$ edges.

Proof. For each integer $n \geq 1$, let $P(n)$ say: every tree with n vertices has $n - 1$ edges.

Base case. A one-vertex graph has no edges under our simple-graph definition. Thus every one-vertex tree has $0 = 1 - 1$ edges.

Inductive step. Fix an integer $k \geq 1$ and assume every tree with k vertices has $k - 1$ edges. Let T be an arbitrary tree with $k + 1$ vertices. It has at least two vertices, so Example 6.14 supplies a leaf. Delete that leaf and its incident edge. By Example 6.15, the resulting T' is a tree with k vertices.

The inductive hypothesis applies to T', giving it $k - 1$ edges. Restoring the deleted edge gives T exactly $(k - 1) + 1 = k$ edges. This is $(k + 1) - 1$, as required. Ordinary induction proves $P(n)$ for every integer $n \geq 1$. □

The hypothesis concerned every tree of the smaller size. We did not assume that all trees have the shape of one particular drawing. The leaf argument made the reduction available for an arbitrary T.

Example 6.17. A tree has exactly three vertices of degree 3, and every other vertex has degree 1. How many leaves does it have?

Let L be the number of leaves. There are $L + 3$ vertices, so the tree has $L + 2$ edges. The degree sum is $L + 9$. Hence

$$L + 9 = 2(L + 2), \qquad L = 5.$$

It therefore has eight vertices and seven edges. The degree-sum formula and the tree edge count have supplied different equations about the same graph.

Example 6.18. Take five vertices a, b, c, d, e and edges $\{a, b\}, \{b, c\}, \{c, a\}, \{d, e\}$. There are 5 vertices and $4 = 5 - 1$ edges, but the graph contains a triangle and a separate edge. It is disconnected and is not a tree.

The statement proved in Example 6.16 is an implication from being a tree to having the specified count. Its converse needs an additional condition.

6.6 Spanning trees

❄ Definition 6.6 — Subgraph and spanning tree

A graph $H = (W, F)$ is a **subgraph** of $G = (V, E)$ if $W \subseteq V$ and $F \subseteq E$. As H is itself a graph, each edge of F must have both endpoints in W.

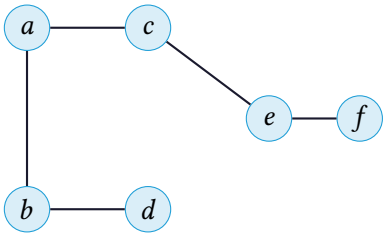
A **spanning tree** of G is a subgraph that is a tree and retains every vertex of G. Thus it has the form (V, F) with $F \subseteq E$.

Keeping a connected part while discarding a troublesome vertex does not produce a spanning tree.

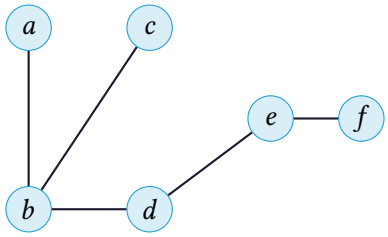
Example 6.19. For the fixed graph G, take

$$F_1 = \{\{a, b\}, \{a, c\}, \{b, d\}, \{c, e\}, \{e, f\}\},$$
$$F_2 = \{\{a, b\}, \{b, c\}, \{b, d\}, \{d, e\}, \{e, f\}\}.$$

The drawings retain every vertex.



$T_1 = (V, F_1)$



$T_2 = (V, F_2)$

Every listed edge belongs to G. The first tree is the path already checked in Section 6.4. The second can be built from b by attaching a, c, d, then attaching e to d and f to e. Each new vertex has exactly one connection to the existing graph, preserving connectedness without creating a cycle. Both are spanning trees, with five edges each.

Any spanning tree of G must retain $\{e, f\}$. It is the only edge incident to f; omitting it would leave f isolated from the other five vertices.

Example 6.20. Every connected graph has a spanning tree.

Start with a connected graph. If it has no cycle, it is already a tree. Otherwise, select an edge on a cycle and delete it, keeping every vertex.

The remaining graph is still connected. The rest of the cycle supplies an alternative path between the deleted edge's endpoints. In any route that used that edge, we can use this alternative instead. The result is a walk; removing repetitions gives a path with the required endpoints.

Repeat whenever a cycle remains. Only finitely many edges were present, and each deletion removes one. The process therefore ends with a connected graph having no cycles and all the original vertices. This is a spanning tree.

The edge selected for deletion must lie on a cycle. Deleting $\{e, f\}$ from G would disconnect it, so arbitrary deletion does not have the same guarantee.

Example 6.21. Let G be connected with n vertices and m edges. By Example 6.20, it contains a spanning tree with $n-1$ edges. Those edges are part of G, so $m \geq n-1$.

If $m = n-1$, the spanning tree uses as many edges as G has. Its edge set must therefore be the whole edge set of G. Consequently, G itself is a tree. We have established the criterion:

$$G \text{ is a tree} \quad \Longleftrightarrow \quad G \text{ is connected and } |E| = |V| - 1.$$

Connectedness is the condition missing from the five-vertex counterexample in Section 6.5.

6.7 Constructing spanning trees

We obtained a spanning tree by deleting edges from a connected graph. We can also build one by starting with all its vertices and adding edges. At first, each vertex is a separate component. An edge is useful when it joins two of the components we have built so far.

This is the unweighted form of **Kruskal's algorithm**. The weighted version assigns a cost to each edge and considers the cheapest edges first. Here we only want a spanning tree, so we need no costs and may choose any order for the edges. Equivalently, all edges can be regarded as having the same cost.

For a connected graph $G = (V, E)$ with n vertices, proceed as follows.

1. Start with the graph $(V, \emptyset)$: retain every vertex and select no edges.
2. Choose an order for the edges of G and examine them one at a time.
3. Accept an edge if its endpoints lie in different components of the graph formed by the edges already accepted. Otherwise, reject it.
4. Stop when $n-1$ edges have been accepted, or when every edge has been examined.

The component test concerns the graph we are building. The input graph G is connected, but the selected edges may still leave several components. If the endpoints of the next edge are already joined by a path of accepted edges, adding it would close a cycle. If they lie in different components, there is no such path, so the new edge joins the components without creating a cycle.

Example 6.22. Use the fixed graph G from Example 6.1. Its edge set has no built-in order; choose to examine the edges as

$$\{a,b\},\ \{a,c\},\ \{b,c\},\ \{b,d\},\ \{c,e\},\ \{d,e\},\ \{e,f\}.$$

Initially, the six vertices form six separate components. The table records the number of components after each decision.

Edge	Decision	Components remaining
$\{a, b\}$	Accept	5
$\{a, c\}$	Accept	4
$\{b, c\}$	Reject	4
$\{b, d\}$	Accept	3
$\{c, e\}$	Accept	2
$\{d, e\}$	Reject	2
$\{e, f\}$	Accept	1

We reject $\{b, c\}$ because the accepted edges already give the path b, a, c. Later, we reject $\{d, e\}$ because the accepted edges give the path d, b, a, c, e. Each rejected edge would therefore create a cycle.

The last edge joins the component on $\{a, b, c, d, e\}$ to the isolated vertex f. The accepted edge set is

$$F = \{\{a, b\}, \{a, c\}, \{b, d\}, \{c, e\}, \{e, f\}\}.$$

This is F_1 from Example 6.19. All six vertices are connected, and we have accepted $6 - 1 = 5$ edges without creating a cycle. A different examination order can produce a different spanning tree.

Why the construction works. Initially there are n components and no cycles. Each accepted edge joins exactly two components, reducing their number by one, and creates no cycle. After k accepted edges, the selected graph is therefore a forest with $n - k$ components. After $n - 1$ accepted edges, it has one component and no cycles. It retains every vertex of G, so it is a spanning tree.

We must also explain why a connected input lets us reach $n - 1$ accepted edges. Suppose instead that we examined every edge but still had separate components. Since G is connected, a path in G between two of these components must contain an edge crossing from one to another. Its endpoints must also have been in different components when we examined it: our components only merge as edges are accepted. We would therefore have accepted that edge, joining those endpoints into the same final component. This contradicts their being in different final components. Thus the procedure reaches a spanning tree.

For a graph with one vertex, $n - 1 = 0$: the starting graph is already a spanning tree. If the input is disconnected, we exhaust the edge list with separate components still present. The same procedure then gives a **spanning forest**, consisting of one tree for each component of the input. No selection of its edges can connect vertices that were in different input components.

6.8 Rooted trees

A tree's definition does not select a top or a starting vertex. A hierarchy needs that extra choice.

❄ Definition 6.7 — Root, parent, child and depth

A **rooted tree** is a tree with one vertex designated as its **root**. For a non-root vertex v, its **parent** is the vertex immediately before v on the unique path from the root to v; v is a **child** of that parent. The root has no parent.

The **depth** of a vertex is the length of its unique path from the root. The root has depth 0. Distinct children of the same parent are **siblings**.

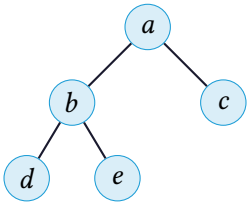
Unique paths make the parent unambiguous. Every non-root vertex has exactly one parent, although a parent may have several children.

❄ Definition 6.8 — Rooted leaves and full binary trees

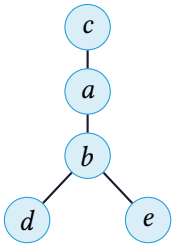
In a rooted tree, a **leaf** is a vertex with no children. An **internal vertex** has at least one child. A **full binary tree** is a rooted tree in which every internal vertex has exactly two children.

The rooted leaf convention agrees with degree 1 at a non-root vertex. The root needs separate care: a root of degree 1 has one child, so is not a rooted leaf. A one-vertex rooted tree has no children, so its root is a leaf despite having degree 0.

Example 6.23. Consider the tree on $\{a, b, c, d, e\}$ with edges $\{a, b\}, \{a, c\}, \{b, d\}, \{b, e\}$. Designate a as root, as in the left drawing. The right drawing shows the same edges with root c.



Root a



Root c

With root a, the hierarchy is

Vertex	Parent	Children	Depth
a	none	b, c	0
b	a	d, e	1
c	a	none	1
d	b	none	2
e	b	none	2

The rooted leaves are c, d, e, while a, b are internal vertices. Both internal vertices have two children, so this is a full binary tree. Its leaves need not all have the same depth.

Example 6.24. Now use root c in the right drawing. The vertex c has child a, a has child b, and b has children d, e. The depths are $0, 1, 2, 3, 3$ for c, a, b, d, e, respectively. The rooted leaves are now d, e.

The vertex c still has graph degree 1, but it has a child under this rooting. It is not a rooted leaf. The rooted tree is not full binary because c and a each have only one child. No edge or graph degree changed when the root changed.

The single-vertex rooted tree is also full binary: it has one leaf and no internal vertices. We will use such base objects when studying recursive structures in Chapter 12.

6.9 Exercises

6.1. Let $V = \{a, b, c\}$. Decide each statement using the simple-graph definition.

a) Can $\{a, a\}$ be an edge?

b) Can $\{a, b, c\}$ be an edge?

c) Are $\{a, b\}$ and $\{b, a\}$ different edges?

d) How many edges are in $\{\{a, b\}, \{b, a\}, \{b, c\}\}$?

6.2. Use this new graph H throughout the exercises that name it:

$$V_H = \{a, b, c, d, e\}, \qquad E_H = \{\{a, b\}, \{a, c\}, \{b, d\}, \{c, d\}, \{d, e\}\}.$$

a) Draw $H = (V_H, E_H)$.

b) Give every neighbour list and degree.

c) Verify the degree-sum formula using the degrees and the edge count.

6.3. Draw H again with different vertex positions, keeping the same labels. Explain how its edge set lets you check that the new picture represents exactly the same graph, even if some edges cross.

6.4. Suppose a graph has degrees $2, 2, 2, 1, 1$.

a) How many vertices and edges does it have?

b) Draw a connected graph with those degrees.

c) Draw a disconnected graph with the same degrees. What does this show about inferring connectedness from the degree list alone?

6.5. In H, decide whether each list is a walk. For every valid walk, give its length and say whether it is a path, a cycle or neither.

a) a, b, d, c, a.

b) a, b, d, b, a.

c) a, c, d, e.

d) a, d, e.

6.6. Start from H separately in each part.

a) Delete $\{a, b\}$, keeping every vertex. Is the remaining graph connected? Give paths that justify your answer.

b) Instead delete $\{d, e\}$. Identify the components and explain why no path joins them.

6.7. Add a new isolated vertex f to H without adding an edge. State the new numbers of vertices and edges, the degree of f and the number of components. Is the new graph connected?

6.8. Explain why each proposal is impossible for a simple graph.

a) Nine vertices, each of degree 3.

b) Five vertices with degrees $4, 4, 1, 1, 0$.

In part (b), the degree sum is even. Identify a different obstruction.

6.9. a) A report lists exactly three vertices of odd degree. Why can it not describe a graph?

b) Draw a graph with four vertices of odd degree.

c) Draw a graph with no vertices of odd degree. Explain why this does not contradict the even-number result.

6.10. Each graph below has vertex set $\{a, b, c, d\}$.

Graph	Edge set
T	$\{\{a, b\}, \{b, c\}, \{c, d\}\}$
J	$\{\{a, b\}, \{b, c\}, \{c, a\}\}$
F	$\{\{a, b\}, \{c, d\}\}$
Z	$\emptyset$

For each graph, determine which of the following properties hold: connected, contains a cycle, tree, forest. List its degree-one vertices.

6.11. Using H, refute the claim that every connected graph has exactly one path between each pair of distinct vertices. State two different paths with the same endpoints and identify the cycle they produce.

6.12. A student says that the endpoints of a longest path have degree 1 in every finite connected graph. Give a counterexample. Explain which alternative in the proof of Example 6.14 can occur when cycles are permitted.

6.13. Let T have vertices $\{a, b, c, d\}$ and edges $\{a, b\}, \{a, c\}, \{a, d\}$.

a) Show that T is a tree.

b) Delete d and its incident edge. Explain why the remaining graph is a tree.

c) Start again from T and delete a and all its incident edges. Is the remaining graph a tree? Identify its components.

6.14. Complete an induction proof that every tree with $n \geq 1$ vertices has $n - 1$ edges.

a) State the base case and the full inductive hypothesis, including which trees it covers.

b) For an arbitrary tree with $k + 1$ vertices, justify the existence of a leaf and explain why removing it leaves a tree.

c) Give the numbers of vertices and edges in the smaller tree, restore the deleted edge and state the conclusion.

6.15. a) How many edges does a tree with 12 vertices have?

b) Construct a graph with six vertices and five edges that is not a tree. Verify its count and explain which tree condition fails.

6.16. A tree has exactly four vertices of degree 3, and all its other vertices have degree 1. Find its number of leaves, vertices and edges. Then draw one such tree or specify its vertex and edge sets to show that these counts are attainable.

6.17. Give two different spanning trees of H by listing their edge sets and keeping all five vertices. Justify connectedness and absence of cycles for each. Explain why $\{d, e\}$ must be retained in every spanning tree of H.

6.18. a) Can a graph with eight vertices and six edges be connected? Give a mathematical reason.

b) A connected graph has eight vertices and seven edges. Must it be a tree? Justify your answer.

c) In the fixed six-vertex graph G from Example 6.1, how many edges must be omitted to obtain a spanning tree? Does every choice of that many edges work? Exhibit a choice that fails.

6.19. Let G be connected and let $\{u, v\}$ lie on a cycle. Prove that deleting this edge while retaining all vertices leaves a connected graph. Explain how an old path using the deleted edge can be replaced, including how to handle repeated vertices in the replacement route.

6.20. On $\{r, a, b, c, d, e, f\}$, take edges

$$\{r, a\}, \{r, b\}, \{a, c\}, \{a, d\}, \{b, e\}, \{b, f\}.$$

a) With root r, draw the hierarchy and give every parent and depth.

b) List its rooted leaves and internal vertices. Is it full binary?

c) Now use root a. Give the children of a and decide whether the rooted tree is full binary. Explain why the graph's edge count has not changed.

6.21. a) For the one-vertex rooted tree, give the degree of its root. Is it a rooted leaf? Is it a full binary tree?

b) Construct a rooted tree whose root has graph degree 1 but is not a rooted leaf. Identify all degree-one vertices and all rooted leaves.

6.22. Let T be a rooted tree with vertex set V, root r, and at least two vertices. Define $p : V \setminus \{r\} \to V$ to assign each non-root vertex its parent.

a) Explain why this is a function on its stated domain and why the root is excluded.

b) Must p be injective? Prove it or give an explicit rooted-tree counterexample.

c) What happens to depth each time p is applied? Explain why repeatedly moving to a parent eventually reaches the root.

6.23. Prove that two distinct siblings in a rooted tree cannot be adjacent. Identify the cycle that an edge between them would create.

6.24. A proposed edge-count proof starts with one vertex and repeatedly attaches a new vertex to an endpoint of the current path. It obtains n vertices and $n-1$ edges, then concludes that every tree has this count.

a) What has the construction failed to justify? Give a four-vertex tree with a shape it cannot produce.

b) Explain how beginning with an arbitrary tree and deleting a leaf addresses that problem.

6.25. Let T be a tree with at least two vertices and let $\{u, v\}$ be one of its edges. Prove that deleting this edge disconnects the graph. Use the unique-path property to explain why an alternative surviving path cannot exist.

6.26. Let u, v be distinct nonadjacent vertices of a tree T. Add the new edge $\{u, v\}$.

a) Explain why the new graph contains a cycle.

b) Prove that it has exactly one cycle, by considering which edge any new cycle must use and the unique old path between u and v.

6.27. a) Apply Kruskal's construction to H, examining its edges in the order

$$\{a, b\},\ \{b, d\},\ \{a, c\},\ \{c, d\},\ \{d, e\}.$$

Record each decision and the components after it. Give the resulting spanning tree.

b) Choose another examination order for H that produces a different spanning tree. Show your decisions.

c) Charlie accepts an edge only if at least one of its endpoints has not appeared in an accepted edge. He applies this rule to the graph with vertex set $\{a, b, c, d\}$, examining $\{a, b\}, \{c, d\}, \{b, c\}$ in that order. Does his rule produce a spanning tree? Explain how Kruskal's rule treats the last edge.

LECTURE NOTES
DISCRETE MATHEMATICS

CHAPTER 7

Equivalence relations



Updated: 19 November 2026

7 Equivalence relations

Suppose every student has one assigned default printer. Alice and Bob have the same default printer, and Bob and Charlie have the same default printer. We can conclude that Alice and Charlie do too. The assignment groups the students without ambiguity.

Now change the rule: two students are related when there is at least one printer they can both use. Alice may share one printer with Bob, while Bob shares a different printer with Charlie. Alice and Charlie need not share any printer. The new rule does not support the same conclusion.

Chapter 4 introduced relations as sets of ordered pairs and used transitivity to examine a two-step argument. Here we develop that language further: we reverse relations, combine them and test their properties. Chapter 6's drawings will help us see the pairs, with arrows to record direction. We will then identify the conditions under which a relation really does divide a set into separate groups.

What you should be able to do

After working through this chapter, you should be able to:

- specify a relation using pairs, a table, a rule or a directed diagram;
- find inverse relations and compute compositions using a middle element;
- test reflexivity, symmetry, antisymmetry and transitivity;
- give proofs of properties and complete counterexamples to failures;
- recognise equivalence relations and determine their classes;
- check a partition and explain its connection with equivalence relations.

7.1 Relations

An ordered pair (a, b) records a first entry and a second entry. Equality of ordered pairs requires equality in each position. Thus $(1, 2) \neq (2, 1)$, although the unordered sets $\{1, 2\}$ and $\{2, 1\}$ are equal.

❄ Definition 7.1 — Cartesian product and relation

The **Cartesian product** of sets A and B is

$$A \times B = \{(a, b) : a \in A \text{ and } b \in B\}.$$

A **relation from** A **to** B is a subset $R \subseteq A \times B$. A **relation on** A is a relation from A to itself, so $R \subseteq A \times A$.

We write aRb to mean $(a, b) \in R$.

The Cartesian product lists all allowed ordered pairs; the relation selects the pairs for which its rule holds. The sets A and B are part of the specification. A relation can leave some elements of A unrelated to anything in B, or relate one element to several others. A function from A to B has the additional requirement of exactly one output in B for every input in A.

Example 7.1. Let $A = \{\text{Alice}, \text{Bob}, \text{Charlie}\}$ be a set of users, and let $B = \{p, q\}$ be a set of folders. Define R by

$$R = \{(\text{Alice}, p), (\text{Alice}, q), (\text{Bob}, p)\}.$$

The pair (Alice, q) means that Alice can access folder q. It does not mean that q can access Alice. The six possible user–folder pairs belong to $A \times B$; only three belong to R.

The same data can be recorded in a table. A T entry means that the row element is related to the column element.

User	Folder p	Folder q
Alice	T	T
Bob	T	F
Charlie	F	F

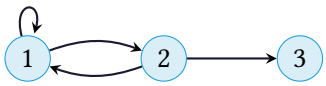
Here R is a relation from A to B, but not a function from A to B. Alice has two related folders, and Charlie has none. Each feature violates the exactly-one-output requirement.

Example 7.2. On $D = \{1, 2, 3\}$, let

$$T = \{(1,1), (1,2), (2,1), (2,3)\}.$$

In a **directed diagram**, draw an arrow from x to y exactly when $(x, y) \in T$. A pair (x, x) is drawn as a **loop** from a vertex back to itself.

T	1	2	3
1	T	T	F
2	T	F	T
3	F	F	F



There are two arrows between 1 and 2, one for each ordered pair. The arrow $2 \to 3$ has no reverse arrow because $(3, 2) \notin T$. Vertex 3 remains in the diagram even though no arrow starts there. Read a table entry from its row to its column; the entry in row 2, column 3 records $2T3$.

These diagrams allow directions and loops. Chapter 6's simple undirected graphs did not allow loops and used unordered pairs as edges. We are now using a different kind of diagram to represent ordered pairs.

7.2 Inverse relations and composition

Two useful operations ask different questions: what happens when we reverse a pair, and what happens when we follow one relation by another?

❄ Definition 7.2 — Inverse relation

If $R \subseteq A \times B$, its **inverse relation** is

$$R^{-1} = \{(b, a) : (a, b) \in R\} \subseteq B \times A.$$

Thus $bR^{-1}a$ holds exactly when aRb holds.

Example 7.3. For the permission relation in Example 7.1,

$$R^{-1} = \{(p, \text{Alice}), (q, \text{Alice}), (p, \text{Bob})\}.$$

The inverse relates a folder to the users who can access it. For example, pR^{-1}Bob because Bob $R\,p$. This reversal always defines a relation, even though it need not define a function. Folder p is related to two users. Reversing the pairs a second time gives $(R^{-1})^{-1} = R$. In a directed diagram on one set, inversion reverses every arrow; a loop remains a loop.

❄ Definition 7.3 — Composition of relations

Let $R \subseteq A \times B$ and $S \subseteq B \times C$. Their **composition** $S \circ R$ is the relation from A to C defined by

$$a(S \circ R)c \iff \exists b \in B\ (aRb \text{ and } bSc).$$

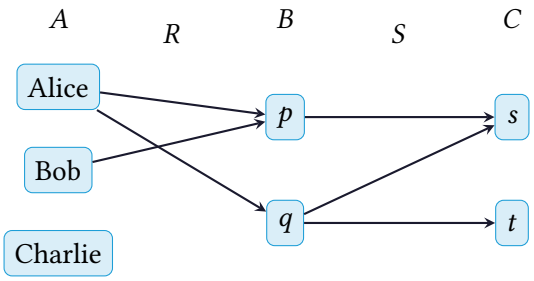
The same middle element b must satisfy both conditions.

In $S \circ R$, follow R first and S second. The symbol on the right is used first. A pair (a, c) belongs to the composition when there is at least one suitable middle element. Several middle elements can produce the same pair, but a set records that pair only once.

Example 7.4. Keep the sets A, B and permission relation R from Example 7.1. Let $C = \{s, t\}$ be a set of servers and define

$$S = \{(p, s), (q, s), (q, t)\}.$$

Here bSc means that folder b is stored on server c. The composition records user–server pairs connected through an accessible folder. It describes these given relations; it does not grant access to every folder on a server.



Alice reaches s through either p or q, and reaches t through q. Bob reaches s through p. Charlie has no first step in R. Consequently,

$$S \circ R = \{(\text{Alice}, s), (\text{Alice}, t), (\text{Bob}, s)\}.$$

The two witnesses for (Alice, s) do not create two copies of that pair. The pair (Bob, t) is absent: Bob's only accessible folder is p, and $(p, t) \notin S$.

Example 7.5. On $A = \{0, 1, 2\}$, let $R = \{(0, 1)\}$ and $S = \{(1, 2)\}$. For $S \circ R$, the middle element 1 connects $0R1$ to $1S2$, so

$$S \circ R = \{(0, 2)\}.$$

For $R \circ S$, we must follow S first. Its only arrow ends at 2, where no R-arrow starts. Hence $R \circ S = \emptyset$. Both compositions are defined on the stated sets, but they are different relations.

