CHAPTER 7 · FUNCTIONS

Properties of functions

One-to-one · onto · bijections · inverses · composition · the pigeonhole principle

Two questions about the arrows

In 7.1 a function was a rule that gives each input exactly one output. The constraint was only on the *left* (domain) side. Now we ask two questions about the *right* (co-domain) side:

Can two arrows