Example 7.6. For the permission relation R in Example 7.1, $R^{-1} \circ R$ is a relation on the user set A. A user first reaches an accessible folder; the inverse then reaches a user who can access that same folder. It therefore relates two users exactly when they share at least one accessible folder.

Folder p gives all four ordered pairs between Alice and Bob, including their self-pairs. Folder q adds no new pair, because only Alice can access it. Thus

$$R^{-1} \circ R = \{(\text{Alice}, \text{Alice}), (\text{Alice}, \text{Bob}),\\ (\text{Bob}, \text{Alice}), (\text{Bob}, \text{Bob})\}.$$

Charlie is not even related to himself, since he cannot access a folder. Composing a relation with its inverse need not produce just the pairs (a, a), or include every such pair.

7.3 Reflexivity, symmetry and antisymmetry

From now on, the properties concern a relation on one specified nonempty set A. Each property makes a claim about all the appropriate elements of A.

❄ Definition 7.4 — Reflexivity, symmetry and antisymmetry

A relation R on A is:

- **reflexive** if xRx for every $x \in A$;
- **symmetric** if, for all $x, y \in A$, $xRy \implies yRx$;
- **antisymmetric** if, for all $x, y \in A$, (xRy and yRx) $\implies x = y$.

In a directed diagram, reflexivity requires a loop at every vertex. Symmetry requires the reverse of every arrow. Antisymmetry permits no pair of opposite arrows between distinct vertices. Loops are allowed by antisymmetry: their two endpoints are already equal.

Example 7.7. The relation $R = \{(1,1),(2,2)\}$ is reflexive on $A = \{1,2\}$: both required self-pairs are present. The same set of pairs also defines a relation on $B = \{1,2,3\}$, but it is not reflexive on B, because $(3,3)$ is missing. One missing self-pair is enough to refute reflexivity; having some loops is insufficient to prove it.

Example 7.8. For T in Example 7.2, the pairs $(1,2)$ and $(2,1)$ reverse each other. However, $(2,3) \in T$ while $(3,2) \notin T$. This pair is a complete counterexample to symmetry. The loop at 1 is its own reverse, so it creates no problem for symmetry.

Reversing all pairs shows a useful test: R is symmetric if and only if $R^{-1} = R$. Indeed, symmetry puts every reversed pair back in R; and equality with the inverse ensures that every pair has its reverse.

Example 7.9. On $\mathbb{Z}$, let aRb mean $a \leq b$. If both aRb and bRa hold, then $a \leq b$ and $b \leq a$, which force $a = b$. This proves antisymmetry. It also has every self-pair, since $a \leq a$. It is not symmetric: $0 \leq 1$, but $1 \leq 0$ is false.

To refute antisymmetry, we would need two *distinct* elements related in both directions. A self-pair such as $(0,0)$ does not meet that requirement.

Example 7.10. Each row below specifies a relation on $A = \{1,2,3\}$.

Relation	Symmetric?	Antisymmetric?
$I = \{(1,1),(2,2),(3,3)\}$	T	T
$J = \{(1,2),(2,1)\}$	T	F
$K = \{(1,2)\}$	F	T
$L = \{(1,2),(2,1),(2,3)\}$	F	F

In I, every pair is its own reverse, and no distinct elements are related both ways. In J, every reverse is present, but 1 and 2 violate antisymmetry. In K, $(2,1)$ is missing, so symmetry fails; there is no distinct pair

related both ways, so antisymmetry holds. Finally, L violates symmetry at $(2, 3)$ and violates antisymmetry at $1, 2$.

✍ Antisymmetric does not mean "not symmetric"

The two properties have separate definitions. A relation can have both, one, or neither. To disprove antisymmetry, exhibit $x \neq y$ with both xRy and yRx.

7.4 Transitivity

❄ Definition 7.5 — Transitivity

A relation R on A is **transitive** if, for all $x, y, z \in A$,

$$(xRy \text{ and } yRz) \Longrightarrow xRz.$$

The elements x, y, z need not be distinct.

The middle element must match in the two premises. To refute the property, give x, y, z for which both premises hold and the conclusion fails. To prove it, show that every two-step chain has the required direct pair. The direct pair may be a loop.

Example 7.11. In T from Example 7.2, $1T2$ and $2T3$ hold, but $1T3$ does not. Thus T is not transitive. There is a directed two-step route from 1 to 3, but the relation does not contain $(1, 3)$. Transitivity requires that pair itself, rather than only a route connecting its entries.

Example 7.12. On $\{a, b\}$, let $R = \{(a, b), (b, a)\}$. The relation is symmetric, but it is not transitive. Taking $x = a$, $y = b$ and $z = a$ gives aRb and bRa, while aRa is false. The other two-step chain would likewise require (b, b).

There are no three distinct elements in this set. Checking only triples of distinct elements would miss both failures.

Example 7.13. On $A = \{1, 2, 3\}$, let $R = \{(1, 2)\}$. For the first premise xRy to hold, we must have $x = 1$ and $y = 2$. But no pair of R starts at 2, so yRz cannot also hold. There is no triple making both premises true. The transitivity implication is therefore true for every triple, and R is transitive.

The empty relation on this nonempty set is also transitive. It is symmetric and antisymmetric for the same logical reason: none of the relevant premises holds. It is not reflexive, because all required self-pairs are missing. These are applications of Chapter 2's truth rule for an implication with a false premise.

Example 7.14. Let R on $\mathbb{Z}$ be given by aRb exactly when $a \leq b$. Take arbitrary $a, b, c \in \mathbb{Z}$ and suppose aRb and bRc. Then $a \leq b$ and $b \leq c$, so $a \leq c$, which is aRc. Thus R is transitive.

Together with the earlier checks, $\leq$ is reflexive, antisymmetric and transitive, and is not symmetric. These properties do not all have to occur together.

7.5 Equivalence relations

For a grouping rule, we want each object related to itself, the relationship to work in both directions, and two successive relationships to keep us in the same group.

❄ Definition 7.6 — Equivalence relation

A relation on a set is an **equivalence relation** if it is reflexive, symmetric and transitive. Related elements are then called **equivalent** under that relation.

Equivalent objects need not be equal. The relation specifies which feature is being treated as the same.

Example 7.15. Let $U = \{\text{Alice}, \text{Bob}, \text{Charlie}, \text{David}, \text{Eve}\}$ and let $d : U \to \{p_1, p_2, p_3\}$ assign each student exactly one default printer. Define

$$uRv \iff d(u) = d(v).$$

This extends Chapter 4's transitivity example.

Reflexivity. For any $u \in U$, $d(u) = d(u)$, so uRu.

Symmetry. If uRv, then $d(u) = d(v)$. Reversing the equality gives $d(v) = d(u)$, so vRu.

Transitivity. If uRv and vRw, then $d(u) = d(v)$ and $d(v) = d(w)$. Hence $d(u) = d(w)$, so uRw.

Thus R is an equivalence relation. For a concrete assignment, use the following table.

Student	Default printer
Alice	p_1
Bob	p_1
Charlie	p_2
David	p_2
Eve	p_3

Alice and Bob are equivalent under R, although they are different students. The proof works for any fixed assignment function d, not just the five values in this table. More generally, equality of the values of any fixed function defines an equivalence relation on its input set.

Example 7.16. On $\mathbb{Z}$, define aRb by $|a - b| \leq 1$, where $|a - b|$ is the distance between the integers. This relation is reflexive, since $|a - a| = 0$. It is symmetric, since $|b - a| = |a - b|$. However, $0R1$ and $1R2$ hold while $0R2$ fails: $|0 - 2| = 2 > 1$. It is not transitive and therefore is not an equivalence relation.

Example 7.17. Suppose Alice can use only p, Bob can use p and q, and Charlie can use only q. On these three students, define uRv when they can both use at least one printer. Every student is related to themselves, because each has an available printer. The relation is symmetric: a printer shared by u, v is also shared by v, u.

Yet Alice R Bob through p, and Bob R Charlie through q, while Alice R Charlie is false. The two premises use different printers. This relation is not transitive. It is the shared-output construction of Example 7.6, with a different permission assignment.

Example 7.18. On $\mathbb{Z}$, define

$$aRb \iff a - b = 2k \text{ for some } k \in \mathbb{Z}.$$

In other words, the difference is even.

Reflexivity. For any $a \in \mathbb{Z}$, $a - a = 0 = 2(0)$, so aRa.

Symmetry. Suppose aRb, so $a - b = 2k$ for an integer k. Then $b - a = 2(-k)$. Since $-k \in \mathbb{Z}$, this gives bRa.

Transitivity. Suppose aRb and bRc. For some integers j, k, we have $a - b = 2j$ and $b - c = 2k$. Adding gives

$$a - c = (a - b) + (b - c) = 2(j + k).$$

Since $j + k \in \mathbb{Z}$, we obtain aRc. All three properties hold, so R is an equivalence relation. Each step used Chapter 3's integer-witness method; no finite list of particular integers could check the whole infinite domain.

7.6 Equivalence classes

❄ **Definition 7.7 — Equivalence class**

Let R be an equivalence relation on A and let $a \in A$. The **equivalence class** of a is the set

$$[a]_R = \{x \in A : aRx\}.$$

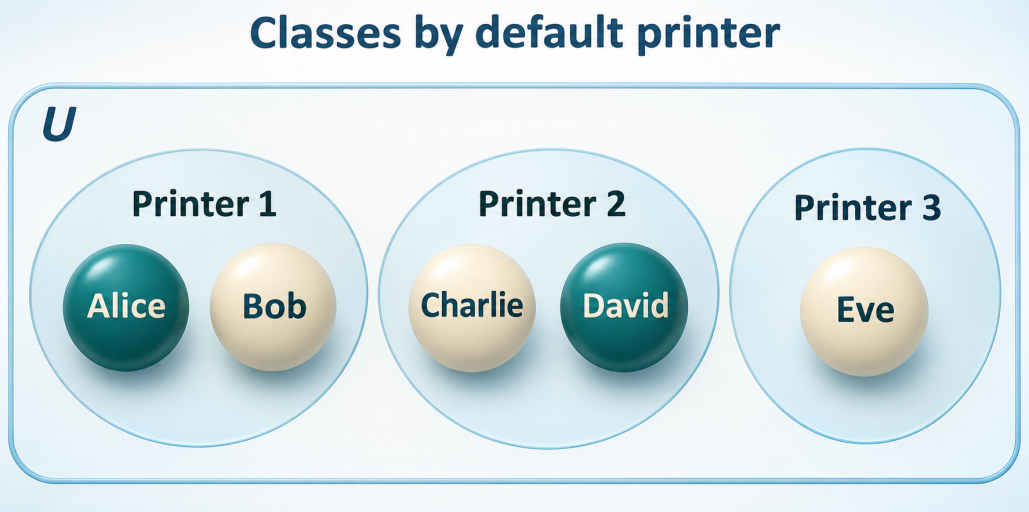
We often write $[a]$ when the relation is clear. The element a is called a **representative** of its class.

The brackets describe a set of elements, not one element or a new name for a alone. To find $[a]$, determine every x in the specified set that satisfies aRx. Reflexivity guarantees that a itself belongs to $[a]$.

Example 7.19. For the printer assignment in Example 7.15,

$$\begin{aligned} [\text{Alice}] &= [\text{Bob}] = \{\text{Alice}, \text{Bob}\}, \\ [\text{Charlie}] &= [\text{David}] = \{\text{Charlie}, \text{David}\}, \\ [\text{Eve}] &= \{\text{Eve}\}. \end{aligned}$$

For instance, [Alice] consists of all students whose default printer equals p_1, namely Alice and Bob. There are five students but only three distinct classes. The equality [Alice] = [Bob] is an equality of sets; it does not identify the two students as one person. Either is a representative of the same class.



Five students form three classes: Alice and Bob belong to the same class, as do Charlie and David; Eve forms a singleton class.

Python (non-examinable)

Here a dictionary records each student's assigned printer. In `printer[student]`, square brackets look up the printer for that student. For each printer label, we start an empty list `members`; append adds a student to that list when the two printer labels agree. Python uses `==` to test equality.

```
printer = {
    "Alice": "p1", "Bob": "p1",
    "Charlie": "p2", "David": "p2",
    "Eve": "p3"
}
for label in ["p1", "p2", "p3"]:
    members = []
    for student in printer:
        if printer[student] == label:
            members.append(student)
    print(label, members)
```

Output.

```
p1 ['Alice', 'Bob']
p2 ['Charlie', 'David']
p3 ['Eve']
```

The three lists contain the three equivalence classes already found. Lists preserve a display order; that order does not form part of the mathematical sets. Each student appears once because each has one assigned printer.

Example 7.20. For the even-difference relation in Example 7.18, $0Rx$ exactly when $0 - x = -x$ is even, which is equivalent to x being even. Likewise, $1Rx$ exactly when x is odd: for an integer k, $1 - x = 2k$ is equivalent to $x = 2(-k) + 1$. Hence

$$[0] = \{2k : k \in \mathbb{Z}\}, \qquad [1] = \{2k + 1 : k \in \mathbb{Z}\}.$$

These are all the distinct classes: every integer is even or odd. Every even integer represents $[0]$; every odd integer represents $[1]$. Thus $[2] = [0]$ and $[-3] = [1]$, even though $2 \neq 0$ and $-3 \neq 1$.

7.7 Partitions

❄ Definition 7.8 — Partition

A **partition** $\mathcal{P}$ of a nonempty set A is a collection of subsets of A, called **blocks**, such that:

- every block is nonempty;
- every element of A lies in a block;
- two distinct blocks are disjoint.

Thus every element of A belongs to exactly one block.

A partition is a set whose elements are themselves sets. The blocks cover A, and no element is shared between different blocks.

Example 7.21. Let $A = \{1, 2, 3, 4, 5\}$. The collection

$$\mathcal{P} = \{\{1, 4\}, \{2, 3\}, \{5\}\}$$

is a partition: its three blocks are nonempty, together contain all five elements, and do not overlap. Here $1 \in \{1, 4\}$ and $\{1, 4\} \in \mathcal{P}$. The number 1 is an element of a block, while the block itself is an element of the partition.

Three changes illustrate separate failures. The collection $\{\{1, 2\}, \{2, 3\}, \{4, 5\}\}$ fails because 2 belongs to two distinct blocks. The collection $\{\{1, 2\}, \{3, 4\}\}$ fails because it omits 5. The collection $\{A, \emptyset\}$ fails because it includes an empty block. In contrast, $\{A\}$ is a valid partition with one block, and $\{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}\}$ is a valid partition with five blocks.

Example 7.22. Let R be an equivalence relation on A. We first prove

$$aRb \iff [a] = [b].$$

Proof. Suppose aRb. If $x \in [a]$, then aRx; symmetry gives bRa, so transitivity gives bRx. Thus $[a] \subseteq [b]$. If $x \in [b]$, then bRx; together with aRb, transitivity gives aRx. Thus $[b] \subseteq [a]$. The two inclusions prove $[a] = [b]$.

Conversely, suppose $[a] = [b]$. Reflexivity gives $b \in [b]$, hence $b \in [a]$. By the class definition, aRb. □

Now suppose $[a]$ and $[b]$ have an element x in common. Then aRx and bRx. Symmetry changes the latter to xRb, and transitivity gives aRb. The result just proved yields $[a] = [b]$. Therefore two distinct equivalence classes cannot overlap.

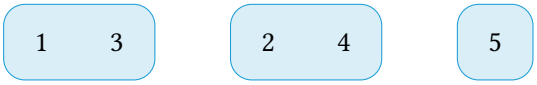
Example 7.23. Let R be an equivalence relation on a nonempty set A. Collect its *distinct* equivalence classes. We check the three partition requirements.

Each class $[a]$ is nonempty because aRa puts a in $[a]$. Every $x \in A$ is covered because $x \in [x]$. Finally, Example 7.22 shows that two distinct classes are disjoint. Thus the classes form a partition of A.

For the printer assignment, listing Alice's and Bob's classes does not create two blocks: they are the same set. The partition has one block for each distinct class.

Example 7.24. Let $A = \{1, 2, 3, 4, 5\}$ and take a different partition

$$\mathcal{P} = \{\{1, 3\}, \{2, 4\}, \{5\}\}.$$



Define xRy when x and y belong to the same block. The first block supplies $(1, 1), (1, 3), (3, 1), (3, 3)$. The second supplies $(2, 2), (2, 4), (4, 2), (4, 4)$. The singleton block supplies $(5, 5)$. Consequently,

$$\begin{aligned} R = \{&(1, 1), (1, 3), (3, 1), (3, 3),\\ &(2, 2), (2, 4), (4, 2), (4, 4), (5, 5)\}. \end{aligned}$$

Self-pairs are included: an element is in the same block as itself. There is no pair between distinct blocks, so, for example, $(1, 2) \notin R$. Its classes are $[1] = [3] = \{1, 3\}$, $[2] = [4] = \{2, 4\}$ and $[5] = \{5\}$.

Example 7.25. For a partition $\mathcal{P}$ of A, define xRy when x, y belong to the same block.

Reflexivity. Every $x \in A$ lies in a block, so it lies in the same block as itself. Hence xRx.

Symmetry. If x, y lie in the same block, then y, x lie in that block too. Thus xRy implies yRx.

Transitivity. Suppose xRy and yRz. The elements x, y lie together in a block B, and y, z lie together in a block C. Both blocks contain y. Since distinct blocks are disjoint, $B = C$. Therefore x, z lie in that same block, so xRz.

Thus R is an equivalence relation. For any $a \in A$, let B be its block. An element x satisfies aRx exactly when $x \in B$, so $[a] = B$. Its distinct classes recover the original partition.

Conversely, if we start with an equivalence relation, form its classes and then relate elements in the same class, we recover the original relation: Example 7.22 gives aRb exactly when $[a] = [b]$. The relation records the related pairs; the partition records the groups those pairs determine.

7.8 Exercises

7.1. Let $A = \{0, 1\}$, $B = \{p, q, r\}$ and $R = \{(0, p), (0, r)\}$.

a) List $A \times B$ and identify which pairs belong to R.

b) Is R a relation from A to B? Is it a function from A to B? Explain both answers.

c) Could a relation from A to B contain $(p, 0)$? Explain using the specified sets.

7.2. Use this relation H in every exercise that names it:

$$A = \{a, b, c\}, \qquad H = \{(a, a), (a, b), (b, c), (c, b), (c, c)\}.$$

a) Draw its directed diagram, including every vertex and loop.

b) Make a table with rows and columns a, b, c, using T and F to record membership.

c) Which arrows start at b? Which end at b?

7.3. On $A = \{0, 1, 2\}$, let $R = \{(0, 1), (1, 2), (2, 2)\}$.

a) Find R^{-1} and then $(R^{-1})^{-1}$.

b) Does R assign exactly one output to every element of A? Does R^{-1}? Justify your answers from the pairs.

c) What happens to the loop at 2 under inversion?

7.4. Let $A = \{\text{Alice}, \text{Bob}, \text{Eve}\}$, $B = \{x, y\}$ and $C = \{p, q\}$. Define

$$R = \{(\text{Alice}, x), (\text{Bob}, x), (\text{Bob}, y), (\text{Eve}, y)\},$$
$$S = \{(x, p), (y, p), (y, q)\}.$$

Find $S \circ R$. For each resulting pair, give a middle element witnessing membership. Which pair has two different middle-element witnesses?

7.5. On $A = \{0, 1, 2, 3\}$, let $R = \{(0, 1), (1, 2)\}$ and $S = \{(2, 3)\}$. Find $S \circ R$ and $R \circ S$. Explain why both expressions are defined and why their values need not agree.

7.6. For H in Exercise 7.2, find $H^{-1} \circ H$ by determining which elements have an H-output in common. List all its pairs. Is this composition reflexive, symmetric, antisymmetric or transitive on $\{a, b, c\}$? Support each decision with a proof or counterexample.

7.7. Let $R = \{(0, 0), (1, 1)\}$. Is R reflexive on $\{0, 1\}$? Is it reflexive on $\{0, 1, 2\}$? Explain why these answers can differ although no pair in R changed.

7.8. On $A = \{a, b, c\}$, give a relation with each combination below, and justify its two properties.

a) Symmetric and antisymmetric.

b) Symmetric, not antisymmetric.

c) Antisymmetric, not symmetric.

d) Neither symmetric nor antisymmetric.

7.9. For $I = \{(a, a), (b, b), (c, c)\}$ on $\{a, b, c\}$, David says that (a, a) violates antisymmetry because it gives an arrow in both directions. Identify the error and prove the correct conclusion.

7.10. On $A = \{a, b, c\}$, let $R = \{(a, b), (b, a)\}$.

a) Give two triples witnessing failures of transitivity.

b) Add as few pairs as possible to obtain a transitive relation on A. Prove that your new relation is transitive and explain why fewer additions cannot work.

c) Is the new relation reflexive on A?

7.11. On $A = \{0, 1, 2\}$, consider $R = \{(0, 2)\}$.

a) Decide all four properties: reflexive, symmetric, antisymmetric and transitive. Explain the implications whose premises never hold.

b) Repeat for the empty relation on A.

7.12. For each relation on $\mathbb{Z}$, decide whether it is reflexive, symmetric, antisymmetric and transitive. Prove each successful property and give explicit witnesses for each failed one.

a) aRb means $a \le b$.

b) aRb means $a < b$.

c) aRb means $a + b = 0$.

7.13. Let F be a nonempty set of files, let L be a set of labels, and let $\ell : F \to L$ be a function. Define fRg by $\ell(f) = \ell(g)$. Prove that R is an equivalence relation. For a fixed file f, describe $[f]_R$ using ℓ.

7.14. On $A = \{a, b, c\}$, construct a relation that is:

a) reflexive and symmetric, but not transitive;

b) reflexive and transitive, but not symmetric;

c) symmetric and transitive, but not reflexive.

Verify the two claimed properties and give a complete counterexample to the third in each part.

7.15. On $\mathbb{Z}$, define aRb when $a + b$ is even. Prove that this is an equivalence relation and determine its distinct equivalence classes. In the transitivity proof, start with $a + b = 2j$ and $b + c = 2k$ and express $a + c$ as twice an integer.

7.16. Five files a, b, c, d, e have the following sizes in a fixed unit.

File	a	b	c	d	e
Size	2	3	2	5	3

Relate two files exactly when their sizes are equal. List $[a], [b], [c], [d], [e]$, then list the distinct classes. Does $[a] = [c]$ imply $a = c$? Explain.

7.17. Which collections are partitions of $A = \{0, 1, 2, 3, 4, 5\}$? For each failure, identify the exact requirement that fails.

a) $\{\{0, 2, 4\}, \{1, 3, 5\}\}$.

b) $\{\{0, 1, 2\}, \{2, 3\}, \{4, 5\}\}$.

c) $\{\{0, 1\}, \{2, 3\}, \{4\}\}$.

d) $\{\{0, 1, 2, 3, 4, 5\}, \emptyset\}$.

7.18. Let $U = \{\text{Alice}, \text{Bob}, \text{Charlie}, \text{David}, \text{Eve}\}$ and

$$\mathcal{P} = \{\{\text{Alice}, \text{Bob}\}, \{\text{Charlie}\}, \{\text{David}, \text{Eve}\}\}.$$

Mark each statement TRUE or FALSE and explain your answer.

a) $\text{Alice} \in \mathcal{P}$.

b) $\text{Alice} \in \{\text{Alice}, \text{Bob}\}$.

c) $\{\text{Alice}, \text{Bob}\} \in \mathcal{P}$.

d) $\{\text{Alice}, \text{Bob}\} \subseteq U$.

7.19. Let R be an equivalence relation. Suppose $c \in [a]_R$ and $c \in [b]_R$. Prove aRb, naming the properties used at each step. Then explain why $[a]_R = [b]_R$.

7.20. For $A = \{a, b, c, d, e\}$ and $\mathcal{P} = \{\{a, d\}, \{b, c\}, \{e\}\}$, define xRy when x, y lie in the same block.

a) List every ordered pair in R.

b) Explain why each self-pair must be included.

c) Give $[a]_R, [d]_R, [e]_R$ and explain why different representatives can name the same block.

7.21. Start with the partition $\{\{a, b\}, \{c, d\}, \{e\}\}$ of $\{a, b, c, d, e\}$ and its same-block relation. Merge its first two blocks, obtaining $\{\{a, b, c, d\}, \{e\}\}$. List the ordered pairs added to the relation. Must any old pair be removed? Explain using the two same-block rules.

7.22. The collection $\{\{a, b\}, \{b, c\}\}$ covers $\{a, b, c\}$. Define xRy when some set in this collection contains both x and y. Prove reflexivity and symmetry, but refute transitivity. Identify the step in Example 7.25 that the overlapping sets prevent.

7.23. For a relation R on A, prove that R is symmetric if and only if $R^{-1} = R$. In each direction, use the definition of the inverse to justify membership of the required pairs.

7.24. For a relation R on A, prove

$$R \text{ is transitive} \iff R \circ R \subseteq R.$$

Translate membership of $R \circ R$ into a statement about a middle element, then prove both directions.

7.25. Let $R \subseteq A \times B$ and $S \subseteq B \times C$. Prove

$$(S \circ R)^{-1} = R^{-1} \circ S^{-1}.$$

Begin with an arbitrary pair $(c, a) \in C \times A$ and show that it belongs to one side exactly when it belongs to the other.

7.26. Let G be a finite simple undirected graph with vertex set V. Define uRv when G has a path from u to v.

a) Prove reflexivity and symmetry. Include the length-0 path.

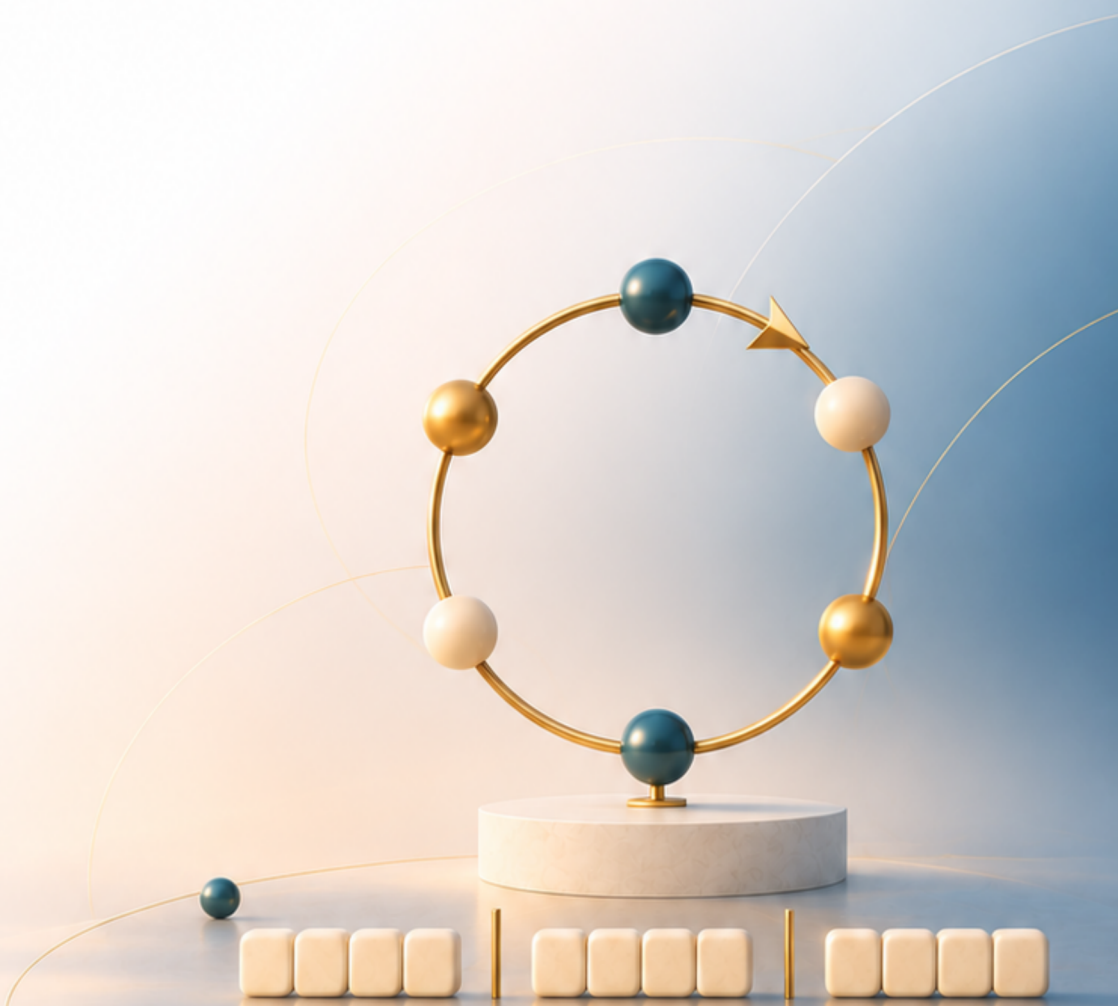
b) Prove transitivity. Joining two paths may produce a walk with repeated vertices; explain why a path with the same endpoints still exists.

c) Identify the equivalence classes of R. What does the partition look like when G is connected?

LECTURE NOTES
DISCRETE MATHEMATICS

CHAPTER 8

Modular arithmetic



Updated: 19 November 2026

8 Modular arithmetic

Alice, Bob, Charlie and David take turns checking a shared printer, in that order, then the cycle repeats. Number their positions 0, 1, 2, 3. Starting with Alice at turn 0, who checks at turn 17? Listing all the turns would work, but four complete cycles use sixteen turns and return to Alice. One further turn reaches Bob. The remainder after division by four identifies the position in the cycle.

Chapter 3 used integer multiples in proofs; Chapter 7 grouped integers according to an even difference. We now develop remainders systematically and group integers by the same remainder. These groups support arithmetic on a finite set of labels, even though the integers themselves extend without bound.

What you should be able to do

After working through this chapter, you should be able to:

- justify divisibility claims using integer witnesses;
- find a quotient and remainder, including for negative integers;
- interpret congruence using remainders and divisibility;
- describe residue classes and their partition of the integers;
- calculate with congruences and diagnose invalid cancellation;
- specify modular addition and multiplication as binary operations;
- check closure and identify two-sided identities in small systems.

8.1 Divisibility

We work with integers, $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$. As in Chapter 3, divisors in divisibility statements are nonzero. The integer being divided may be zero.

❄ Definition 8.1 — Divisibility

For $d, a \in \mathbb{Z}$ with $d \neq 0$, write $d \mid a$ when

$$a = dk \quad \text{for some } k \in \mathbb{Z}.$$

We say that d **divides** a, that d is a **divisor** of a, or that a is a **multiple** of d. Write $d \nmid a$ when no such integer k exists.

The symbol $\mid$ states a relation between two integers; it does not denote the numerical quotient. The order matters: the number on the left is the divisor.

Example 8.1. We have $4 \mid 28$ because $28 = 4(7)$. The witness is 7. However, $4 \nmid 30$: the equation $30 = 4k$ forces $k = 15/2$, which is not an integer. Writing $30 = 4(15/2)$ is an equality of numbers, but it does not establish integer divisibility.

Also, $28 \nmid 4$. The claim $4 \mid 28$ cannot be reversed just by reversing the notation.

Example 8.2. Since $-24 = 6(-4)$, we have $6 \mid -24$. Since $24 = (-6)(-4)$, we also have $-6 \mid 24$. Neither the divisor nor its integer multiplier has to be positive.

For every nonzero integer d, $0 = d(0)$ gives $d \mid 0$. In particular, $1 \mid 0$, just as 1 divides every other integer. We do not use zero as a divisor.

Example 8.3. Let $d \neq 0$, and suppose $d \mid a$ and $d \mid b$. Prove that $d \mid (a - 2b)$.

Proof. There are integers j, k such that $a = dj$ and $b = dk$. Consequently,

$$a - 2b = dj - 2dk = d(j - 2k).$$

The multiplier $j - 2k$ is an integer. This is the required divisibility witness. □

The argument applies to all integers in the hypotheses. It also explains a useful general rule: integer combinations of multiples of d remain multiples of d.

8.2 Quotients and remainders

Exact divisibility does not always hold, but we can separate an integer into a multiple of a positive divisor and a bounded remainder. We use the following elementary result, often called the **division algorithm**.

❄ Definition 8.2 — Quotient and remainder

For each integer a and each positive integer n, there are unique integers q, r such that

$$a = nq + r, \qquad 0 \le r < n.$$

The integer q is the **quotient** and r the **remainder** when a is divided by n. We write $a \bmod n = r$ for this chosen remainder.

The bounds on r are part of the description. They choose the multiple nq at or immediately below a, so that $nq \le a < n(q+1)$. The distance from nq to a is the remainder. For a fixed positive n, $n \mid a$ exactly when that remainder is zero.

Example 8.4. A batch contains 47 records, grouped in packets of 6. Since

$$47 = 6(7) + 5, \qquad 0 \le 5 < 6,$$

there are seven complete packets and five records left. Thus $47 \bmod 6 = 5$ and $6 \nmid 47$. For comparison, $48 = 6(8) + 0$, so $48 \bmod 6 = 0$ and $6 \mid 48$.

Example 8.5. To divide -13 by 5, locate it between the consecutive multiples -15 and -10:

$$-13 = 5(-3) + 2, \qquad 0 \le 2 < 5.$$

The quotient is -3 and the remainder is 2.



The equality $-13 = 5(-2) - 3$ is also correct, but its final summand -3 is not the chosen remainder: it violates $0 \le r < 5$. We move to the preceding multiple of 5 and add 2.

Python (non-examinable)

For integer inputs and a positive divisor, Python's `divmod` returns the quotient and remainder with the bounds used here. The assignment `q, r = divmod(a, n)` stores the first returned value in `q` and the second in `r`.

```
a = -13
n = 5
q, r = divmod(a, n)
print(q, r)
print(n * q + r)
```

Output.

```
-3 2
-13
```

Thus $q = -3$ and $r = 2$, and the second line reconstructs $5(-3) + 2 = -13$. The positive-divisor condition matters: it ensures that the remainder lies between 0 and $n - 1$.

Example 8.6. Bob writes $29 = 6(3) + 11$ and calls 11 the remainder. The equality holds, but $11 \geq 6$. Move one further group of six into the quotient:

$$29 = 6(4) + 5.$$

Now $0 \leq 5 < 6$. Similarly, $-12 = 6(-2) + 0$ already has a valid remainder: a negative input can be an exact multiple.

Uniqueness of the remainder. Suppose $a = nq + r = nq' + r'$ with both remainders between 0 and $n - 1$. Then $n(q - q') = r' - r$. The right-hand side lies strictly between $-n$ and n. The only multiple of the positive integer n in that interval is 0. Thus $r' = r$, and then $q' = q$. Without the bounds, many decompositions would be possible.

8.3 Congruence

From now on our modulus n is an integer with $n \geq 2$. The quotient-and-remainder result also allows $n = 1$, but our modular systems will have at least two labels.

❄ Definition 8.3 — Congruence modulo n

Integers a, b are **congruent modulo** n if they have the same remainder on division by n. Write

$$a \equiv b \pmod{n}.$$

If their remainders differ, write $a \not\equiv b \pmod{n}$. The positive integer n is the **modulus**.

Example 8.7. We have

$$17 = 5(3) + 2, \qquad 7 = 5(1) + 2, \qquad -8 = 5(-2) + 2.$$

Therefore $17 \equiv 7 \pmod 5$ and $-8 \equiv 7 \pmod 5$. The three integers are distinct; their remainders on division by 5 are equal. In particular, the congruence $17 \equiv 7 \pmod 5$ does not assert $17 = 7$.

Example 8.8. For integers a, b and fixed $n \geq 2$, we claim

$$a \equiv b \pmod n \iff n \mid (a - b).$$

Proof. First suppose the remainders agree. Write $a = nq + r$ and $b = nt + r$. Then $a - b = n(q - t)$, an integer multiple of n.

Conversely, suppose $a - b = nk$ for some integer k. Write $b = nt + r$ with $0 \leq r < n$. Then

$$a = b + nk = n(t + k) + r.$$

This is a valid quotient-and-remainder description of a with the same remainder r. Uniqueness of the remainder gives $a \equiv b \pmod n$. □

Equivalently, $a \equiv b \pmod n$ means $a = b + nk$ for some $k \in \mathbb{Z}$. This form turns a congruence into an ordinary integer equation with a witness.

Example 8.9. We have $23 \equiv -1 \pmod 6$, because $23 - (-1) = 24 = 6(4)$. Nevertheless, $23 \bmod 6 = 5$, not -1. The chosen remainder must lie in $\{0, 1, 2, 3, 4, 5\}$. Both 23 and -1 have that same remainder 5.

Thus $\equiv$ compares two integers, whereas $a \bmod n$ names one particular integer in $\{0, \ldots, n - 1\}$.

Example 8.10. The difference $17 - 7 = 10$ is divisible by 5, so $17 \equiv 7 \pmod 5$. It is not divisible by 4, so $17 \not\equiv 7 \pmod 4$. Stating that two integers are congruent without specifying a modulus leaves the relation unspecified.

8.4 Residue classes

Congruence generalises Chapter 7's even-difference relation. Modulo 2, equal remainders mean equal parity. For any fixed modulus, the same proof pattern establishes an equivalence relation.

Example 8.11. Fix $n \geq 2$ and let the domain be $\mathbb{Z}$. We use the divisibility test in Example 8.8.

Reflexivity. For every a, $a - a = 0 = n(0)$, so $a \equiv a \pmod{n}$.

Symmetry. If $a \equiv b \pmod{n}$, write $a - b = nk$ with $k \in \mathbb{Z}$. Then $b - a = n(-k)$, so $b \equiv a \pmod{n}$.

Transitivity. If $a \equiv b \pmod{n}$ and $b \equiv c \pmod{n}$, write $a - b = nj$ and $b - c = nk$ for integers j, k. Adding gives

$$a - c = (a - b) + (b - c) = n(j + k),$$

so $a \equiv c \pmod{n}$. All three properties hold on the whole domain.

❄ Definition 8.4 — Residue class

For $a \in \mathbb{Z}$, its **residue class modulo** n is

$$[a]_n = \{x \in \mathbb{Z} : x \equiv a \pmod{n}\} = \{a + nk : k \in \mathbb{Z}\}.$$

It is also called a **remainder class**. The integer a is a **representative** of the class.

The symbol $[a]_n$ denotes a set of integers. It is not the same object as the integer a. The subscript records which equivalence relation is being used.

Example 8.12. The classes modulo 4 are:

Chosen representative	Residue class
0	$[0]_4 = \{\ldots, -8, -4, 0, 4, 8, \ldots\}$
1	$[1]_4 = \{\ldots, -7, -3, 1, 5, 9, \ldots\}$
2	$[2]_4 = \{\ldots, -6, -2, 2, 6, 10, \ldots\}$
3	$[3]_4 = \{\ldots, -5, -1, 3, 7, 11, \ldots\}$

Every integer has exactly one remainder in $\{0, 1, 2, 3\}$. Consequently, every integer belongs to one of these classes and cannot belong to two different ones. The classes are nonempty, disjoint and together cover $\mathbb{Z}$: they form a partition, in Chapter 7's terminology. Each class is infinite, although there are only four classes.

Example 8.13. Since $-1 \equiv 3 \equiv 7 \pmod{4}$, we have

$$[-1]_4 = [3]_4 = [7]_4.$$

For instance, an integer differs from 7 by a multiple of 4 exactly when it differs from 3 by a multiple of 4: the two differences themselves differ by 4. The integers $-1, 3, 7$ are different representatives of one class; they do not name three new classes.

In general, there are exactly n distinct classes modulo n, represented once each by $0, \dots, n-1$. We write

$$\mathbb{Z}_n = \{[0]_n, [1]_n, \dots, [n-1]_n\}$$

for this set of classes. To keep computations short, we will also use the unique representatives as labels.

8.5 Arithmetic with congruences

Suppose $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$, with the same modulus. Then

$$\begin{aligned} a + c &\equiv b + d \pmod{n}, \\ a - c &\equiv b - d \pmod{n}, \\ ac &\equiv bd \pmod{n}. \end{aligned}$$

These rules allow us to replace an integer by a congruent one inside a sum, difference or product.

Write $a - b = nj$ and $c - d = nk$ for integers j, k. The differences of the respective sums and differences are $n(j+k)$ and $n(j-k)$, hence are multiples of n. For products,

$$ac - bd = (a-b)c + b(c-d) = n(jc + bk).$$

Again the multiplier is an integer. The divisibility test therefore gives each congruence. Repeated multiplication also permits positive integer powers: if $a \equiv b \pmod{n}$, then $a^m \equiv b^m \pmod{n}$ for every integer $m \geq 1$.

Example 8.14. Modulo 5, we have $38 \equiv 3$ and $27 \equiv 2$. Therefore

$$38 + 27 \equiv 3 + 2 = 5 \equiv 0 \pmod{5},$$

and

$$38(27) \equiv 3(2) = 6 \equiv 1 \pmod{5}.$$

The sum has remainder 0 and the product has remainder 1. The ordinary product is 1026, but computing that full product was unnecessary for the remainder.

Example 8.15. Using the same inputs modulo 5,

$$38 - 27 \equiv 3 - 2 = 1 \pmod{5}.$$

In the reverse order,

$$27 - 38 \equiv 2 - 3 = -1 \equiv 4 \pmod{5}.$$

The last step uses $-1 = 5(-1) + 4$. Leaving the answer as a congruence to -1 is valid; the remainder itself is 4.

Example 8.16. Since $3^2 = 9 \equiv -1 \pmod{5}$,

$$3^8 = (3^2)^4 \equiv (-1)^4 = 1 \pmod{5}.$$

Thus 3^8 has remainder 1 when divided by 5. We used a small congruent value for 3^2 and then multiplied it four times. There was no need to expand 3^8 first.

Example 8.17. We have $2(1) \equiv 2(4) \pmod 6$, since $2 - 8 = -6$. Cancelling the factor 2 would give $1 \equiv 4 \pmod 6$, which is false: $1 - 4 = -3$ is not divisible by 6.

The multiplication rule proves a congruence after multiplying; it does not supply the converse. Indeed, from $6 \mid 2(1-4)$ we cannot conclude $6 \mid (1-4)$. Ordinary division by a nonzero number does not automatically preserve a congruence with the same modulus.

Example 8.18. Every integer x is congruent to one of $0, 1, 2, 3$ modulo 4. Squaring its representative gives:

$x \bmod 4$	0	1	2	3
$x^2 \bmod 4$	0	1	0	1

For example, $3^2 = 9 \equiv 1 \pmod 4$. The power rule ensures that every integer with representative 3 has its square in class $[1]_4$. The other columns work the same way. Thus every integer square is congruent to 0 or 1 modulo 4.

This is a proof for all integers, rather than a test of just four integers: every possible remainder is covered, and congruence preserves squaring. In particular, an equation $x^2 = 4k + 2$ has no solution with $x, k \in \mathbb{Z}$, since its right-hand side has remainder 2.

8.6 Modular systems

So far we have used congruence to compare integers. We can also calculate entirely with the finite set of representative labels. To describe this precisely, we must state the set and the operations used on it.

❄ Definition 8.5 — Binary operation and closure

A **binary operation on a set** S assigns exactly one element of S to each ordered pair of elements of S. In function notation it is a map $S \times S \to S$. For a proposed operation, **closure** means that every permitted pair has its result in S.

The word binary refers to two inputs. A **mathematical system** here means a specified set together with specified operations on that set. The set alone does not tell us which calculations are intended.

Example 8.19. Let $S = \{0, 1, 2, 3\}$. Ordinary addition gives $2 + 3 = 5 \notin S$. It therefore does not define a binary operation on S. Ordinary multiplication also fails closure, since $2(3) = 6 \notin S$. The fact that some pairs, such as $0 + 1$, stay in S does not establish closure for all pairs.

❄ Definition 8.6 — Modular addition and multiplication

Fix $n \geq 2$ and let $S_n = \{0, 1, \ldots, n-1\}$. For $a, b \in S_n$, define

$$a \oplus_n b = (a + b) \bmod n, \qquad a \otimes_n b = (ab) \bmod n.$$

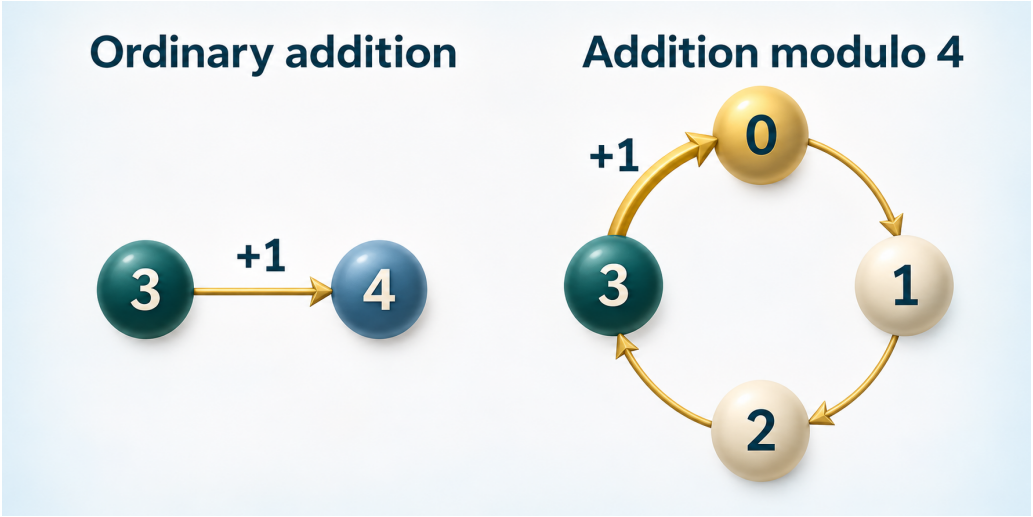
The symbols $\oplus_n$ and $\otimes_n$ denote addition and multiplication followed by reduction to the chosen remainder. Both operations have domain $S_n \times S_n$ and outputs in S_n.

Existence and uniqueness of a remainder ensure exactly one output in S_n for every input pair. Thus both are binary operations on S_n. The notation $(S_n, \oplus_n, \otimes_n)$ records the set and its two specified operations.

Example 8.20. In $S_4 = \{0, 1, 2, 3\}$,

$$2 \oplus_4 3 = 5 \bmod 4 = 1, \qquad 2 \otimes_4 3 = 6 \bmod 4 = 2.$$

The ordinary intermediate values 5 and 6 lie outside S_4; the operation outputs 1 and 2 lie inside it. Also, $3 \oplus_4 1 = 0$. This is an equality of representative labels under the defined operation; it does not assert the ordinary integer equality $3 + 1 = 0$.



Ordinary addition takes 3 to 4; adding 1 modulo 4 takes label 3 back to 0.

Operations on residue classes. The label $a \in S_n$ represents the class $[a]_n \in \mathbb{Z}_n$. On classes, adding $[a]_n$ to $[b]_n$ means forming $[a + b]_n$; multiplying means forming $[ab]_n$. These results do not depend on which integers represent the two classes.

For example, modulo 4, $[2]_4 = [6]_4$. Adding $[3]_4$ using the first representative gives $[2 + 3]_4 = [5]_4 = [1]_4$; using 6 gives $[6 + 3]_4 = [9]_4 = [1]_4$ again.

If $a \equiv a' \pmod{n}$ and $b \equiv b' \pmod{n}$, the arithmetic rules give

$$a + b \equiv a' + b' \pmod{n}, \qquad ab \equiv a'b' \pmod{n}.$$

So the resulting classes agree. Our operations on S_n record each resulting class by its unique label in $\{0, \ldots, n-1\}$. We keep brackets for classes and use unbracketed labels when calculating with $\oplus_n, \otimes_n$.

Example 8.21. Here are both operations on S_4. The row gives the first input and the column gives the second.

$\oplus_4$	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

$\otimes_4$	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

For instance, row 3, column 2 in the addition table gives $3 \oplus_4 2 = 1$. Row 2, column 2 in the multiplication table gives $2 \otimes_4 2 = 0$, because $4 \bmod 4 = 0$. Two nonzero labels can therefore have product zero.

Each table covers all $4^2 = 16$ ordered input pairs, and every output lies in S_4. This is a complete finite check of closure. The general argument using bounded remainders already establishes closure for every S_n.

Python (non-examinable)

The symbol `%` computes a remainder in Python. With the positive modulus 4, it returns one of the labels $0, 1, 2, 3$. The outer loop chooses a row label a; the inner loop visits each column label b. Each empty list `row` receives the four outputs of addition modulo 4, using `append` to add each entry. Each `range(n)` supplies $0, 1, \ldots, n-1$, so here both loops use the labels $0, 1, 2, 3$.

```
n = 4
for a in range(n):
    row = []
    for b in range(n):
        row.append((a + b) % n)
    print(row)
```

Output.

```
[0, 1, 2, 3]
[1, 2, 3, 0]
[2, 3, 0, 1]
[3, 0, 1, 2]
```

These are exactly the rows of the addition table above. Replacing `a + b` by `a * b` generates the multiplication table on the same representative labels. The loops cover every ordered input pair in $S_4 \times S_4$.

❄ **Definition 8.7 — Identity element**

For a binary operation $\star$ on S, an **identity element** is an element $e \in S$ such that

$$e \star a = a \quad \text{and} \quad a \star e = a \qquad \text{for every } a \in S.$$

It leaves every element unchanged in either input position.

An identity belongs to a specified operation. There need not be one element that acts as the identity for every operation in a system.

Example 8.22. For $a \in S_n$, the integer a already satisfies $0 \le a < n$, so $a \bmod n = a$. Consequently,

$$0 \oplus_n a = a \oplus_n 0 = a, \qquad 1 \otimes_n a = a \otimes_n 1 = a.$$

Thus 0 is the additive identity and 1 the multiplicative identity for every $n \ge 2$. In the tables of Example 8.21, row and column 0 of the addition table reproduce their input labels. Row and column 1 of the multiplication table do likewise.

Zero is not a multiplicative identity: $0 \otimes_n 1 = 0 \neq 1$. One is not an additive identity: $1 \oplus_n 0 = 1 \neq 0$. Checking just an input that happens to stay unchanged would not establish the identity condition.

Example 8.23. On S_4, define $a \ominus_4 b = (a - b) \bmod 4$. This is a binary operation because its reduced output always lies in S_4. For every $a \in S_4$, $a \ominus_4 0 = a$. But $0 \ominus_4 1 = (-1) \bmod 4 = 3$, not 1. So 0 works on the right but is not a two-sided identity.

No other label can be a two-sided identity either. If e were such an identity, $e \ominus_4 0 = 0$ would force $e = 0$, which we have just ruled out. A set with a binary operation need not have an identity.

Example 8.24. Use addition modulo 4, but restrict the allowed labels to $A = \{1, 2, 3\}$. Since $1 \oplus_4 3 = 0 \notin A$, the restricted calculation is not a binary operation on A. It remains an operation on S_4. Specifying the modulus without the permitted input and output set would miss this change.

8.7 Exercises

8.1. Decide each divisibility claim, using the nonzero-divisor convention. Justify each answer.

a) $7 \mid -35$.

b) $-4 \mid 28$.

c) $9 \mid 0$.

d) $8 \mid 30$.

8.2. a) For integers d, a, b with $d \neq 0$, prove that $d \mid a$ and $d \mid b$ imply $d \mid (2a - 3b)$.

b) For nonzero integers d, e and an integer a, prove that $d \mid e$ and $e \mid a$ imply $d \mid a$.

8.3. For each pair (a, n), find q, r with $a = nq + r$ and $0 \leq r < n$.

a) $(53, 7)$.

b) $(-53, 7)$.

c) $(-42, 7)$.

d) $(0, 6)$.

e) $(18, 1)$.

8.4. The equalities below hold, but the final summand does not satisfy the required remainder bounds. Give a valid quotient and remainder for each.

a) $17 = 5(2) + 7$.

b) $-17 = 5(-3) - 2$.

Explain why the bound is $r < n$, rather than $r \leq n$.

c) $28 = 7(3) + 7$.

8.5. There are 1028 records, packed into full groups of 32. Find the number of full groups and the number left over. State whether $32 \mid 1028$ and justify the conclusion from your quotient-and-remainder equation.

8.6. State TRUE or FALSE for each claim, justifying it by divisibility of the difference.

a) $14 \equiv -1 \pmod 5$.

b) $18 \equiv 2 \pmod 4$.

c) $-12 \equiv 3 \pmod 5$.

d) $7 \equiv 0 \pmod 3$.

e) $0 \equiv 6 \pmod 6$.

f) $11 \equiv 5 \pmod 4$.

8.7. Show that $20 \equiv -1 \pmod 7$. Find $20 \bmod 7$ and $(-1) \bmod 7$. Why is -1 a valid congruent integer but not a valid chosen remainder modulo 7?

8.8. Show that $5 \equiv 8 \pmod 3$ but $5 \not\equiv 8 \pmod 6$. Explain why these conclusions are compatible. Give another pair of distinct integers with the same two properties.

8.9. Let $n \geq 2$, and suppose $a \equiv b \pmod n$ and $b \equiv c \pmod n$. Use equations $a - b = nj$ and $b - c = nk$ to prove $a \equiv c \pmod n$. Identify which relation property this establishes and why $j + k$ is an allowed witness.

8.10. List the three distinct classes modulo 3, giving a set-builder description of each. For each class, include two negative and two nonnegative members. Explain why every integer belongs to exactly one class.

8.11. Decide whether each equality is correct and justify it. For a failed equality, give an integer that belongs to one set but not the other.

a) $[2]_5 = [12]_5$.

b) $[-1]_5 = [4]_5$.

c) $[0]_4 = [2]_4$.

d) $[1]_3 = [1]_5$.

8.12. List all integers x with $-15 \le x \le 15$ such that $x \equiv 2 \pmod 5$. Give an equation $x = 2 + 5k$ and identify all integer values of k that produce your list.

8.13. Find the chosen remainder in each case, showing congruence steps.

a) $(46 + 39) \bmod 7$.

b) $(46 \cdot 39) \bmod 7$.

c) $(39 - 46) \bmod 7$.

d) $(4 - 19) \bmod 6$.

8.14. Suppose $a \equiv 2 \pmod 5$ and $b \equiv 4 \pmod 5$. Find the chosen remainders of $a + b$, ab and $a^2 + b^2$ modulo 5. Explain why your answers do not depend on the particular integers represented by a and b.

8.15. Show that $2^3 \equiv 1 \pmod 7$. Use $20 = 3(6) + 2$ to find $2^{20} \bmod 7$, without expanding 2^{20}. State where repeated multiplication of congruences is used.

8.16. Suppose $a - b = nj$ and $c - d = nk$, with all letters integer and $n \ge 2$. Express $ac - bd$ as n times an integer to prove that $a \equiv b \pmod n$ and $c \equiv d \pmod n$ imply $ac \equiv bd \pmod n$. Explain why this also makes multiplication of residue classes independent of representatives.

8.17. Check that $4(2) \equiv 4(5) \pmod 6$. Charlie cancels the nonzero factor 4 and concludes $2 \equiv 5 \pmod 6$. Show that this conclusion is false. Explain which implication the valid multiplication rule supplies and which Charlie has assumed.

8.18. On $S_5 = \{0, 1, 2, 3, 4\}$, which of the following rules define binary operations? For a failed rule give an input pair whose result lies outside S_5; for a successful rule justify all input pairs.

a) Ordinary addition $a + b$.

b) Ordinary multiplication ab.

c) $(a + b) \bmod 5$.

d) $(ab) \bmod 5$.

8.19. Construct the addition and multiplication tables on $S_3 = \{0, 1, 2\}$ using $\oplus_3$ and $\otimes_3$. Explain how the entries show closure. Use the tables to identify the additive and multiplicative identities, checking both the relevant row and the relevant column.

8.20. In S_5, Eve notes that $2 \otimes_5 0 = 0$ and claims that 2 is a multiplicative identity. Give an input that disproves her claim. State the full identity condition and identify the actual multiplicative identity.

8.21. On S_5, define $a \ominus_5 b = (a - b) \bmod 5$.

a) Calculate $1 \ominus_5 4$ and $4 \ominus_5 1$.

b) Explain why the operation is closed.

c) Show that $a \ominus_5 0 = a$ for every $a \in S_5$, but 0 is not a two-sided identity.

d) Explain why no other element can be a two-sided identity.

8.22. Use addition and multiplication modulo 4.

a) On $A = \{1, 2, 3\}$, give a pair showing that addition is not closed.

b) On $B = \{0, 2\}$, check every ordered pair to decide whether addition is closed.

c) On the same B, check every ordered pair to decide whether multiplication is closed.

8.23. On $S = \{u, v\}$, define $\star$ by the following table, with the first input selecting the row.

$\star$	u	v
u	u	v
v	v	u

Explain why $\star$ is a binary operation on S. Identify its identity and show why the other element is not an identity. The labels u, v are elements of this set; no ordinary numerical addition is being assumed.

8.24. The printer-checking order is Alice, Bob, Charlie, David, repeating, with labels $0, 1, 2, 3$ respectively. At turn 0, Charlie checks.

a) Give a formula for the checking label at turn $t \geq 0$ using a remainder.

b) Who checks at turn 14?

c) If the same repeating allocation is extended backwards to integer turns, who checks at turn -3?

Explain why including the initial label in the formula is necessary.

8.25. For any integer k, its remainder modulo 2 is 0 or 1. Use these two cases and congruence arithmetic to prove $2 \mid k(k+1)$ for every $k \in \mathbb{Z}$. Explain why two remainder cases cover infinitely many integers.

8.26. Use Example 8.18 to find all possible remainders of $x^2 + y^2$ modulo 4, where $x, y \in \mathbb{Z}$. Prove that there are no integers x, y, m satisfying

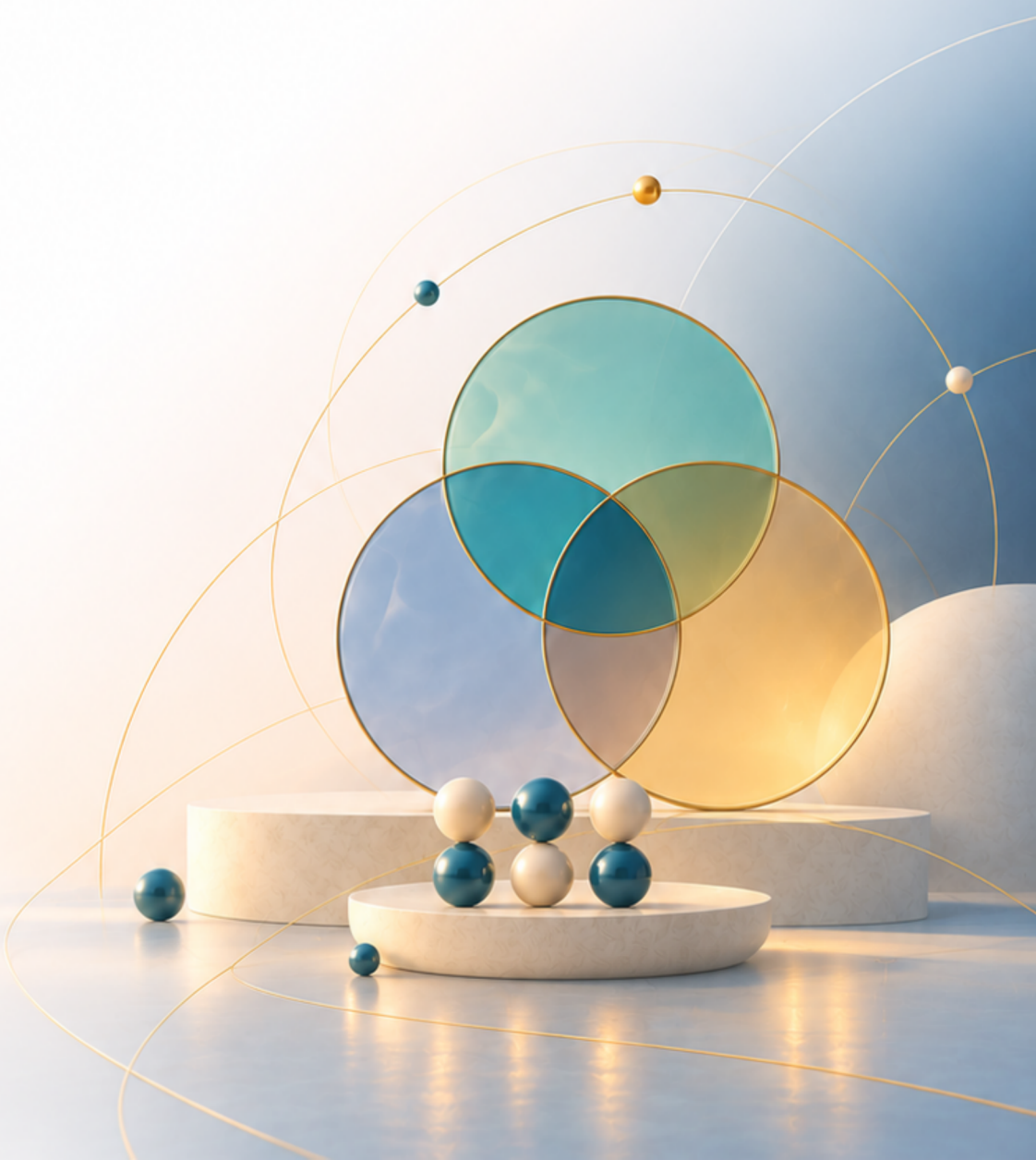
$$x^2 + y^2 = 4m + 3.$$

Explain why this is a general impossibility proof rather than a check of a few values of x and y.

LECTURE NOTES
DISCRETE MATHEMATICS

CHAPTER 9

Counting principles



Updated: 19 November 2026

9 Counting principles

A workshop organiser has two lists: 18 students know some Python, and 14 know some HTML. Does that give 32 different students? Some names may occur on both lists. Adding their lengths would count those students twice.

Counting becomes more demanding when we must specify what counts as one outcome. A two-person team and an assignment of two people to different roles are different objects. A code may start with 0, while a four-digit number cannot. These choices change the count before any calculation begins.

Chapter 4 gave us unions, intersections and set difference. Chapter 7 added Cartesian products and partitions. We will use those ideas to count finite sets without always listing every element. The aim is to explain why the chosen method counts each allowed outcome exactly once.

What you should be able to do

After working through this chapter, you should be able to:

- specify the outcomes and check small counts by listing;
- add counts from disjoint, exhaustive cases;
- multiply counts of successive choices when each stage has a fixed number of extensions;
- handle restrictions on repetition, leading zeros and available choices;
- count a complement when the unwanted outcomes are easier to describe;
- use inclusion–exclusion for two or three overlapping sets;
- interpret “at least”, “exactly”, “both” and “neither” in a counting problem.

9.1 Counting outcomes

❄ Definition 9.1 — Outcome set and cardinality

An **outcome** is one complete object allowed by the task. The **outcome set** consists of all such objects. For a finite set S, its **cardinality** $|S|$ is the number of its distinct elements. In particular, $|\varnothing| = 0$.

We are counting objects, not probabilities. Writing an outcome explicitly helps us see which choices are part of it, which restrictions apply, and when two descriptions refer to the same object.

Example 9.1. A code consists of one letter from $\{A, B\}$ followed by one digit from $\{0, 1, 2\}$. Its outcome set is

$$S = \{\mathrm{A0}, \mathrm{A1}, \mathrm{A2}, \mathrm{B0}, \mathrm{B1}, \mathrm{B2}\}.$$

Thus $|S| = 6$. A table organises the same six codes.

First character	0	1	2
A	A0	A1	A2
B	B0	B1	B2

Choosing one character from $\{A, B, 0, 1, 2\}$ would instead give five outcomes. That is a different task: each outcome would have one character, rather than a letter and a digit in specified positions.

Python (non-examinable)

We can construct the six codes by trying each digit after each letter. Square brackets create a *list*, and `append` adds an item to its end. The indented inner loop runs once for every digit before the outer loop moves to the next letter. Quotation marks make the digits text, so + joins the two characters.

```
codes = []
for letter in ["A", "B"]:
    for digit in ["0", "1", "2"]:
        codes.append(letter + digit)
print(codes)
print(len(codes))
```

Output.

```
['A0', 'A1', 'A2', 'B0', 'B1', 'B2']
6
```

Here `len` counts the items in the list. Each permitted letter–digit pair is constructed exactly once.

Example 9.2. Choose an editor and a reviewer from Alice, Bob and Charlie, with different people in the two roles. An outcome is an ordered pair (editor, reviewer). The six outcomes are

$$\begin{aligned} &(\text{Alice}, \text{Bob}), \quad (\text{Alice}, \text{Charlie}),\\ &(\text{Bob}, \text{Alice}), \quad (\text{Bob}, \text{Charlie}),\\ &(\text{Charlie}, \text{Alice}), \quad (\text{Charlie}, \text{Bob}). \end{aligned}$$

For example, swapping Alice and Bob changes the assignment. If we only wanted a two-person team with no roles, the outcomes would instead be the three sets

$$\{\text{Alice}, \text{Bob}\}, \quad \{\text{Alice}, \text{Charlie}\}, \quad \{\text{Bob}, \text{Charlie}\}.$$

The same people are available, but the two tasks have different outcome sets. Chapter 10 will develop systematic methods for these selections and arrangements.

9.2 Sum rule

Recall that sets are **disjoint** when their intersection is empty. To add the sizes of cases, those cases must not count the same outcome, and together they must cover every outcome we want.

❄ Definition 9.2 — Sum rule

For finite disjoint sets A and B,

$$|A \cup B| = |A| + |B|.$$

More generally, add the sizes of any finite collection of pairwise disjoint cases to count their union.

To justify the formula, list the elements of A, then the elements of B. No element is repeated because $A \cap B = \emptyset$, and every element of $A \cup B$ occurs on the combined list.

Example 9.3. A folder contains five PDF files, four PNG files and three TXT files. Each file has exactly one of these formats. Selecting one PDF or PNG file gives

$$5 + 4 = 9$$

outcomes: the two format categories are disjoint. If TXT files are also allowed, there are $5 + 4 + 3 = 12$ outcomes. An outcome here is one file, not a package containing one file of each format.

Example 9.4. A system has seven queued records, three processing records and twelve completed records. Each record has exactly one of these statuses, and every record has a status. There are

$$7 + 3 + 12 = 22$$

records. The three status sets form a partition: they cover the records and are pairwise disjoint. Chapter 7's partition conditions justify adding the three counts without repeating a record.

9.3 Product rule

A complete outcome may require several choices. We must count all the possible continuations after each choice.

❄ Definition 9.3 — Product rule

Suppose two successive choices produce each allowed outcome exactly once. There are m possibilities for the first choice, and *each* first choice allows exactly n possibilities for the second. Then there are mn outcomes.

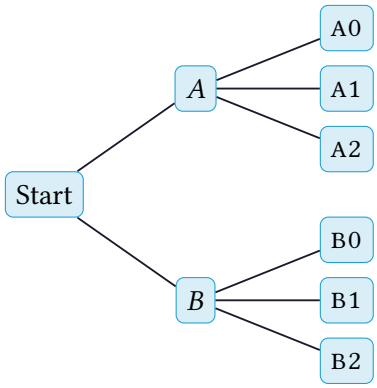
In particular, for finite sets A, B,

$$|A \times B| = |A|\,|B|.$$

For each first choice, there is a group of n complete outcomes. The groups are disjoint because their first choices differ. Adding their sizes gives n repeated m times, namely mn. The requirement of unique specification ensures that we count outcomes rather than several descriptions of the same outcome.

For three stages with m, n, p choices at their respective stages, the count is mnp, provided every partial outcome at a stage has the stated number of extensions. The same reasoning works for further stages.

Example 9.5. A choice tree. Return to Example 9.1. Choose the letter first: there are two choices. For either letter, there are three choices of digit. Thus the product rule gives $2(3) = 6$. The following rooted tree records these choices.



The root records no choice yet. Its children record the first character; each leaf records one complete code. Every code reaches one leaf, and different leaves give different codes. Reading away from the root gives the order of the choices.

Example 9.6. A profile uses one of three icons, one of two colour themes and one of two font sizes. Every combination is allowed. An outcome is the triple (icon, theme, size), so there are

$$3(2)(2) = 12$$

configurations. After any icon is selected, both themes remain available; after either theme, both sizes remain available.

Example 9.7. A four-character digit code uses digits 0 to 9; repetition and a leading zero are allowed. Each position has ten choices, giving

$$10^4 = 10(10)(10)(10) = 10{,}000$$

codes. For example, 0042 is a valid code. The exponent 4 records the four factors of 10.

For a four-digit positive integer, the first digit must be one of $1, \dots, 9$. Repetition is still allowed, and each later digit has ten choices. The count becomes

$$9(10)(10)(10) = 9{,}000.$$

The integer 42 has two digits, so writing it with two leading zeros does not create a new four-digit integer.

Example 9.8. A three-letter code uses $\{A, B, C, D\}$ without repetition. The first position has four choices. Whichever letter is chosen, the second position has three unused letters available. After any pair of distinct letters, the third position has two unused letters available. There are

$$4(3)(2) = 24$$

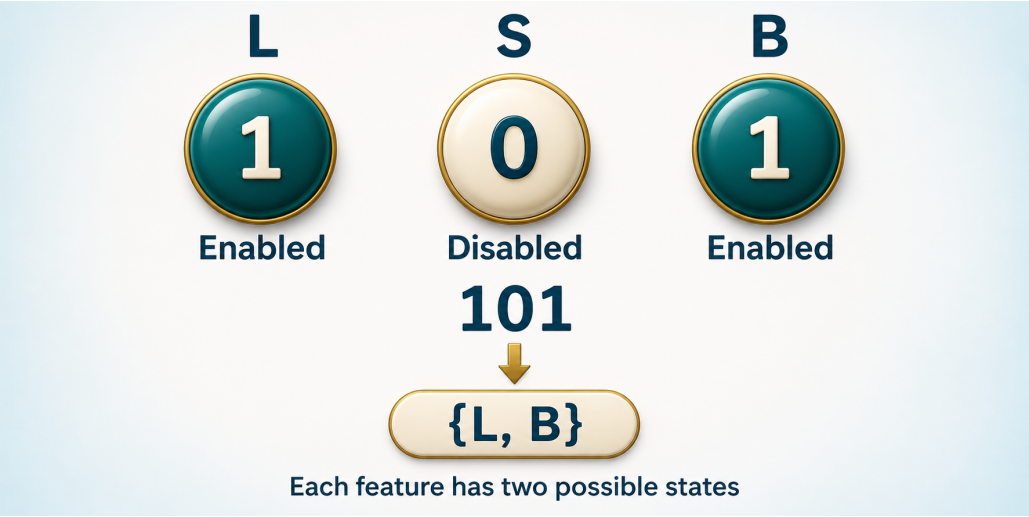
codes. The available *letters* change between branches, but the number available at each stage is fixed. The product rule does not require the same letter set to remain available.

Example 9.9. Let $F = \{p, q, r\}$ be three files and let $S = \{x, y\}$ be two servers. A function $f : F \to S$ assigns each file exactly one server. Different files may use the same server.

Choose $f(p)$, then $f(q)$, then $f(r)$. There are two choices each time, so there are $2^3 = 8$ functions. Each triple of values defines exactly one function, and two different triples define different functions. Nothing requires both servers to be used.

Example 9.10. Three optional features are labelled L, S, B. Any subset of these features may be enabled, including none. Record three bits in that fixed order: 1 means enabled and 0 means disabled. For example, 101 represents $\{L, B\}$, and 000 represents $\emptyset$.

Each of the three positions has two choices, giving $2^3 = 8$ selections. Writing $\{B, L\}$ instead of $\{L, B\}$ does not give another selection: both have the same bit record. The encoding includes every subset exactly once.



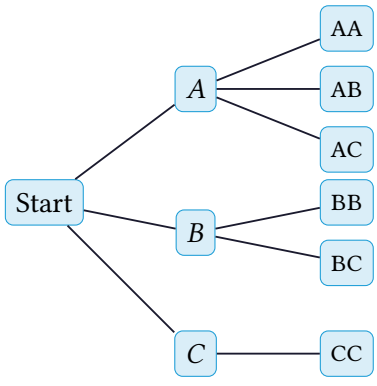
In the fixed order L, S, B, the bit string 101 encodes the subset $\{L, B\}$. Three independent binary choices give eight subsets.

For a set of n labelled features, the same argument gives 2^n subsets. When $n = 0$, the empty feature set has one subset, itself, agreeing with $2^0 = 1$.

9.4 Counting by cases

If the number of extensions changes with an earlier choice, a single product of fixed stage counts may not apply. We can often separate the possibilities into disjoint cases and count each case.

Example 9.11. Unequal branches. A two-letter code uses $\{A, B, C\}$, with the second letter not earlier than the first in alphabetical order. Repetition is allowed.



After A, the second letter has three choices; after B, two; after C, one. These cases are disjoint because they have different first letters. The count is therefore

$$3 + 2 + 1 = 6.$$

Multiplying 3(3) would include the forbidden codes BA, CA and CB.

Example 9.12. An identifier has either two letters or one letter followed by one digit. Letters come from $\{A, B, C, D\}$, digits from $\{0, 1, 2\}$, and repetition is allowed.

The two-letter format has $4(4) = 16$ possibilities. The letter–digit format has $4(3) = 12$. A code cannot have both formats, because its second character cannot be both a letter and a digit. Thus there are

$$16 + 12 = 28$$

identifiers. Each product counts one case; the sum combines the disjoint cases.

Example 9.13. How many length-3 strings over $\{A, B, C\}$ contain exactly one A? A **string** is an ordered list of characters; it need not spell a word.

The unique A can be first, second or third. For each of these three cases, each other position has two choices, B or C, giving $2(2) = 4$ strings. The cases do not overlap because there is only one A. The total is

$$4 + 4 + 4 = 3(2^2) = 12.$$

Equivalently, choosing the position of the unique A and the remaining two characters specifies each string once.

9.5 Counting complements

Sometimes the allowed outcomes have several forms, while their complement has one simple description.

❄ Definition 9.4 — Counting a complement

Let U be a finite set of all outcomes under consideration, and let $A \subseteq U$ be the wanted outcomes. The **complement of A in U** is $U \setminus A$. Then

$$|A| = |U| - |U \setminus A|.$$

The sets A and $U \setminus A$ are disjoint and together make up U. The formula follows by subtracting one term from their sum. Both counts must use the same underlying set U.

Example 9.14. For length-3 strings over $\{A, B, C\}$, repetition allowed, count those containing at least one A. There are $3^3 = 27$ strings in total. Those with no A use only B, C, so there are $2^3 = 8$. Hence the required count is

$$27 - 8 = 19.$$

This includes strings with one, two or three occurrences of A. It is larger than the count 12 for *exactly* one A in Example 9.13. For instance, AAB belongs to the present outcome set but not the earlier one.

Example 9.15. A length-4 bit string uses 0 or 1 at each position. Count those with at least two 1s. There are $2^4 = 16$ strings altogether.

The unwanted strings have either no 1s or exactly one 1. There is one all-zero string. There are four strings with exactly one 1, one for each possible position:

$$1000, \quad 0100, \quad 0010, \quad 0001.$$

These two unwanted cases are disjoint, so their total is $1 + 4 = 5$. The wanted count is $16 - 5 = 11$.

9.6 Two overlapping sets

When an outcome belongs to two cases, adding their counts includes it twice. We must remove its extra copy.

❄ Definition 9.5 — Inclusion–exclusion for two sets

For finite sets A, B,

$$|A \cup B| = |A| + |B| - |A \cap B|.$$

The union includes elements in A, in B, or in both.

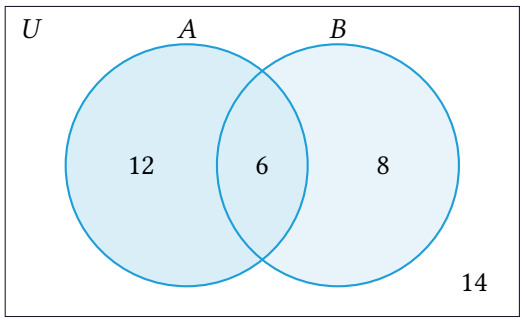
An element in exactly one set contributes 1 to the right-hand side. An element in both contributes $1 + 1 - 1 = 1$. An element in neither contributes 0. Thus each element of the union is counted once. For disjoint sets, $|A \cap B| = 0$, giving the sum rule.

Example 9.16. In a group U of 40 students, let A contain the 18 who know some Python and B the 14 who know some HTML. Suppose 6 students know both. Then

$$|A \cup B| = 18 + 14 - 6 = 26.$$

We subtract six extra counts, retaining one count for each student who appears on both lists.

The three regions inside the circles have sizes $18 - 6 = 12$ for A only, 6 for both, and $14 - 6 = 8$ for B only. Outside both are $40 - 26 = 14$.



Knowing *at least one* of the two topics gives 26 students. Knowing *exactly one* gives $12 + 8 = 20$, excluding the six who know both. Knowing *neither* gives 14. In mathematical "A or B", the word "or" includes the intersection unless the task explicitly excludes it.

Example 9.17. Count length-4 strings over $\{A, B, C\}$ that start with A or end with A, with repetition allowed. Let P be the strings starting with A, and Q the strings ending with A.

Fixing one endpoint leaves three free positions, so $|P| = |Q| = 3^3 = 27$. Fixing both endpoints leaves two free positions, so $|P \cap Q| = 3^2 = 9$. Therefore

$$|P \cup Q| = 27 + 27 - 9 = 45.$$

For example, ABBA appears in both initial counts, but is wanted only once. The middle two characters remain unrestricted in the intersection.

Example 9.18. Using the groups in Example 9.16, assign a Python helper from A and an HTML helper from B. An assignment is an ordered pair (u, v), where $u \in A$ is the Python helper and $v \in B$ the HTML helper. Initially allow the same student to fill both roles. Every one of the 18 first choices allows all 14 second choices, so there are $18(14) = 252$ assignments.

If the helpers must be different people, remove the six assignments (u, u) with $u \in A \cap B$. The count is $252 - 6 = 246$. The overlap of A and B does not prevent counting $A \times B$ by multiplication. By contrast, the six students in $A \cap B$ are single students, not 252 ordered pairs.

9.7 Three overlapping sets

With three sets, subtracting the pairwise overlaps removes too much from the common centre.

❄ Definition 9.6 — Inclusion–exclusion for three sets

For finite sets A, B, C,

$$\begin{aligned} |A \cup B \cup C| = {} & |A| + |B| + |C| \\ & - |A \cap B| - |A \cap C| - |B \cap C| \\ & + |A \cap B \cap C|. \end{aligned}$$

Each pairwise intersection includes the elements that also belong to the third set.

To verify the formula, follow one element. If it lies in exactly one set, it contributes 1. If it lies in exactly two, it contributes $2 - 1 = 1$. If it lies in all three, it contributes $3 - 3 + 1 = 1$: it belongs to three individual sets, three pairwise intersections and the triple intersection. An element outside all three contributes 0.

Example 9.19. In a set U of 50 computers, let A contain those with backup enabled, B those with file synchronisation enabled, and C those with event logging enabled. The counts are

$$\begin{aligned} |A| = 26, \quad |B| = 24, \quad |C| = 20, \\ |A \cap B| = 12, \quad |A \cap C| = 10, \quad |B \cap C| = 9, \\ |A \cap B \cap C| = 5. \end{aligned}$$

The pairwise counts include the five computers with all three features. Inclusion–exclusion gives

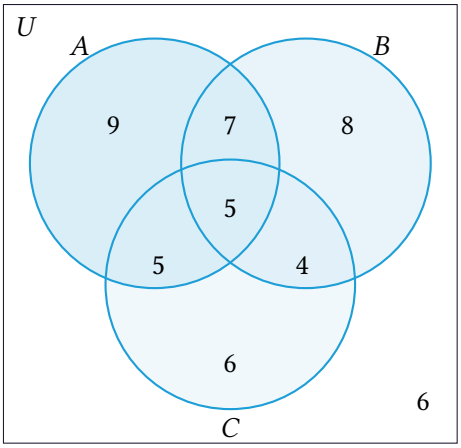
$$|A \cup B \cup C| = 26 + 24 + 20 - 12 - 10 - 9 + 5 = 44.$$

Thus $50 - 44 = 6$ computers have none of the three enabled.

To see all the regions, put 5 in the common centre. The regions belonging to exactly two sets have sizes

$$12 - 5 = 7, \qquad 10 - 5 = 5, \qquad 9 - 5 = 4$$

for A, B only; A, C only; and B, C only, respectively. For A only, remove the 7, 5 and 5 already placed inside A, giving $26 - 7 - 5 - 5 = 9$. Similarly, B only contains $24 - 7 - 4 - 5 = 8$, and C only contains $20 - 5 - 4 - 5 = 6$.



The seven disjoint inner regions sum to $9+8+6+7+5+4+5 = 44$, agreeing with the formula. Including the outside region gives all 50 computers.

Example 9.20. For Example 9.19, exactly two features are enabled on $7 + 5 + 4 = 16$ computers. At least two are enabled on $16 + 5 = 21$. Exactly one is enabled on $9 + 8 + 6 = 23$.

Adding the pairwise intersection counts gives $12 + 10 + 9 = 31$. Each computer in exactly two sets occurs once in that sum, while each computer in all three occurs three times. To count exactly two, remove all three copies of each central computer: $31 - 3(5) = 16$. To count at least two, retain one central copy: $31 - 2(5) = 21$.

Example 9.21. Suppose $|A| = 15$, $|B| = 12$ and $|A \cap B| = 18$. These data are impossible: an intersection is a subset of each set, so it cannot have more than 12 elements.

Now suppose a universe has 30 elements, $|A| = 20$, $|B| = 18$ and $|A \cap B| = 5$. The intersection size is no larger than either set, but inclusion–exclusion would give $|A \cup B| = 20 + 18 - 5 = 33$. A union contained in this universe cannot have 33 elements. The data are again impossible. A numerical result from a formula must still describe subsets of the stated universe.

9.8 Exercises

9.1. Let $L = \{A, B, C\}$ and $D = \{0, 1\}$.

a) List all codes consisting of a letter from L followed by a digit from D.

b) How many outcomes are there if the task is instead to choose one character from $L \cup D$?

c) Explain the two different counts using the objects that constitute an outcome.

9.2. The available people are Alice, Bob, Charlie and David. List all two-person teams with no roles. Then list all assignments of different people to the roles of coder and tester. Explain why exchanging two people changes an assignment but does not change a team.

9.3. A folder contains eight PDF files, five SVG files and four TXT files, with each file having exactly one format. How many ways can you select one SVG or TXT file? How many ways can you select one file of any of the three formats? State why addition is valid.

9.4. An equivalence relation partitions a set into three classes of sizes 4, 3, 2. How many elements are in the set, and how many distinct equivalence classes are there? Explain why the class sizes can be added.

9.5. A profile uses one of four icons, one of three colours and one of two sizes. Every combination is allowed. Describe an outcome as a triple and count the configurations. Explain what each factor counts.

9.6. Use the digits 0 to 9, with repetition allowed. Count:

a) three-character digit codes;

b) three-digit positive integers;

c) three-digit positive integers whose final digit is even.

Include 0 among the allowed even final digits.

9.7. How many length-3 strings over $\{A, B, C, D, E\}$ are possible in each case?

a) Repetition is allowed.

b) No character may repeat.

In part (b), explain why changing the available letters does not invalidate the product rule.

9.8. Let $F = \{p, q, r, s\}$ and $S = \{x, y, z\}$. Different elements of F may have the same image. Count the functions $f : F \to S$ satisfying:

a) no additional restriction;

b) $f(p) = x$;

c) $f(p) \neq f(q)$, with no other restriction.

9.9. Any subset of four labelled features may be enabled. Count selections with:

a) no restriction;

b) no feature enabled;

c) exactly one feature enabled;

d) at least one feature enabled.

Explain how a four-bit record represents each selection exactly once.

9.10. A two-character code has first character A, B or C. Its allowed second characters are given below.

First character	Allowed second characters
A	0, 1
B	0, 1, 2
C	2

Draw a choice tree, list all codes and give their count. Why would 3(3) be an incorrect count?

9.11. An identifier has either two letters or one letter followed by a digit. Letters come from $\{A, B, C, D, E\}$, digits from $\{0, 1, 2, 3\}$.

a) Count identifiers when repetition is allowed.

b) Count identifiers when the two-letter format must use different letters.

Explain why the two format counts can be added.

9.12. For length-4 strings over $\{A, B, C\}$, with repetition allowed, count strings containing:

a) no A;

b) exactly one A;

c) at least one A;

d) at least two As.

Identify the disjoint cases or complement used for each calculation.

9.13. How many length-5 bit strings contain at least two 1s? Count the unwanted cases separately and explain why their counts can be added.

9.14. In a universe U of 45 records, $|A| = 24$, $|B| = 20$ and $|A \cap B| = 9$. Find the number of records in:

a) A only;

b) B only;

c) both sets;

d) at least one set;

e) exactly one set;

f) neither set.

Draw and label the corresponding Venn diagram.

9.15. In a universe of 60 elements, $|A| = 35$, $|B| = 28$ and 10 elements belong to neither set. Find $|A \cup B|$, $|A \cap B|$ and the number belonging to exactly one set. Check that the four disjoint region counts sum to 60.

9.16. Consider all length-5 bit strings. Let A contain those starting with 1, and B those ending with 0. Find:

a) $|A|$ and $|B|$;

b) $|A \cap B|$;

c) $|A \cup B|$;

d) the number in exactly one of A, B.

Explain why 10000 is counted twice by $|A| + |B|$.

9.17. In a group of 30 students, 12 are in A, 15 are in B, and 4 are in both. Bob says that 12(15) counts the students in both sets.

a) Explain the error and give the correct count.

b) What does 12(15) count if an outcome is an ordered pair with first entry in A and second in B?

c) How many such ordered pairs have different entries?

9.18. A universe of 60 elements has subsets with counts

$$|A| = 30, \quad |B| = 28, \quad |C| = 25,$$
$$|A \cap B| = 14, \quad |A \cap C| = 12, \quad |B \cap C| = 11,$$
$$|A \cap B \cap C| = 5.$$

The pairwise counts include the triple intersection. Draw a Venn diagram and label all eight disjoint regions, including the outside region. Find the numbers belonging to:

a) at least one set;

b) no set;

c) exactly one set;

d) exactly two sets;

e) at least two sets.

9.19. In a universe of 50 elements, exactly 5 belong to none of A, B, C. The other data are

$$|A| = 25, \quad |B| = 22, \quad |C| = 20,$$
$$|A \cap B| = 10, \quad |A \cap C| = 9, \quad |B \cap C| = 8.$$

Find $|A \cap B \cap C|$, the number in A only, and the number in exactly two sets.

9.20. Explain why each report is impossible.

a) $|A| = 12$, $|B| = 9$, $|A \cap B| = 10$.

b) A universe has 25 elements, $|A| = 18$, $|B| = 15$ and $|A \cap B| = 5$.

c) A universe has 30 elements, $|A| = 10$, $|B| = 12$, $|C| = 11$, $|A \cap B| = 8$, $|A \cap C| = 7$, $|B \cap C| = 5$ and $|A \cap B \cap C| = 2$. Compute the proposed A-only region.

9.21. To count length-3 strings over $\{A, B, C\}$ containing at least one A, Charlie chooses a position for A and then chooses the other two characters freely. He obtains $3(3^2) = 27$.

a) How many times does this method count ABA? How many times does it count AAA?

b) Explain which requirement of the product rule has failed.

c) Give a correct count using a complement.

9.22. For finite A, B, use the disjoint sets $A \setminus B$, $A \cap B$ and $B \setminus A$ to prove

$$|A \cup B| = |A| + |B| - |A \cap B|.$$

Express $|A|$, $|B|$ and $|A \cup B|$ in terms of those three region sizes.

9.23. A proposed three-set formula adds the three set sizes, subtracts the three pairwise intersection sizes, and stops there. Track how many times it counts an element belonging to exactly two sets and an element belonging to all three. Give the missing term and explain why the corrected formula counts every element of the union once.

9.24. An equivalence relation R partitions a set of 12 records into three classes of sizes 3, 4, 5.

a) Count the ordered pairs (x, y) satisfying xRy, including self-pairs.

b) How many of these pairs have $x = y$?

c) How many have $x \neq y$?

Explain why products within the classes can be added without repeating a pair.

9.25. Fix the labelled vertex set $V = \{a, b, c, d\}$. Graphs here are simple and undirected, as in Chapter 6.

a) List the six possible edges as two-element subsets.

b) Choose independently whether to include each possible edge. How many graphs on this fixed V result?

c) How many contain $\{a, b\}$?

d) How many have no edges?

Retain all four labelled vertices in every graph, including any isolated ones.

LECTURE NOTES
DISCRETE MATHEMATICS

CHAPTER 10

Permutations and combinations



Updated: 19 November 2026

10 Permutations and combinations

A system selects three of eight distinct records for review. Choosing a first record, a second and a third gives $8(7)(6) = 336$ ordered lists. But if the system stores only which three records were selected, different orders name the same selection. How do we remove those repeated descriptions?

If the system instead assigns first, second and third review priorities, changing the order changes the outcome. Chapter 9's instruction to specify one complete outcome therefore remains our starting point.

This chapter develops formulas for arrangements and selections from the sum and product rules. Chapter 7's equivalence classes will explain why a division can remove repeated orders. We will then use counting to prove identities, revisit Chapter 5's induction, and explain the coefficients in binomial expansions.

What you should be able to do

After working through this chapter, you should be able to:

- calculate and simplify factorial expressions;
- count ordered arrangements of distinct objects;
- count subsets and justify division by the number of orders;
- handle restrictions using the earlier counting rules;
- recognise when a string is determined by a set of positions;
- prove selected identities by counting and by induction;
- explain and use binomial coefficients in an expansion.

10.1 Factorials and arrangements

When all objects must be used once, each position leaves one fewer object available for the next. We begin with notation for the resulting product.

❄ Definition 10.1 — Factorial

For a positive integer n, its **factorial** is the product of the positive integers from 1 through n:

$$n! = n(n-1)(n-2)\cdots 2(1).$$

We also define $0! = 1$. The notation is read "n factorial".

Example 10.1. Alice, Bob and Charlie each give one presentation. Write A, B, C for their names, and record the speaking order as a string using each letter once. There are three choices for the first speaker, two for the second and one for the last, giving $3(2)(1) = 3! = 6$.

First speaker	Complete orders	
Alice	ABC	ACB
Bob	BAC	BCA
Charlie	CAB	CBA

The table lists every outcome once. All three people participate; only their order changes.

Example 10.2. Expanding the relevant factors gives

$$\frac{8!}{5!} = \frac{8(7)(6)\,5!}{5!} = 8(7)(6) = 336.$$

We cancel the entire common factor $5!$. The quotient is not $8/5$. Similarly, $5! = 5(4)(3)(2)(1) = 120$ and $1! = 1$.

❄ Definition 10.2 — Permutation of all the objects

A **permutation** of n distinct objects is an ordered arrangement using every object exactly once. The number of such arrangements is $n!$.

For $n \geq 1$, the product rule gives n choices, then $n-1$, continuing down to one. For $n = 0$, there is one arrangement: the empty list. This interpretation agrees with $0! = 1$. Having no objects to arrange does not give zero possible empty arrangements.

Example 10.3. Five distinct test cases must each be run once in a schedule. If any order is allowed, there are $5! = 120$ schedules. If one particular case must run first, the remaining four cases can be arranged in $4! = 24$ ways. If two particular, different cases must run first and last respectively, the three middle cases have $3! = 6$ possible orders.

The fixed positions supply no additional choice. We arrange only the cases whose positions remain free.

Example 10.4. The printed strings using two As and one B are

$$\text{AAB}, \qquad \text{ABA}, \qquad \text{BAA}.$$

There are three, rather than $3! = 6$. If the As were labelled A_1, A_2, there would be six orders of three distinct objects. Removing those labels identifies two orders for each printed string. Thus the factorial count for distinct objects cannot be transferred unchanged to indistinguishable symbols.

10.2 Permutations

We may need only k positions from a pool of n distinct objects. In this section an object cannot be reused within an arrangement.

❄ Definition 10.3 — Ordered arrangements without repetition

For integers $0 \le k \le n$, let $P(n, k)$ count the ordered lists of k distinct objects chosen from n available objects. Then

$$P(n, k) = \frac{n!}{(n-k)!}.$$

Such an arrangement is also called a k-**permutation**.

For $k \ge 1$, choose the entries successively:

$$P(n, k) = n(n-1)\cdots(n-k+1).$$

There are k factors, ending at $n - k + 1$. The factorial quotient cancels the remaining factor $(n-k)!$ in $n!$. When $k = 0$, there is one empty list, and the quotient is $n!/n! = 1$. When $k = n$, it becomes $n!/0! = n!$.

If $k > n$, no list of k distinct objects can be formed from that pool. The count is zero, but the displayed factorial quotient is not applicable: $(n-k)!$ would have a negative argument.

Example 10.5. Assign three different records to first, second and third review priority, from eight distinct records. The positions are labelled, so an outcome is an ordered triple. There are

$$P(8, 3) = 8(7)(6) = \frac{8!}{5!} = 336$$

assignments. We stop after three factors. Using 8! would also arrange the five records that were not assigned a priority.

Example 10.6. A three-character digit code uses 0 to 9 without repetition. Its positions give $P(10, 3) = 10(9)(8) = 720$ codes. For instance, 017 is a valid code.

A three-digit positive integer cannot start with 0. There are nine choices for its first digit, nine remaining choices for its second, and eight for its third. Thus there are $9(9)(8) = 648$ such integers. The second position may use 0; it must only avoid the digit already used. The difference $720 - 648 = 72$ counts the codes beginning with 0.

Example 10.7. Let $F = \{p, q, r\}$ be three tasks and let S be five available servers. An injective function $f : F \to S$ sends different tasks to different servers. Choose $f(p)$ in five ways, then $f(q)$ in four, then $f(r)$ in three. There are $P(5, 3) = 5(4)(3) = 60$ injective functions.

If different tasks may share a server, the count is instead $5^3 = 125$, as in Chapter 9. An injective assignment here uses only three of the five servers; it need not use every available server.

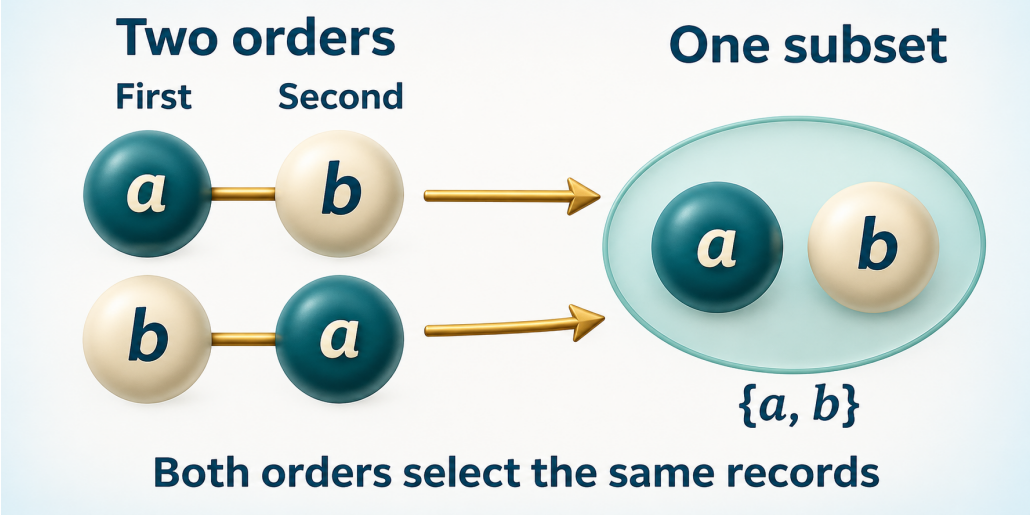
10.3 Combinations

An unordered selection records which objects are included, rather than assigning them to positions. We can start by counting ordered lists and determine how often each selection is represented.

Example 10.8. Choose two distinct records from $\{a, b, c, d\}$. The ordered pairs number $P(4, 2) = 4(3) = 12$. Grouping pairs with the same underlying subset gives:

Selected subset	Its two orders
$\{a, b\}$	$(a, b),\ (b, a)$
$\{a, c\}$	$(a, c),\ (c, a)$
$\{a, d\}$	$(a, d),\ (d, a)$
$\{b, c\}$	$(b, c),\ (c, b)$
$\{b, d\}$	$(b, d),\ (d, b)$
$\{c, d\}$	$(c, d),\ (d, c)$

Every subset has exactly two orders, so there are $12/2 = 6$ subsets. In Chapter 7's language, two ordered pairs are equivalent when they contain the same records. The six equivalence classes all have size $2! = 2$. Their equal size is what justifies this division.



The orders (a, b) and (b, a) belong to one group because both select the subset $\{a, b\}$. Each selected pair has $2!$ orders.

Python (non-examinable)

Python's standard `itertools` module supplies `permutations` for ordered arrangements and `combinations` for unordered selections. The `from ... import ...` line makes those two tools available; `list` collects their results so that `len` can count them. The square-bracketed list `records` holds our four distinct inputs. The argument `2` specifies that we take two records at a time.

```
from itertools import permutations, combinations
records = ["a", "b", "c", "d"]
orders = list(permutations(records, 2))
selections = list(combinations(records, 2))
print(len(orders), len(selections))
print(selections)
```

Output.

```
12 6
[('a', 'b'), ('a', 'c'), ('a', 'd'), ('b', 'c'), ('b', 'd'), ('c', 'd')]
```

The parenthesised pairs are Python's representations of the selections: the tool lists each selected pair once, using the input order. That display order does not give the mathematical subset an order. In contrast, `orders` contains both `('a', 'b')` and `('b', 'a')`. The four input records are distinct, as required by our counting argument.

❄ Definition 10.4 — Combination and binomial coefficient

A k-**combination** of n distinct objects is a k-element subset of those objects. For integers $0 \le k \le n$, its count is

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}.$$

The notation is read "n choose k". It is called a **binomial coefficient**; the two stacked numbers do not denote a fraction.

To justify the formula, every k-element subset has exactly $k!$ ordered arrangements. Conversely, every ordered list of k distinct objects belongs to exactly one underlying subset. The lists therefore form equal groups of size $k!$:

$$P(n,k) = \binom{n}{k} k!, \qquad \binom{n}{k} = \frac{P(n,k)}{k!}.$$

Substituting the formula for $P(n,k)$ gives the stated factorial expression. The argument also works for $k = 0$, with one empty subset and one empty ordering.

Example 10.9. The system at the start of the chapter selects three of eight distinct records, with no priorities. Each selected set has $3! = 6$ possible orders. Thus

$$\binom{8}{3} = \frac{8(7)(6)}{3(2)(1)} = \frac{336}{6} = 56.$$

For the boundary cases, $\binom{8}{0} = 1$ counts the empty subset and $\binom{8}{8} = 1$ counts the full set. Also $\binom{0}{0} = 1$: the empty set has exactly one subset, itself.

We will use a counting convention in identities: for $n \geq 0$ and an integer k outside $0 \leq k \leq n$, set $\binom{n}{k} = 0$. There cannot be a subset of negative size, or one larger than the whole set. This convention does not extend factorials to negative integers. Use the factorial formula only in its stated range.

Example 10.10. Six students are available. Selecting a three-person team with no roles gives $\binom{6}{3} = 20$ teams. Assigning different students to three named roles gives $P(6,3) = 6(5)(4) = 120$ assignments.

Selecting a three-person team and designating one member as leader gives $20(3) = 60$ outcomes. Only the leader is distinguished: the other two members are not ordered. Equivalently, choose the leader in six ways and the other two members from the remaining five: $6\binom{5}{2} = 6(10) = 60$.

Example 10.11. A pool contains four members of Group H and three of Group S, with no person in both groups. A three-person team with exactly two from H and one from S can be chosen in

$$\binom{4}{2}\binom{3}{1} = 6(3) = 18$$

ways. Each team determines its H-subset and its S-subset uniquely, so the product counts teams once.

For teams with at least one member from S, count all $\binom{7}{3} = 35$ teams and exclude the $\binom{4}{3} = 4$ teams drawn entirely from H. The result is $35 - 4 = 31$.

Example 10.12. Alice and Bob are among eight available students. A three-person team containing Alice is determined by selecting its other two members from the remaining seven: $\binom{7}{2} = 21$. If Bob must also be excluded, only six candidates remain for those two places, giving $\binom{6}{2} = 15$.

To contain at least one of Alice and Bob, exclude the teams containing neither:

$$\binom{8}{3} - \binom{6}{3} = 56 - 20 = 36.$$

Adding the Alice-containing and Bob-containing counts without correction would count teams containing both twice.

10.4 Counting via subsets

A final record may have ordered positions even when a combination is the appropriate count. Identify which set of positions or objects determines it.

Example 10.13. A length-5 bit string with exactly two 1s is determined by the two positions containing them. For example,

$$\{1,2\} \longleftrightarrow 11000, \qquad \{2,5\} \longleftrightarrow 01001.$$

There are $\binom{5}{2} = 10$ choices of position set, and hence ten strings. Choosing position 2 and then 5 produces the same string as choosing 5 and then 2. $P(5,2) = 20$ would count every string twice.

The string has ordered positions, but the order in which we select its one-positions is not part of the outcome.

Example 10.14. Form a three-letter string from $\{A, B, C, D, E, F\}$ with distinct letters in strictly increasing alphabetical order. Any three-element subset has exactly one allowed order: $\{B, D, F\}$ produces BDF, for instance. Thus the count is $\binom{6}{3} = 20$.

There is no extra factor of 3!. The other orders of the same three letters fail the alphabetical condition.

Example 10.15. Fix a labelled set of five vertices. In a simple undirected graph, an edge is a two-element subset of the vertices, so there are $\binom{5}{2} = 10$ possible edges. Each graph with exactly three edges is determined by a three-element subset of those ten possibilities. There are therefore

$$\binom{10}{3} = \frac{10(9)(8)}{3(2)(1)} = 120$$

such graphs. Retain all five labelled vertices, including any isolated ones. If any number of edges is allowed, Chapter 9's yes-or-no choices give $2^{10} = 1{,}024$ graphs.

10.5 Combinatorial proofs

❄ Definition 10.5 — Combinatorial proof

A **combinatorial proof** establishes a counting identity by counting one finite collection in two ways, or by giving a reversible correspondence between the objects counted by its two sides.

State what the objects are and explain why each expression counts them. A numerical check can illustrate the identity, but a proof must cover the stated range of parameters.

Example 10.16. Binomial symmetry. For integers $0 \le k \le n$, prove

$$\binom{n}{k} = \binom{n}{n-k}.$$

Fix an n-element set U. Every k-element subset S has a unique $(n-k)$-element complement $U \setminus S$. Conversely, complementing an $(n-k)$-element subset produces a k-element subset. Taking the complement twice returns the original set. The correspondence pairs every subset on one side with exactly one on the other, proving the identity.

For example, choosing nine of twelve records is equivalent to specifying the three left out: $\binom{12}{9} = \binom{12}{3} = 220$. The complement argument, rather than this one value, proves the formula for all allowed n, k.

Example 10.17. Pascal's identity. For $n \ge 0$ and $0 \le k \le n+1$, prove

$$\binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k}.$$

Consider k-element subsets of a set containing Alice and n other people. There are $\binom{n+1}{k}$ such subsets.

If a subset contains Alice, its remaining $k-1$ members are chosen from the n others: $\binom{n}{k-1}$ choices. If it does not contain Alice, all k members come from those n others: $\binom{n}{k}$ choices. The two cases are disjoint and cover all subsets. The sum rule therefore proves the identity.

This is **Pascal's identity**. Our zero convention handles the endpoints. For $k = 0$, the Alice-containing case is impossible; for $k = n + 1$, the Alice-excluding case is impossible. No negative factorial is used.

Example 10.18. Pascal's triangle. Arrange $\binom{n}{0}, \binom{n}{1}, \ldots, \binom{n}{n}$ in row n, starting with row 0. The first seven rows are:

$$
\begin{array}{ll}
n = 0 & \qquad\qquad\qquad\quad 1 \\
n = 1 & \qquad\qquad\quad 1 \qquad 1 \\
n = 2 & \qquad\qquad 1 \qquad 2 \qquad 1 \\
n = 3 & \qquad\quad 1 \qquad 3 \qquad 3 \qquad 1 \\
n = 4 & \qquad 1 \qquad 4 \qquad \boxed{6} \qquad \boxed{4} \qquad 1 \\
n = 5 & \quad 1 \qquad 5 \qquad 10 \qquad \boxed{10} \qquad 5 \qquad 1 \\
n = 6 & \; 1 \qquad 6 \qquad 15 \qquad 20 \qquad 15 \qquad 6 \qquad 1
\end{array}
$$

The outside entries are $\binom{n}{0} = \binom{n}{n} = 1$. Every interior entry is the sum of the two entries immediately above it, by Pascal's identity. The outlined entries illustrate $\binom{5}{3} = \binom{4}{2} + \binom{4}{3} = 6 + 4 = 10$. Complement symmetry explains why each row reads the same from either end.

Example 10.19. Teams with a leader. For $n \geq 1$ and $1 \leq k \leq n$, prove

$$k\binom{n}{k} = n\binom{n-1}{k-1}.$$

An outcome is a k-person team selected from n people, with one member designated as leader. Choose the team first in $\binom{n}{k}$ ways, then its leader in k ways. This gives the left side.

Alternatively, choose the leader in n ways, then the other $k-1$ members from the remaining $n-1$ people. This gives the right side. Each method specifies the same team and leader exactly once, so the counts are equal. For $n = 6, k = 3$, the two methods give $3\binom{6}{3} = 6\binom{5}{2} = 60$.

Example 10.20. Binomial row sum. For every integer $n \geq 0$, prove

$$\sum_{j=0}^{n} \binom{n}{j} = 2^n.$$

The sum adds the counts for sizes $0, 1, \ldots, n$. Fix an n-element set U. Its subsets split into disjoint cases according to their size. Adding the case counts gives the left-hand side.

Alternatively, for each element of U, decide whether to include it or exclude it. These n yes-or-no choices specify a subset once and give 2^n outcomes, as in Chapter 9. Both counts include every subset, including $\emptyset$, so they are equal. For $n = 0$, both sides are 1.

10.6 Row sums by induction

The counting proof explains the subsets. We can also derive the same identity from Pascal's identity using Chapter 5's ordinary induction.

Example 10.21. Let $P(n)$ be the statement $\sum_{j=0}^{n} \binom{n}{j} = 2^n$, for integers $n \geq 0$.

Base case. When $n = 0$, the sum has one term: $\binom{0}{0} = 1 = 2^0$.

Inductive step. Fix $m \geq 0$ and assume

$$\sum_{j=0}^{m} \binom{m}{j} = 2^m.$$

We must prove the corresponding row sum for $m + 1$. Pascal's identity, including the zero endpoint terms, gives

$$\sum_{j=0}^{m+1} \binom{m+1}{j} = \sum_{j=0}^{m+1} \binom{m}{j-1} + \sum_{j=0}^{m+1} \binom{m}{j}.$$

In the first sum on the right, $j = 0$ contributes $\binom{m}{-1} = 0$. The remaining terms are exactly $\binom{m}{0}, \binom{m}{1}, \ldots, \binom{m}{m}$. In the second sum, those same row-m terms appear, followed by $\binom{m}{m+1} = 0$. Each sum is therefore 2^m by the inductive hypothesis. The new row sum is

$$2^m + 2^m = 2^{m+1},$$

as required. Induction proves the identity for every $n \geq 0$. □

The extra endpoint terms were zero counts of impossible subset sizes. The inductive hypothesis was used only for the row indexed by m.

10.7 Binomial expansions

A **binomial** is a sum of two terms. When we multiply copies of $x + y$, distribution chooses one term from each copy. A coefficient records how many contributions produce the same powers of x and y.

Example 10.22. Expand $(x+y)^3 = (x+y)(x+y)(x+y)$. Label the factors $1, 2, 3$. Each contribution is determined by the positions from which y is chosen; the other factors supply x.

Positions supplying y	Product	Number of choices
$\varnothing$	x^3	1
$\{1\},\{2\},\{3\}$	x^2y	3
$\{1,2\},\{1,3\},\{2,3\}$	xy^2	3
$\{1,2,3\}$	y^3	1

For instance, selecting y from factors $1, 3$ and x from factor 2 contributes $yxy = xy^2$. Collecting all eight contributions gives

$$(x+y)^3 = x^3 + 3x^2y + 3xy^2 + y^3.$$

The coefficients are $\binom{3}{0}, \binom{3}{1}, \binom{3}{2}, \binom{3}{3}$.

❄ Definition 10.6 — Binomial theorem

For a nonnegative integer n, the polynomial identity

$$(x+y)^n = \sum_{j=0}^{n} \binom{n}{j} x^{n-j}y^j$$

is the **binomial theorem**. The coefficient of $x^{n-j}y^j$ is $\binom{n}{j}$. When $n = 0$, both sides are the constant polynomial 1.

For general n, choose the j factors supplying y. There are $\binom{n}{j}$ such position sets. Every one contributes $x^{n-j}y^j$, and every contribution with those powers comes from one of the sets. This proves the coefficient formula. Summing over $j = 0, \ldots, n$ accounts for all choices.

Example 10.23. In $(x+2)^4$, replace y by 2 in the theorem:

$$\begin{aligned}(x+2)^4 &= x^4 + 4x^3(2) + 6x^2(2^2) + 4x(2^3) + 2^4\\ &= x^4 + 8x^3 + 24x^2 + 32x + 16.\end{aligned}$$

For the x^2 term, choose two factors to supply 2 and two to supply x. Its coefficient is $\binom{4}{2}\,2^2 = 6(4) = 24$. Using just $\binom{4}{2} = 6$ would omit the contribution from the two selected constants.

Example 10.24. For $(1+3x)^5$, an x^2 contribution must select $3x$ from exactly two factors. Choose those factors in $\binom{5}{2} = 10$ ways. Each contribution supplies $(3x)^2 = 9x^2$, so the coefficient is $10(9) = 90$.

The same method retains signs. For the coefficient of x^4 in $(x-2)^5$, choose the one factor supplying -2. The other four supply x, giving coefficient $\binom{5}{1}(-2) = -10$. The binomial coefficient counts the position choices. The constants in each selected product supply its additional numerical factor.

10.8 Exercises

10.1. Evaluate or simplify. In part (f), n is an integer with $n \geq 0$.

a) $0!$.

b) $1!$.

c) $4!$.

d) $\dfrac{6!}{3!}$.

e) $\dfrac{8!}{6!\,2!}$.

f) $\dfrac{(n+1)!}{n!}$.

10.2. Alice, Bob and Charlie are available. List all speaking orders using all three people. Then list all two-person teams with no roles. Explain what constitutes one outcome in each task.

10.3. Six distinct jobs must be scheduled in a line. Count schedules with:

a) no restriction;

b) one particular job fixed first;

c) two particular jobs occupying the first two positions, in either order.

10.4. Evaluate:

a) $P(8,3)$.

b) $P(6,2)$.

c) $P(5,0)$.

d) $P(4,4)$.

e) $P(0,0)$.

How many ordered lists of five distinct objects can be formed from only three available objects? Explain why the factorial quotient cannot be used with $n = 3, k = 5$.

10.5. Use $\{0,1,2,3,4,5\}$ without repetition.

a) Count four-character codes, allowing a leading zero.

b) Count four-digit positive integers.

c) Count the codes beginning with zero. Explain how the three counts are related.

10.6. Let $F = \{p,q,r\}$ and $S = \{1,2,3,4,5,6\}$. Count functions $f : F \to S$ with:

a) no restriction;

b) injectivity;

c) injectivity and $f(p) = 1$.

Explain the available choices of image at each stage.

10.7. List the two-element subsets of $\{a,b,c,d,e\}$. For each, write its two ordered pairs. Use the grouping to justify the relation between $P(5,2)$ and $\binom{5}{2}$.

10.8. Evaluate, using symmetry where convenient. Use the zero convention rather than negative factorials.

a) $\binom{9}{2}$.

b) $\binom{9}{7}$.

c) $\binom{6}{0}$.

d) $\binom{6}{6}$.

e) $\binom{0}{0}$.

f) $\binom{4}{5}$.

g) $\binom{4}{-1}$.

10.9. Eight students are available. Count outcomes for:

a) selecting a three-person team with no roles;

b) assigning different students to three named roles;

c) selecting a three-person team with one designated leader.

Describe why the outcomes differ.

10.10. There are five members of Group A and four of Group B, with no overlap. Count four-person teams containing:

a) exactly two members of each group;

b) at least one member of Group B;

c) at most one member of Group B.

Explain each product, sum or complement used.

10.11. Alice and Bob are among ten available students. Count four-person teams satisfying:

a) Alice is included.

b) Alice is included and Bob excluded.

c) Alice and Bob are both included.

d) At least one of Alice and Bob is included.

10.12. Give the count and explain the underlying subset in each case.

a) Three distinct letters from $\{A, B, C, D, E, F, G, H\}$ in strictly increasing alphabetical order.

b) A length-6 bit string with exactly two 1s.

Why is a permutation count inappropriate for each task?

10.13. Consider length-8 bit strings with exactly three 1s. Count those:

a) with no additional restriction;

b) starting with 1;

c) ending with 00;

d) starting with 1 or ending with 00.

In part (d), identify the intersection and state how many positions remain free there.

10.14. Fix the labelled vertex set $V = \{a, b, c, d, e, f\}$. Graphs are simple and undirected.

a) How many edges are available?

b) How many graphs on V have exactly three edges?

c) How many of those contain $\{a, b\}$?

d) How many of those omit $\{a, b\}$?

Check that parts (c) and (d) add to part (b). Keep all six labelled vertices in every graph.

10.15. Group the four strings

$$00, \quad 01, \quad 10, \quad 11$$

according to their number of 1s. Grace divides 4 by 2, arguing that 01 and 10 describe the same group. List the actual groups and explain why her division does not count them.

10.16. For $U = \{a, b, c, d, e\}$, describe the complement correspondence between three-element and two-element subsets, and give the image of $\{a, c, e\}$. Explain its reverse, then use the same argument to prove $\binom{n}{k} = \binom{n}{n-k}$ for all integers $0 \le k \le n$.

10.17. For $n \ge 0$ and $0 \le k \le n+1$, consider all length-$(n+1)$ bit strings with exactly k ones. Classify them by their first bit and prove

$$\binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k}.$$

Explain how the zero convention handles $k = 0$ and $k = n+1$.

10.18. Use Pascal's identity to construct rows 7 and 8 of Pascal's triangle, with the top row numbered 0. Check the middle entry of row 8 by computing $\binom{8}{4}$ from factorials. Explain why every row has matching entries at equal distances from its ends.

10.19. For $1 \le k \le n$, an outcome consists of a length-n bit string with exactly k ones, and one of those one-positions marked. Count by choosing the string first and by choosing the marked position first. Use the two counts to prove

$$k\binom{n}{k} = n\binom{n-1}{k-1}.$$

Explain why the marked position is excluded from the later choice of $k-1$ positions.

10.20. For an n-element set, with $n \ge 2$:

a) Explain why $\sum_{j=0}^{n} \binom{n}{j}$ counts all subsets.

b) How many nonempty subsets are there? Identify the term omitted from the full sum.

c) How many subsets have at least two elements? Explain the additional term removed.

10.21. Prove $\sum_{j=0}^{n} \binom{n}{j} = 2^n$ for all $n \ge 0$ by ordinary induction, using Pascal's identity. State the base case and the inductive hypothesis. In the step from m to $m+1$, identify the two zero endpoint terms before applying the hypothesis.

10.22. Use the binomial theorem to expand:

a) $(x+1)^5$. b) $(2x+1)^4$.

Explain one non-endpoint coefficient in each expansion by describing the choices from the factors.

10.23. Find the coefficient of:

a) x^3 in $(1+2x)^6$;

b) x^4 in $(x+3)^7$;

c) x^5 in $(2x-1)^6$.

Identify which terms are selected from how many factors, including any constant powers and signs.

10.24. a) Mike gives $7!/3!$ for the number of three-element subsets of seven records. What ordered arrangements does that quotient count? Give the correct subset count.

b) Judy says that the coefficient of x^2 in $(1+2x)^4$ is $\binom{4}{2}$. Identify the missing factor and give the coefficient.

10.25. Fix a labelled set of five vertices. Use the ten possible edges to explain combinatorially why

$$\sum_{j=0}^{10} \binom{10}{j} = 2^{10}.$$

Describe the graph outcomes counted by each term and why the cases are disjoint and exhaustive.

10.26. David proposes

$$3\binom{8}{3} = 8\binom{7}{3},$$

claiming that both sides count three-person teams with one designated leader. Describe the outcomes actually counted by each side. Correct the identity and explain how the second construction must change.

LECTURE NOTES
DISCRETE MATHEMATICS

CHAPTER 11

Sequences



Updated: 19 November 2026

11 Sequences

Before any import batches run, a database contains five records. Each batch adds three records. To find the count after fifty batches, we could update the count fifty times, or calculate the initial count plus fifty additions of three. These are two ways to describe the same indexed quantity.

The index records how many batches have finished. Index 0 describes the initial snapshot, before any batch, so the value after three batches is the fourth entry if we list that initial snapshot too.

Chapter 5 used recursive rules and induction; Chapters 9 and 10 counted objects depending on a size parameter. We now study such numbered values as sequences. We will specify their indices, compare direct formulas with rules using earlier values, and prove that a proposed formula describes every term.

What you should be able to do

After working through this chapter, you should be able to:

- interpret a sequence as a function of an integer index;
- distinguish a term, its index and the sequence's set of values;
- evaluate explicit formulas and recursive descriptions;
- supply initial values and valid ranges for a recurrence;
- describe arithmetic and geometric sequences with either type of rule;
- translate simple updating processes into mathematical descriptions;
- verify a proposed formula and identify incomplete specifications.

11.1 Sequences and indices

❄ Definition 11.1 — Numerical sequence

Fix a starting index $s \in \mathbb{N}$, where $\mathbb{N} = \{0, 1, 2, \ldots\}$. A numerical sequence $(a_n)_{n \geq s}$ assigns a real number a_n to each index in $I_s = \{s, s+1, s+2, \ldots\}$. It is a function $a : I_s \to \mathbb{R}$, with $a_n = a(n)$. The value a_n is the **term at index** n.

We usually start at 0 or 1, but state the choice. A subscript labels an input: a_n does not mean a multiplied by n. Order and repeated values are retained. A finite sequence similarly uses a finite consecutive set of indices, such as $\{0, 1, \ldots, 6\}$.

Example 11.1. For the database in the opening, let a_n be the number of records after n batches. The first five terms are:

Index n	0	1	2	3	4
Records a_n	5	8	11	14	17

Thus $a_3 = 14$ is the count after three batches. It is the fourth displayed term because the list begins with a_0. The third displayed term is $a_2 = 11$.

Example 11.2. The sequences

$$2, 5, 2, 5, \ldots \qquad \text{and} \qquad 5, 2, 5, 2, \ldots$$

both start at index 0, alternating as shown. Their sets of values are both $\{2, 5\}$. Nevertheless, they are different functions: their values at index 0 differ. Repeating a value is allowed; a sequence need not be injective.

Likewise, the constant sequence $7, 7, 7, \ldots$ has infinitely many indexed terms but a set of values containing just one element.

Example 11.3. For $(a_n)_{n \geq 0}$ in Example 11.1, define $(b_n)_{n \geq 1}$ by $b_n = a_{n-1}$. Then

$$b_1 = 5, \quad b_2 = 8, \quad b_3 = 11, \quad b_4 = 14.$$

Both lists display the same values in the same order, but their index sets differ. If $a_n = 5 + 3n$, then

$$b_n = 5 + 3(n-1) \qquad (n \geq 1).$$

Substituting $n = 1$ gives the correct first value 5. The functions are reindexed descriptions, rather than literally the same function on the same domain.

Example 11.4. Let $a_n = n^2 + 1$ for $n \geq 0$. Then $a_2 = 5$ and $a_3 = 10$. More generally,

$$a_{n+1} = (n+1)^2 + 1 = n^2 + 2n + 2, \qquad a_n + 1 = n^2 + 2.$$

For example, $a_{2+1} = 10$, whereas $a_2 + 1 = 6$. The parentheses around $n + 1$ are needed when substituting the new index into the formula.

11.2 Explicit and recursive descriptions

❄ Definition 11.2 — Explicit formula

An **explicit formula** gives a_n directly as a function of its index n, without using earlier values of the sequence. Its index domain is part of the description.

For example, $a_n = 5 + 3n$ for $n \geq 0$ allows us to calculate a term by substituting its index. A finite list of values alone does not specify which formula or continuing rule is intended.

Example 11.5. For $n \geq 0$, define

$$u_n = n + 1, \qquad v_n = n + 1 + n(n-1)(n-2).$$

At $n = 0, 1, 2$, at least one factor of the extra product is zero. The formulas agree there, but they differ at the next index:

n	0	1	2	3
u_n	1	2	3	4
v_n	1	2	3	10

Thus the prefix 1, 2, 3 does not force the next value to be 4. A stated arithmetic rule would add information; the three observations by themselves do not. More generally, a finite prefix can be followed by different continuations.

❄ Definition 11.3 — Recursive description

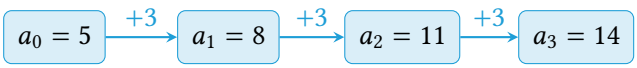
A **recursive description** specifies initial values and a **recurrence relation**: a rule computing later terms from earlier ones. State the indices on which the sequence is defined and the range on which the rule applies. The initial values and rule must allow every term in the domain to be determined.

The rule describes a transition. The initial values supply the starting information. Earlier here means a smaller index.

Example 11.6. The complete recursive description for Example 11.1 is

$$a_0 = 5, \qquad a_{n+1} = a_n + 3 \quad (n \geq 0).$$

It gives $a_1 = 5 + 3 = 8$, then $a_2 = 8 + 3 = 11$, then $a_3 = 11 + 3 = 14$.



The explicit formula $a_n = 5 + 3n$ gives $a_{10} = 35$ directly. The displayed recursive rule calculates that term by continuing the successive updates from a_0. The two descriptions will be justified by the constant-difference reasoning below.

Python (non-examinable)

The list a starts with $a_0 = 5$. Python counts list positions from zero, so `a[n]` represents a_n here. The loop uses n from 0 through 9; each call to append adds the next term to the list.

```
a = [5]
for n in range(10):
    a.append(a[n] + 3)
print(a)
print(a[10], 5 + 3 * 10)
```

Output.

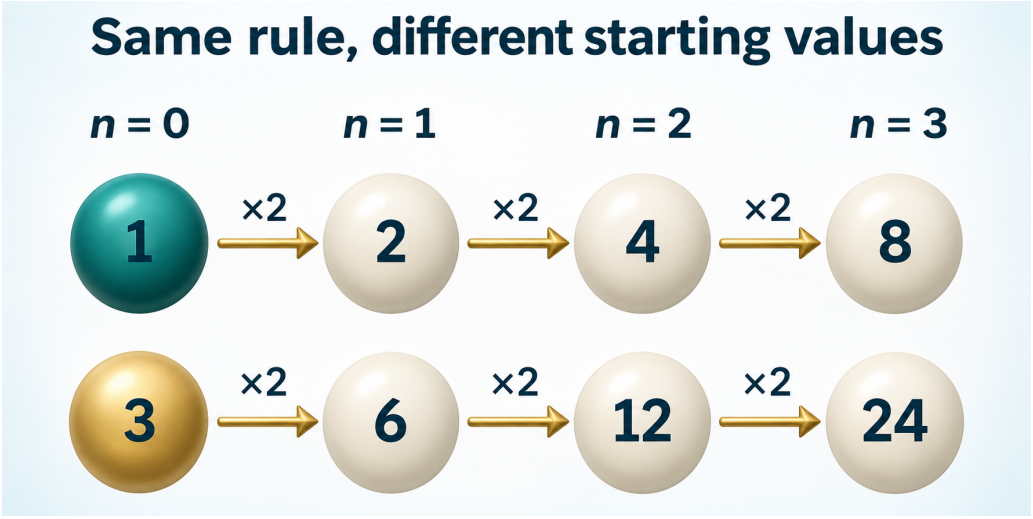
```
[5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35]
35 35
```

The last line compares the term obtained by successive updates with the explicit formula at $n = 10$; * denotes multiplication. A recursive description can be implemented by a loop: the program need not call itself. This finite calculation does not establish the formula for every index.

Example 11.7. Both sequences below use the rule "double the previous term", starting at index 0:

$$1, 2, 4, 8, \ldots \qquad \text{and} \qquad 3, 6, 12, 24, \ldots .$$

They satisfy $a_{n+1} = 2a_n$, but have different initial values. Specifying $a_0 = 1$ selects the first; specifying $a_0 = 3$ selects the second. The recurrence equation alone does not distinguish them.



Both sequences double the preceding term, but starting at 1 or 3 gives different values at every displayed index.

Example 11.8. The Fibonacci sequence. Define

$$f_0 = 0, \qquad f_1 = 1, \qquad f_{n+2} = f_{n+1} + f_n \quad (n \geq 0).$$

This is the Fibonacci sequence with its stated starting indices. First $f_2 = f_1 + f_0 = 1$, then $f_3 = f_2 + f_1 = 2$, and $f_4 = f_3 + f_2 = 3$. Continuing gives:

n	0	1	2	3	4	5	6
f_n	0	1	1	2	3	5	8

At the first allowed step $n = 0$, both required inputs f_1, f_0 are already known. Each subsequent step uses two values that have already been determined.

Python (non-examinable)

The list `f` starts with $f_0 = 0$ and $f_1 = 1$. Its positions are numbered from zero, so `f[n]` represents f_n. The loop takes n from 0 through 4, and `append` adds the newly computed term to the end of the list. At each step, the two required entries already exist.

```
f = [0, 1]
for n in range(5):
    f.append(f[n + 1] + f[n])
print(f)
print(f[6])
```

Output.

```
[0, 1, 1, 2, 3, 5, 8]
8
```

The first loop step adds $f_2 = f_1 + f_0$; the last adds $f_6 = f_5 + f_4$. Starting with just `[0]` would leave `f[1]` unavailable at the first step.

Example 11.9. The specification

$$c_0 = 0, \qquad c_{n+2} = c_{n+1} + c_n \quad (n \geq 0)$$

leaves c_1 unspecified. Choosing $c_1 = 1$ produces $0, 1, 1, 2, 3, \ldots$; choosing $c_1 = 2$ produces $0, 2, 2, 4, 6, \ldots$. Both follow the same stated rule and value of c_0. They are different sequences, so more initial information is needed.

There is also an index problem in writing $d_0 = 2$ and $d_n = 3d_{n-1}$ for $n \geq 0$, if the domain is $\mathbb{N}$. At $n = 0$ it refers to d_{-1}, outside that domain. Use the rule for $n \geq 1$, or write equivalently

$$d_0 = 2, \qquad d_{n+1} = 3d_n \quad (n \geq 0).$$

The corrected first step computes d_1 from d_0.

11.3 Arithmetic sequences

❄ Definition 11.4 — Arithmetic sequence

A sequence starting at s is **arithmetic** if a fixed real number d, its **common difference**, satisfies

$$a_{n+1} = a_n + d \qquad (n \geq s).$$

If its initial term is $a_s = A$, its explicit formula is

$$a_n = A + (n - s)d \qquad (n \geq s).$$

There are $n - s$ transitions from index s to index n. Each adds d, so the total change from the initial term is $(n - s)d$. The difference can be positive, negative or zero. For a finite arithmetic sequence, apply the rule only while both consecutive indices are in its domain.

Example 11.10. For the imports, $a_0 = 5$ and every batch adds $d = 3$. Thus

$$a_n = 5 + 3n \qquad (n \geq 0).$$

After fifty batches the count is $a_{50} = 5 + 3(50) = 155$. The fifty additions occur after the initial snapshot; there is no addition at index 0.

Example 11.11. Rows of seats are numbered $1, 2, 3, \ldots$. Row 1 has 12 seats, and each later row has four more than its predecessor. The recursive and explicit descriptions are

$$a_1 = 12, \quad a_{n+1} = a_n + 4 \ (n \geq 1); \qquad a_n = 12 + 4(n - 1) \ (n \geq 1).$$

Row 10 has $12 + 4(9) = 48$ seats. There are nine transitions from row 1 to row 10. The formula $12 + 4n$ would already give the wrong value at $n = 1$.

Example 11.12. A queue begins with 20 records. Each full round processes exactly three records; such a round runs only when at least three remain. After n full rounds, the count is

$$q_n = 20 - 3n \qquad (0 \leq n \leq 6).$$

The values are $20, 17, 14, 11, 8, 5, 2$. The recursive description is $q_0 = 20$ and $q_{n+1} = q_n - 3$ for $0 \leq n \leq 5$.

At $n = 6$, only two records remain. Extending the formula to $n = 7$ would give -1, which cannot represent a number of waiting records. Finishing with a partial round would require a different final update. The algebraic formula can extend to negative values; this counting model cannot.

Example 11.13. Suppose an arithmetic sequence starting at 0 has $a_2 = 11$ and $a_5 = 23$. Three transitions occur between these indices, so

$$23 - 11 = 3d, \qquad d = 4.$$

Then $a_0 = a_2 - 2d = 11 - 8 = 3$. The complete descriptions are

$$a_0 = 3, \quad a_{n+1} = a_n + 4 \ (n \geq 0); \qquad a_n = 3 + 4n \ (n \geq 0).$$

We used the stated arithmetic assumption. The two observed values alone would not force the intervening or later values.

11.4 Geometric sequences

❄ Definition 11.5 — Geometric sequence

A sequence starting at s is **geometric** when a fixed real multiplier r satisfies

$$a_{n+1} = ra_n \qquad (n \geq s).$$

For $r \neq 0$ and $a_s = A$, its explicit formula is

$$a_n = Ar^{n-s} \qquad (n \geq s).$$

Whenever $a_n \neq 0$, the multiplier is the **common ratio** $r = a_{n+1}/a_n$.

There are $n - s$ multiplications by r after the initial term. The multiplier definition remains meaningful when terms are zero, even though division by a zero term is not allowed. For $r = 0$, retain the stated initial term and write all later terms as zero; we treat that case separately from the displayed power formula.

Example 11.14. Let b_n count length-n bit strings. There is one length-0 string, the empty string, so $b_0 = 1$. Each length-n string extends in exactly two ways by appending 0 or 1. Different prefixes or final bits give different extended strings, and every length-$(n + 1)$ string is obtained once. Thus

$$b_0 = 1, \qquad b_{n+1} = 2b_n \quad (n \geq 0).$$

This is geometric with $r = 2$, giving $b_n = 2^n$. It agrees with Chapter 9's product-rule count.

Example 11.15. Let $a_1 = 81$ and $a_{n+1} = \frac{1}{3}a_n$ for $n \geq 1$. The sequence begins $81, 27, 9, 3, 1, \frac{1}{3}, \ldots$. Its explicit formula is

$$a_n = 81\left(\frac{1}{3}\right)^{n-1} \qquad (n \geq 1).$$

In particular, $a_5 = 81/3^4 = 1$. The exponent counts four multiplications between indices 1 and 5. Our numerical sequences may have real values; they are not all counting sequences.

Example 11.16. Let $a_0 = 2$ and $a_{n+1} = -3a_n$ for $n \geq 0$. The first terms are

$$2, -6, 18, -54, 162, \ldots, \qquad a_n = 2(-3)^n.$$

The signs alternate because each update multiplies by a negative number. Keep the parentheses: $2(-3)^2 = 18$. The expression $-2(3^2) = -18$ has a different sign and does not describe the same term.

Example 11.17. The constant sequence $7, 7, 7, \ldots$ is arithmetic with difference 0 and geometric with multiplier 1. The two types can overlap.

The all-zero sequence is also arithmetic with difference 0. It satisfies $a_{n+1} = ra_n$ for every fixed real r, so it is geometric under our multiplier definition. The quotient $0/0$ does not identify a ratio.

If instead $a_0 = 7$ and the multiplier is $r = 0$, the sequence is $7, 0, 0, \ldots$. Write its explicit description as

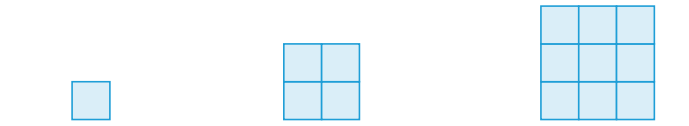
$$a_0 = 7, \qquad a_n = 0 \quad (n \geq 1).$$

No evaluation of a zeroth power of zero is needed. Once a geometric sequence reaches zero, a fixed multiplier keeps every later term at zero.

11.5 Further recurrence relations

Not every numerical sequence has a fixed difference or a fixed multiplier. A recurrence can still describe a changing update precisely.

Example 11.18. At stage $n \geq 0$, a square array has $n + 1$ rows and $n + 1$ columns. Let s_n count its cells. The explicit formula is $s_n = (n + 1)^2$.



$n = 0 : \ s_0 = 1$ $\qquad$ $n = 1 : \ s_1 = 4$ $\qquad$ $n = 2 : \ s_2 = 9$

To grow the side from $n + 1$ to $n + 2$, add a row of $n + 1$ cells, a column of $n + 1$ cells and the new corner. The number added is $2n + 3$. Thus another description is

$$s_0 = 1, \qquad s_{n+1} = s_n + 2n + 3 \quad (n \geq 0).$$

The first terms $1, 4, 9, 16$ have differences $3, 5, 7$, so the sequence is not arithmetic. It is not geometric either: the first two successive ratios are 4 and $9/4$, which differ.

Example 11.19. A process starts with one record. At each round, it replaces each current record by two records and then adds one new record. Let a_n count records after n rounds. The description is

$$a_0 = 1, \qquad a_{n+1} = 2a_n + 1 \quad (n \geq 0).$$

It gives $1, 3, 7, 15, 31, \ldots$. The factor 2 describes the replacements; the added 1 describes the extra record. This is not the geometric rule $a_{n+1} = 2a_n$.

The values suggest the explicit formula $a_n = 2^{n+1} - 1$. It matches the terms just calculated, but we must justify it for every index rather than rely on that finite match.

11.6 Verification and comparison

A candidate formula must agree with all prescribed initial values and with each permitted update. Induction connects those checks to a claim about the whole sequence.

Example 11.20. For Example 11.19, prove $a_n = 2^{n+1} - 1$ for every $n \geq 0$.

Proof. Let $P(n)$ state this equality.

Base case. At $n = 0$, $a_0 = 1 = 2^{0+1} - 1$.

Inductive step. Fix $k \geq 0$ and assume $a_k = 2^{k+1} - 1$. The recurrence then gives

$$\begin{aligned} a_{k+1} &= 2a_k + 1 \\ &= 2(2^{k+1} - 1) + 1 \\ &= 2^{k+2} - 1 \\ &= 2^{(k+1)+1} - 1. \end{aligned}$$

This is $P(k+1)$. Ordinary induction proves the formula for all $n \geq 0$. □

We may now use it at an index not checked in the initial list: for example, $a_6 = 2^7 - 1 = 127$.

Example 11.21. The formula $b_n = 2^n - 1$ also satisfies the update equation:

$$2b_n + 1 = 2(2^n - 1) + 1 = 2^{n+1} - 1 = b_{n+1}.$$

But $b_0 = 0$, not the required $a_0 = 1$. Its terms begin $0, 1, 3, 7, \ldots$, so it does not describe the sequence of Example 11.19. Checking the update while omitting the initial value would not establish the proposed answer.

Example 11.22. Chapter 5 studied

$$c_0 = 1, \quad c_1 = 3, \quad c_{n+2} = 2c_{n+1} - c_n \quad (n \geq 0)$$

and proved $c_n = 2n + 1$ for every $n \geq 0$, using both initial values in the induction. The first terms are $1, 3, 5, 7, \ldots$. The explicit formula shows

$$c_{n+1} - c_n = (2n + 3) - (2n + 1) = 2.$$

Consequently, the shorter recursive description $c_0 = 1$, $c_{n+1} = c_n + 2$ for $n \geq 0$ gives the same sequence. Having two earlier terms in a rule does not prevent its sequence from being arithmetic.

More generally, the rules considered here determine later terms from known earlier terms. If two candidate descriptions agree on the required initial values and obey the same such update, agreement propagates to every later index. For a rule using two earlier terms, both starting values must be checked.

11.7 Exercises

11.1. Let $a_n = n^2 + 2$ for $n \geq 0$.

a) List the first five terms with their indices.

b) Find a_3 and a_5. Which displayed position does a_3 occupy?

c) Simplify a_{n+1} and $a_n + 1$ separately. Show that they do not denote the same expression.

11.2. For $n \geq 0$, let $a_n = (-1)^n$ and $b_n = (-1)^{n+1}$. Give the first four terms of each and their sets of values. Are the indexed functions equal? Give an index witnessing your answer.

11.3. Let $a_n = 4 + 5n$ for $n \geq 0$ and define $b_n = a_{n-1}$ for $n \geq 1$.

a) Give an explicit formula for b_n.

b) Find b_1 and b_5.

c) Explain why the two descriptions list the same ordered values but have different domains.

11.4. List the first four terms and their indices:

a) $a_n = 3(2^n)$, $n \geq 0$.

b) $b_n = 3(2^{n-1})$, $n \geq 1$.

c) $c_n = 10 - 2n$, $n \geq 0$.

For part (c), the values are real numbers; no nonnegative-count restriction is imposed.

11.5. Let $a_0 = 2$ and $a_{n+1} = a_n + 5$ for $n \geq 0$. Calculate $a_1, \ldots, a_4$ recursively. Give an explicit formula and use it to find a_{10}. Check that it gives the specified initial value.

11.6. Define $u_0 = 1$ and $u_{n+1} = u_n + 2(n + 1)$ for $n \geq 0$. Find u_1, u_2, u_3, u_4. Is this sequence arithmetic? Explain why a formula based on a constant increment would be inappropriate.

11.7. Let

$$c_0 = 2, \quad c_1 = 1, \quad c_{n+2} = c_{n+1} + c_n \quad (n \geq 0).$$

Find $c_2, \ldots, c_6$, displaying the two earlier terms used in each step. Explain why the first allowed step requires no index below 0.

11.8. Someone specifies only $a_0 = 1$ and $a_{n+2} = a_{n+1} + a_n$ for $n \geq 0$. Choose two different possible values of a_1, and generate three further terms for each. Use them to show why the given data do not determine a unique sequence.

11.9. A sequence has domain $\mathbb{N}$, and is described by $a_0 = 4$ and $a_n = 2a_{n-1}$ for $n \geq 0$. Identify the undefined index used at the first permitted step. Give two equivalent repairs, one with a_n on the left and one with a_{n+1}. Using the repaired description, find a_3.

11.10. Consider these proposed descriptions on $\mathbb{N}$:

a) $a_0 = 1$ and $a_{n+1} = a_{n+1} + 1$ for $n \geq 0$.

b) $b_0 = 1$ and $b_{n+1} = b_{n+1}$ for $n \geq 0$.

Which is contradictory, and which leaves later terms undetermined? Explain why neither is a rule computing the next value from earlier values.

11.11. The sequences below are assumed arithmetic. For each, give its initial value, common difference, explicit formula and recursive rule, including the valid ranges.

a) $7, 11, 15, \ldots$, with first index 0.

b) $20, 17, 14, \ldots$, with first index 1.

c) $-2, -2, -2, \ldots$, with first index 0.

d) $9, 12, 15, \ldots$, with first index 2.

11.12. An arithmetic sequence starting at 0 satisfies $a_3 = 17$ and $a_8 = 42$. Find its common difference and a_0. Give an explicit formula and compute a_{15}.

11.13. A queue starts with 23 records. Each full round processes exactly four records, and a full round runs only if at least four remain. Describe the count after n full rounds explicitly and recursively. State the largest permitted n. If a final partial round then processes all remaining records, does that last step follow the same constant-difference rule? Explain.

11.14. For each geometric sequence, give the first four terms and an explicit description on its stated domain.

a) $a_0 = 5$, $r = 3$, $n \geq 0$.

b) $b_1 = 64$, $r = \frac{1}{2}$, $n \geq 1$.

c) $c_0 = -1$, $r = -2$, $n \geq 0$.

d) $d_0 = 9$, $r = 0$, $n \geq 0$.

11.15. A real geometric sequence starting at 0 has $g_0 = 3$ and $g_2 = 12$. Find all possible real multipliers and give the corresponding explicit formulas. Do the two supplied values determine g_1 uniquely?

11.16. a) Find the sequence with $a_0 = 0$ and $a_{n+1} = 5a_n$ for $n \geq 0$.

b) Could a geometric sequence begin $1, 0, 2$? Justify your answer from the fixed-multiplier rule.

c) Does the all-zero sequence determine a unique multiplier? Explain why term ratios do not answer that question.

11.17. For $n \geq 0$, decide whether each sequence is arithmetic, geometric, both or neither.

a) $a_n = 6 + 2n$.

b) $b_n = 3(-2)^n$.

c) $c_n = 4$.

d) $d_n = n^2$.

For a claimed property, justify the constant difference or multiplier for every allowed index. For a failed property, give incompatible steps.

11.18. In both models, index 0 is before any cycle.

a) There are initially four stored images; each cycle adds six, with none removed. Give both descriptions and the count after eight cycles.

b) There are initially four copies; each round replaces every copy by three copies. Give both descriptions and the count after three rounds.

11.19. At stage $n \geq 0$, an array has $n+1$ rows and $n+2$ columns. Let r_n count its cells.

a) Give an explicit formula and its first four terms.

b) Show that $r_{n+1} - r_n = 2n + 4$, and give a complete recursive description.

c) Is the sequence arithmetic or geometric? Support each decision with differences or multipliers.

11.20. Let $a_0 = 2$ and $a_{n+1} = 2a_n + 3$ for $n \geq 0$. Compute the first four terms, then prove

$$a_n = 5(2^n) - 3 \qquad (n \geq 0).$$

State the base case, the hypothesis and the target at index $k+1$.

11.21. For the sequence of Example 11.19, someone proposes $b_n = 3(2^n) - 1$. Show that $b_{n+1} = 2b_n + 1$, but explain why this formula does not describe the given sequence. Identify the missing part of the proposed verification.

11.22. For $n \geq 0$, let

$$u_n = 2n + 1, \qquad v_n = 2n + 1 + n(n-1)(n-2)(n-3).$$

Show that the first four terms agree, but the terms at index 4 differ. What does this show about a claim based only on observing $1, 3, 5, 7$?

11.23. Let $f_n = 7 + 4n$ and $g_n = 3 + 4(n+1)$, both for $n \geq 0$. Also let $h_n = 7 + 4(n-1)$ for $n \geq 1$.

a) Prove that f and g are the same indexed function.

b) Explain how h lists the same ordered values with different indices.

c) Compare f_3, g_3, h_3.

11.24. Let

$$c_0 = 2, \quad c_1 = 5, \quad c_{n+2} = 2c_{n+1} - c_n \quad (n \geq 0).$$

Verify that $c_n = 3n+2$ matches both initial values and the recurrence equation. Explain, using induction, why those checks establish the formula for every $n \geq 0$. Give another complete recursive description that uses only the immediately preceding term.

11.25. For the sequence in Exercise 11.6, prove by ordinary induction that

$$u_n = n^2 + n + 1 \qquad (n \geq 0).$$

Show explicitly how the added $2(k+1)$ turns the hypothesis into the target at $k+1$.

11.26. Let t_n be the number of length-n bit strings with no two adjacent 1s. Include the empty string when $n = 0$.

a) Find t_0, t_1, t_2, t_3 by listing.

b) Classify strings of length $n + 2$ by their final bit. If the final bit is 1, what must the preceding bit be? Use the two cases to justify $t_{n+2} = t_{n+1} + t_n$ for $n \geq 0$.

c) Give the complete recursive description and use it to find t_5.

LECTURE NOTES
DISCRETE MATHEMATICS

CHAPTER 12

Recursion



Updated: 19 November 2026

12 Recursion

A task either finishes immediately or splits into exactly two smaller tasks, each of which follows the same rule. Suppose every resulting task eventually finishes. How does the number of finished end tasks compare with the number of splits? A tree drawing can record one execution, but we want an answer covering every possible finite pattern of splits.

We will define counting functions on these trees and prove a relationship between their values. First we sharpen the function language from Chapters 1 and 4, including what it means to cover every output and how to find all inputs leading to a chosen output. The tree construction then brings together Chapter 6's rooted trees, Chapter 11's recursive descriptions and Chapter 5's induction.

What you should be able to do

After working through this chapter, you should be able to:

- distinguish injectivity, surjectivity and bijectivity;
- calculate images and inverse images on stated domains;
- distinguish an inverse image from an inverse function;
- describe full binary trees through a base object and a constructor;
- translate a recursive counting procedure into a mathematical function;
- trace recursive evaluation on a finite tree;
- prove a tree property using both subtree hypotheses in structural induction.

12.1 Injectivity and surjectivity

A function $f : X \to Y$ assigns exactly one output in Y to each input in X. The **domain** X lists the permitted inputs; the **codomain** Y lists the permitted outputs. Its **range** $f(X) = \{f(x) : x \in X\}$ is the set of outputs actually obtained, which may be smaller than Y.

Chapter 4 studied injectivity. We now place that property beside surjectivity, which asks whether any element of the codomain is missed.

❄ Definition 12.1 — Injective, surjective and bijective

Let $f : X \to Y$. The function is **injective** if equal outputs force equal inputs:

$$\text{for all } a, b \in X, \quad f(a) = f(b) \Longrightarrow a = b.$$

It is **surjective**, or **onto**, if

$$\text{for every } y \in Y, \text{ there is an } x \in X \text{ with } f(x) = y.$$

It is **bijective** if it is both injective and surjective.

For a particular output in Y, injectivity permits at most one input leading to it; surjectivity requires at least one. A bijection gives exactly one. Every function already gives one output per input. That requirement alone does not settle either property.

Example 12.1. Use the following new data. Let $U = \{\text{Alice}, \text{Bob}, \text{Charlie}, \text{David}\}$, $J = \{10, 20, 30, 40, 50\}$ and $P = \{\text{Printer 1}, \text{Printer 2}, \text{Printer 3}\}$. The complete table defines $i : U \to J$ and $d, h : U \to P$.

Input	Alice	Bob	Charlie	David
i	10	20	30	40
d	Printer 1	Printer 1	Printer 2	Printer 3
h	Printer 1	Printer 1	Printer 2	Printer 2

The identifier function i is injective: every input has a different output. It is not surjective onto J, because $50 \in J$ is not the output of any user. One missed codomain element is enough to refute surjectivity.

Example 12.2. The function d in Example 12.1 is surjective: Printer 1 is reached by Alice, Printer 2 by Charlie, and Printer 3 by David. These witnesses account for every element of P. But d is not injective, since Alice and Bob are distinct and share Printer 1.

Surjectivity asks whether an output is reached, not whether it is reached only once.

Example 12.3. Keep the identifier assignments, but define $j : U \to J'$ with $J' = \{10, 20, 30, 40\}$ and $j(u) = i(u)$. The outputs remain distinct, so j is injective. They also cover every element of the stated codomain J', so j is surjective and therefore bijective.

The rules for i and j agree at every input. The codomain change alters the surjectivity question. We have specified a function with a new codomain, not proved that the unused value 50 was reached.

Example 12.4. For h in Example 12.1, Alice and Bob share Printer 1, so h is not injective. No user is assigned Printer 3, so h is not surjective.

Together, i, d, j, h show all four possibilities: injective only, surjective only, both, and neither. Injectivity and surjectivity are not opposites.

Example 12.5. Let $g : \mathbb{Z} \to \mathbb{Z}$ be given by $g(k) = k + 3$. Prove that g is bijective.

Injectivity. If $g(a) = g(b)$ for integers a, b, then $a + 3 = b + 3$, hence $a = b$.

Surjectivity. Let y be an arbitrary integer in the codomain. Choose $x = y - 3$, which is an integer and therefore lies in the domain. Then $g(x) = (y - 3) + 3 = y$. Every permitted output has an input witness, so g is onto. Both properties hold. □

For example, the witness for output -2 is input -5. The general choice $x = y - 3$ covers all outputs; the single numerical example would not do so.

12.2 Images and inverses

Recall that for $A \subseteq X$, the **image** under $f : X \to Y$ is

$$f(A) = \{f(a) : a \in A\} \subseteq Y.$$

An image follows inputs forwards. An inverse image instead collects all inputs whose outputs belong to a specified subset of Y.

❄ Definition 12.2 — Inverse image

For $f : X \to Y$ and $B \subseteq Y$, its **inverse image**, or **preimage**, is

$$f^{-1}(B) = \{x \in X : f(x) \in B\} \subseteq X.$$

This notation defines a set of inputs for any function, whether or not it is injective or surjective.

To find the inputs leading to one output y, use the singleton set $\{y\}$ and write $f^{-1}(\{y\})$. We retain the braces so that a set of inputs is visibly distinguished from an inverse-function value. The superscript -1 here does not mean a reciprocal.

Example 12.6. Use $h : U \to P$ from Example 12.1. For $A = \{\text{Alice}, \text{Bob}, \text{Charlie}\}$,

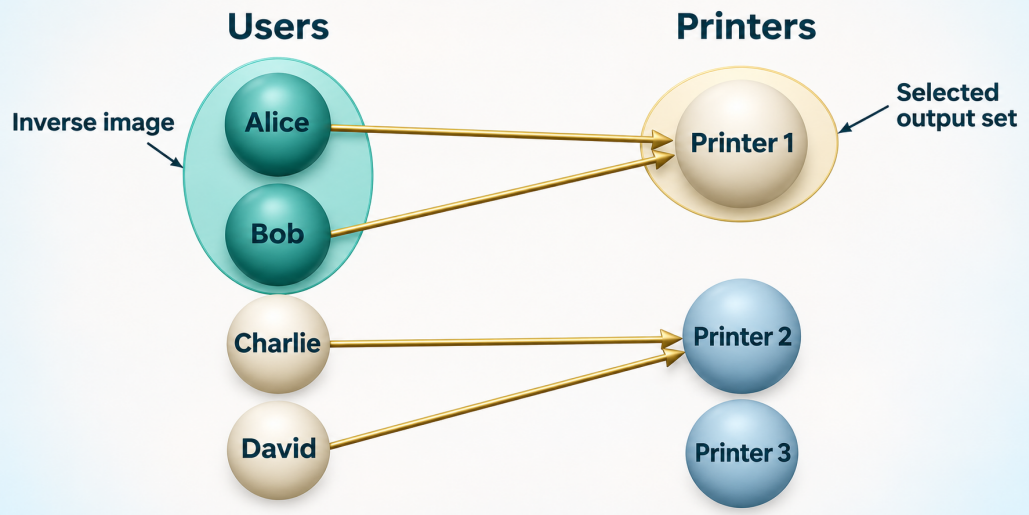
$$h(A) = \{\text{Printer 1}, \text{Printer 2}\}.$$

Repeated outputs are included only once. For an individual input, $h(\text{Alice}) = \text{Printer 1}$, whereas $h(\{\text{Alice}\}) = \{\text{Printer 1}\}$. The first is one printer label; the second is a set.

Example 12.7. For the same function h,

$$h^{-1}(\{\text{Printer 1}\}) = \{\text{Alice}, \text{Bob}\},$$
$$h^{-1}(\{\text{Printer 1}, \text{Printer 2}\}) = U.$$

In the second calculation, a user is included if their output belongs to either of the two listed categories. Every user satisfies that condition. The result is a subset of the domain, not a set of printers.



The inverse image of {Printer 1} under h contains both Alice and Bob.

Example 12.8. No input of h leads to Printer 3, so $h^{-1}(\{\text{Printer 3}\}) = \emptyset$. This is not a failure of the inverse-image definition. The set simply has no matching inputs.

Also, $h^{-1}(\emptyset) = \emptyset$: no function output belongs to the empty set. By contrast, $h^{-1}(P) = U$, since every output of the function lies in its codomain.

Example 12.9. Define $q : \mathbb{Z} \to \mathbb{Z}$ by $q(k) = k^2$. Solving the relevant integer equations gives

$$q^{-1}(\{4\}) = \{-2, 2\}, \qquad q^{-1}(\{0\}) = \{0\}, \qquad q^{-1}(\{-1\}) = \emptyset.$$

Both -2 and 2 square to 4; only 0 squares to 0; no integer square is negative. Even a one-element inverse image is written as a set. The failure of injectivity does not prevent any of these inverse images from being defined.

❄ Definition 12.3 — Inverse function

If $f : X \to Y$ is bijective, its **inverse function** $f^{-1} : Y \to X$ sends each $y \in Y$ to the unique $x \in X$ with $f(x) = y$. It satisfies

$$f^{-1}(f(x)) = x \quad (x \in X), \qquad f(f^{-1}(y)) = y \quad (y \in Y).$$

Surjectivity supplies an input for every y; injectivity makes that input unique.

Example 12.10. For the bijection $g(k) = k + 3$ in Example 12.5, the inverse function is $g^{-1} : \mathbb{Z} \to \mathbb{Z}$, $g^{-1}(y) = y - 3$. In particular,

$$g^{-1}(7) = 4, \qquad g^{-1}(\{7\}) = \{4\}.$$

The first expression evaluates the inverse function at one integer. The second gives the inverse image of a singleton set. Also, $g(g^{-1}(7)) = g(4) = 7$, as required. The inverse is not $1/g(7) = 1/10$.

Example 12.11. For $h : U \to P$, an attempted inverse $P \to U$ would have two possible users for Printer 1 and no possible user for Printer 3. It cannot assign every printer the unique user required to undo h.

The set $h^{-1}(\{\text{Printer 1}\}) = \{\text{Alice}, \text{Bob}\}$ remains meaningful. It reports both inputs rather than pretending there is one inverse-function value.

12.3 Full binary trees

Recall that, in a rooted tree, a leaf has no children and an internal vertex has at least one child. A full binary tree has exactly two children at every internal vertex. Its one-vertex tree has one leaf and no internal vertices.

For the recursive description, distinguish a left child from a right child. This makes our trees **ordered**: swapping different left and right subtrees changes the ordered tree. The counting results below do not depend on that order.

❄ Definition 12.4 — The family $\mathcal{T}$ of finite full binary trees

The family $\mathcal{T}$ is specified by the following rules.

a) **Base object.** The one-vertex rooted tree • belongs to $\mathcal{T}$.

b) **Constructor.** If $S, R \in \mathcal{T}$, then $\mathrm{Node}(S, R) \in \mathcal{T}$. It has a new root whose left subtree is a copy of S and whose right subtree is a separate copy of R.

c) **Exclusion.** Only trees obtained by finitely many applications of these rules belong to $\mathcal{T}$.

Here tree objects record the rooted, left/right shape. Renaming vertices or moving a drawing does not create a different member of $\mathcal{T}$. The copies have disjoint vertex sets, even if S and R have the same shape. The constructor joins their roots to the new root by two new edges. Every non-leaf tree in the family has a unique left subtree and right subtree at its root, so we can read its construction back from the tree.

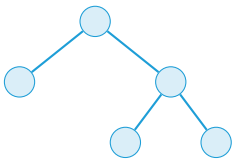
Example 12.12. The base object • has one vertex. Using it twice gives $B = \mathrm{Node}(\bullet, \bullet)$.



$B = \mathrm{Node}(\bullet, \bullet)$

The tree B has two distinct leaves, one in each copy of •, and one internal vertex. Using the same base shape twice does not identify its two vertices with one another.

Example 12.13. Using B from Example 12.12, form $A = \mathrm{Node}(\bullet, B)$. Its left subtree is a single leaf and its right subtree contains a branch.



$A = \mathrm{Node}(\bullet, B)$

There are three leaves and two internal vertices. The leaves have different depths, but every internal vertex has exactly two children, so A is full binary. The ordered tree $\mathrm{Node}(B, \bullet)$ has the opposite left/right arrangement and the same counts.

Example 12.14. A root with exactly one leaf child is not in $\mathcal{T}$. It is not the base object, and the constructor always gives its new root two children. Likewise, a root with three children cannot be obtained.

An infinite branching tree is also excluded. Every construction allowed here uses finitely many steps. This finiteness will let a recursive evaluation reach base objects rather than continue indefinitely.

12.4 Recursive counting functions

A mathematical function may take a tree as its input. The input need not be a number. Our opening question needs a function that returns the number of leaves of a tree in $\mathcal{T}$.

Example 12.15. Counting leaves. Consider this pseudocode, with input a tree $T \in \mathcal{T}$. Here `Node(S,R)` means that S and R are its left and right subtrees.

```
CountLeaves(T):
    if T is a single leaf:
        return 1
    else:
        write T as Node(S,R)
        return CountLeaves(S) + CountLeaves(R)
```

To translate it, name its output $L(T)$ and state the function's type: $L : \mathcal{T} \to \mathbb{N}$, where $\mathbb{N} = \{0, 1, 2, \ldots\}$. The first branch says $L(\bullet) = 1$. The other branch says

$$L(\mathrm{Node}(S, R)) = L(S) + L(R).$$

Every leaf of the combined tree lies in exactly one child subtree; the new root is not a leaf. Thus the two returned counts are added without including an extra leaf for the root. These rules describe the procedure mathematically, with an input domain, an output set and both cases.

Each recursive call uses a proper child subtree, which has fewer vertices than its parent tree. Since the input tree is finite, successive calls eventually reach the one-vertex base case. The base rule and the unique root decomposition therefore determine one value for every allowed input.

❄ Definition 12.5 — Leaf and internal-vertex counting functions

For $T \in \mathcal{T}$, let $L(T)$ count its rooted leaves and $I(T)$ its internal vertices. Their recursive descriptions are

$$L(\bullet) = 1, \quad L(\mathrm{Node}(S, R)) = L(S) + L(R),$$
$$I(\bullet) = 0, \quad I(\mathrm{Node}(S, R)) = 1 + I(S) + I(R),$$

for all $S, R \in \mathcal{T}$. Both functions have domain $\mathcal{T}$ and codomain $\mathbb{N}$.

The new root is an internal vertex, which accounts for the added 1 in the rule for I. All other internal vertices occur in the two disjoint child subtrees.

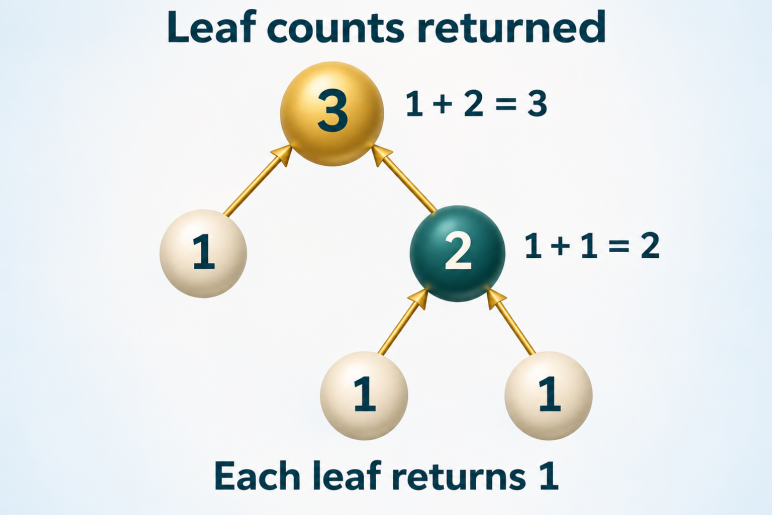
Example 12.16. Recursive evaluation. For $A = \text{Node}(\bullet, B)$ with $B = \text{Node}(\bullet, \bullet)$, the procedure makes one call on each child of the root. The call on B itself makes two leaf calls. Its numerical evaluation is

$$\begin{aligned} L(A) &= L(\bullet) + L(B) \\ &= 1 + \big(L(\bullet) + L(\bullet)\big) \\ &= 1 + (1 + 1) = 3. \end{aligned}$$

The two leaf calls inside B each return 1; the call on B returns 2; the outer call returns $1 + 2 = 3$. The matching internal count is

$$I(A) = 1 + I(\bullet) + I(B) = 1 + 0 + (1 + 0 + 0) = 2.$$

The recursive values agree with the drawing.



Each node displays its subtree's leaf count: the right subtree returns 2 and the root returns $1 + 2 = 3$.

Python (non-examinable)

To run the procedure, represent a single leaf by `None` and $\text{Node}(S, R)$ by the ordered pair `(S, R)`, called a *tuple* in Python. Here `None` represents one vertex, not an empty tree. The line beginning `def` defines the function; `return` gives its output. The condition `tree is None` recognises the base leaf.

```
def count_leaves(tree):
    if tree is None:
        return 1
    left, right = tree
    return count_leaves(left) + count_leaves(right)

B = (None, None)
A = (None, B)
print(count_leaves(B), count_leaves(A))
```

Output.

```
2 3
```

Python (non-examinable) — continued

The assignment `left, right = tree` reads the two child subtrees. Each recursive call applies the same function to a child, and its returned value supplies one of the two counts being added. These are exactly the trees B and A traced above. The code assumes an input built from the stated leaf and two-child rules; it does not check arbitrary Python data. The example computes two values, while the mathematical definitions and proofs describe the whole family.

Example 12.17. Repeated subtrees. For $D = \mathrm{Node}(B, B)$,

$$L(D) = 2 + 2 = 4, \qquad I(D) = 1 + 1 + 1 = 3.$$

Each occurrence of B supplies a separate subtree. Both must contribute to the count. The whole tree has $4 + 3 = 7$ vertices. The formula $L(D) = L(B)$ would omit all leaves in the second copy.

Example 12.18. Suppose someone writes only $F(\mathrm{Node}(S, R)) = F(S) + F(R)$ for $F : \mathcal{T} \to \mathbb{N}$. There is no rule yet for $F(\bullet)$. Setting that base value to 0 gives zero at every tree; setting it to 1 gives the leaf count. Both choices obey the displayed constructor rule.

The function description needs its base case, just as a sequence recurrence in Chapter 11 needs its initial information. The tree domain also excludes the one-child inputs for which the constructor rule supplies no case.

Example 12.19. Define $W : \mathcal{T} \to \mathbb{N}$ by $W(\bullet) = 1$ and $W(\mathrm{Node}(S, R)) = 1 + W(S) + W(R)$. Then $W(B) = 1 + 1 + 1 = 3$, although B has only two leaves. The extra 1 counts the new root. In fact, this rule counts all vertices rather than leaves. The recurrence can be complete and evaluable while describing a quantity different from the intended one.

12.5 Structural induction

Ordinary induction follows the construction of integers from a starting value and a next step. Here an object is built from a leaf or from two already constructed child trees. Our proof can follow those same construction rules.

❄ Definition 12.6 — Structural induction on $\mathcal{T}$

To prove a property $Q(T)$ for every $T \in \mathcal{T}$:

a) **Base case.** Prove $Q(\bullet)$.

b) **Constructor case.** Let $S, R \in \mathcal{T}$ be arbitrary. Assume $Q(S)$ and $Q(R)$, and use them to prove $Q(\mathrm{Node}(S, R))$.

These checks establish Q for every tree generated by the defining rules.

The principle works because every allowed tree has a finite construction. The property starts at each base leaf and is preserved whenever two constructed trees are joined. Following the construction to its root therefore establishes the property of the finished tree. The exclusion clause ensures that no other kinds of object need a separate case.

The two subtree hypotheses may concern trees of different shapes and sizes. We assume the property for the child trees, then prove it for their newly constructed parent. Assuming it for that parent instead would assume the target we are trying to establish.

Example 12.20. Leaves and internal vertices. Prove that for every $T \in \mathcal{T}$,

$$L(T) = I(T) + 1.$$

Proof. We use structural induction.

Base case. For the single-leaf tree, $L(\bullet) = 1 = 0 + 1 = I(\bullet) + 1$.

Constructor case. Let $S, R \in \mathcal{T}$ and assume both

$$L(S) = I(S) + 1, \qquad L(R) = I(R) + 1.$$

The target is $L(\mathrm{Node}(S, R)) = I(\mathrm{Node}(S, R)) + 1$. Using the recursive definitions and the two hypotheses,

$$\begin{aligned} L(\mathrm{Node}(S, R)) &= L(S) + L(R) \\ &= (I(S) + 1) + (I(R) + 1) \\ &= I(S) + I(R) + 2 \\ &= \big(1 + I(S) + I(R)\big) + 1 \\ &= I(\mathrm{Node}(S, R)) + 1. \end{aligned}$$

The base and constructor cases cover the defining rules of $\mathcal{T}$, so the claim holds for every $T \in \mathcal{T}$. □

Each child tree supplies its own extra leaf in the induction hypotheses. The constructor adds one internal root, leaving one extra leaf for the whole tree.

Example 12.21. If a tree in $\mathcal{T}$ has 18 internal vertices, it has 19 leaves and $18 + 19 = 37$ vertices in total. Conversely, 24 leaves require 23 internal vertices. We can make these deductions without knowing the particular shape, because the proof covers every tree in the specified family.

In the opening task model, internal vertices record splits and leaves record finished end tasks. There is therefore one more finished end task than split, provided the execution is finite and each split creates exactly two child tasks.

Example 12.22. A proposed proof handles the base case and then checks only $\mathrm{Node}(S, S)$. That does not establish the claim for $\mathrm{Node}(S, R)$ when S and R differ. For example, $A = \mathrm{Node}(\bullet, B)$ has different child shapes.

The constructor permits any pair of trees from $\mathcal{T}$. Its proof case must therefore allow arbitrary S, R and use the property of each. The two hypotheses need not be numerically identical.

Example 12.23. Consider the rooted tree with a root and exactly one leaf child. It has one leaf and one internal vertex, so $L = I + 1$ would read $1 = 2$. This does not refute Example 12.20: the tree is not full binary and is outside $\mathcal{T}$.

Its leaf count is still an ordinary count, but our two-child constructor and recursive specification do not describe this input. Changing the family requires changing the rules and checking which conclusions remain valid.

12.6 Exercises

12.1. Let $X = \{0, 1, 2\}$ and $Y = \{a, b, c\}$. Three functions $X \to Y$ are defined by the complete table.

Input	0	1	2
f	a	b	c
g	a	a	b
h	c	b	a

For each function, give the range and decide whether it is injective, surjective and bijective. Give witnesses for any failed property.

12.2. Let $i : \{1, 2, 3\} \to \{10, 20, 30, 40\}$ be $i(k) = 10k$. Let $j : \{1, 2, 3\} \to \{10, 20, 30\}$ use the same rule. Classify each function as injective, surjective, both or neither. Explain exactly what changed.

12.3. Use complete tables to give:

a) an injective but non-surjective function $\{1, 2\} \to \{a, b, c\}$;

b) a surjective but non-injective function $\{1, 2, 3\} \to \{a, b\}$;

c) a function $\{1, 2, 3\} \to \{a, b, c\}$ with neither property.

12.4. For $f : \mathbb{Z} \to \mathbb{Z}$, $f(k) = k - 4$, prove injectivity and surjectivity separately. Give the inverse function, including its domain and codomain, and verify its value at 7.

12.5. Let $f : \mathbb{Z} \to \mathbb{Z}$ be $f(k) = 2k$. Prove that f is injective, but give a codomain element showing it is not surjective. David proposes the inverse rule $y \mapsto y/2$ on all of $\mathbb{Z}$. Explain why it does not define an inverse function $\mathbb{Z} \to \mathbb{Z}$.

12.6. Let $U = \{\text{Alice}, \text{Bob}, \text{Charlie}, \text{David}\}$ and $P = \{\text{Printer 1}, \text{Printer 2}, \text{Printer 3}\}$. Define $p : U \to P$ by

User	Alice	Bob	Charlie	David
p	Printer 2	Printer 1	Printer 2	Printer 1

Find each set:

a) $p(\{\text{Alice}, \text{Charlie}\})$;

b) $p(\{\text{Bob}, \text{Charlie}\})$;

c) $p^{-1}(\{\text{Printer 2}\})$;

d) $p^{-1}(\{\text{Printer 3}\})$;

e) $p^{-1}(\{\text{Printer 1}, \text{Printer 3}\})$.

12.7. For the function p in Exercise 12.6, find $p(\varnothing)$, $p^{-1}(\varnothing)$, $p^{-1}(P)$ and $p(U)$. For each expression, say whether its definition selects inputs or outputs.

12.8. Let $q : \mathbb{Z} \to \mathbb{Z}$ be $q(k) = k^2$. Find the following:

a) $q(\{-3, 0, 3\})$.

b) $q^{-1}(\{9\})$.

c) $q^{-1}(\{0\})$.

d) $q^{-1}(\{-4, 4\})$.

Does q have an inverse function $\mathbb{Z} \to \mathbb{Z}$? Justify your answer using the stated domain and codomain.

12.9. Let $b : \mathbb{Z} \to \mathbb{Z}$, $b(k) = k - 4$. Find $b(7)$, $b(\{7\})$, $b^{-1}(7)$ and $b^{-1}(\{7\})$. Explain the role of the braces in each expression and why $b^{-1}(7)$ is not $1/b(7)$.

12.10. For p in Exercise 12.6, Faith writes $p^{-1}(\{\text{Printer 2}\}) = \text{Alice}$. Identify both the omitted input and the type of object that the answer must be. Give the corrected set and explain why its existence does not give p an inverse function.

12.11. Let $B = \text{Node}(\bullet, \bullet)$. Draw each ordered tree with its root at the top:

a) $\bullet$;

b) B;

c) $\text{Node}(B, \bullet)$;

d) $\text{Node}(\bullet, B)$;

e) $\text{Node}(B, B)$.

For each drawing, identify its leaves and internal vertices.

12.12. Which descriptions belong to $\mathcal{T}$? Justify each answer from the base or constructor rules.

a) A single rooted vertex.

b) A root with exactly one leaf child.

c) A root with two leaf children.

d) A root with three leaf children.

e) A finite full binary tree whose leaves have different depths.

12.13. For $\bullet$, give the root's graph degree, number of children, leaf count and internal count. Why would treating a leaf as only a degree-one vertex give the wrong base value for L on this family?

12.14. Let $B = \text{Node}(\bullet, \bullet)$, $A = \text{Node}(B, \bullet)$ and $C = \text{Node}(A, B)$. Find L and I for B, A, C, showing the substitution into each constructor rule. Check the resulting values against $L = I + 1$.

12.15. Let $B = \text{Node}(\bullet, \bullet)$ and $T = \text{Node}(B, B)$. Trace `CountLeaves(T)` using Example 12.15. Show the two calls on copies of B, the values returned by their leaf calls, and the final output. Why must the second occurrence of B contribute even though it has the same shape as the first?

12.16. The input of this procedure is $T \in \mathcal{T}$.

```
CountInternal(T):
    if T is a single leaf:
        return 0
    else:
        write T as Node(S,R)
        return 1 + CountInternal(S) + CountInternal(R)
```

Specify its function domain and codomain, base value and constructor rule. Trace it on the tree

$$\text{Node}(\bullet, \text{Node}(\bullet, \bullet)).$$

Explain what the added 1 counts.

12.17. Suppose $F : \mathcal{T} \to \mathbb{N}$ satisfies $F(\text{Node}(S, R)) = F(S) + F(R)$, but no base value is supplied. Choose two different base values and calculate $F(B)$ for $B = \text{Node}(\bullet, \bullet)$. Use the two results to explain the missing information.

12.18. A proposed leaf-counting procedure returns 1 at a leaf, but at $\text{Node}(S, R)$ it returns

$$1 + \texttt{CountLeaves(S)} + \texttt{CountLeaves(R)}.$$

Assume both recursive calls use this same proposed rule. Translate it into a recursive function W, evaluate $W(B)$, and compare with the actual leaf count. Explain which quantity the rule counts.

12.19. In the procedure from Example 12.15, what changes when a call passes to a child subtree? Use the finiteness of the input tree to justify termination. Why would replacing the recursive call on a child by another call on the whole unchanged tree fail to provide this justification?

12.20. For a structural-induction proof of $L(T) = I(T) + 1$ on $\mathcal{T}$, state:

a) the property of the base object;

b) the two subtree hypotheses;

c) the target for $\text{Node}(S, R)$;

d) the recursive identity that introduces the new internal root.

12.21. In an attempted independent proof of $L(T) = I(T) + 1$ by structural induction, a student assumes only $L(S) = I(S) + 1$, then writes

$$L(\text{Node}(S, R)) = L(S) + L(R) = (I(S) + 1) + (I(R) + 1).$$

Which replacement is unsupported by the stated hypothesis? Repair the hypotheses and finish the argument. Explain why restricting to $R = S$ would not repair the proof for all trees in $\mathcal{T}$.

12.22. Judy checks $L = I + 1$ for $\bullet$, $\text{Node}(\bullet, \bullet)$ and $\text{Node}(B, B)$, where $B = \text{Node}(\bullet, \bullet)$. Are these checks a structural-induction proof? Identify the general implication still needed.

12.23. For trees in $\mathcal{T}$, find the requested counts.

a) A tree has 12 internal vertices. Find its leaves and total vertices.

b) A tree has 20 leaves. Find its internal vertices and total vertices.

c) Can a tree have 8 leaves and 8 internal vertices?

12.24. Define $V : \mathcal{T} \to \mathbb{N}$ by $V(\bullet) = 1$ and $V(\text{Node}(S, R)) = 1 + V(S) + V(R)$. Use structural induction to prove $V(T) = 2I(T) + 1$ for every $T \in \mathcal{T}$. State both subtree hypotheses and show how the new-root contribution enters the algebra.

12.25. Give a finite rooted tree outside $\mathcal{T}$ for which the ordinary numbers of leaves and internal vertices do not satisfy $L = I + 1$. Explain why it does not contradict the proved theorem. Identify which recursive construction rule would need to change to include your example.

12.26. Consider the counting function $L : \mathcal{T} \to \mathbb{N}$.

a) Construct two distinct ordered trees with the same leaf count. What does this say about injectivity?

b) Show that 0 is not a leaf count. What does this say about surjectivity onto the stated codomain?

c) Describe $L^{-1}(\{2\})$ by identifying every tree in that inverse image.

Use $L = I + 1$ when it helps, and keep inverse images as sets of tree objects.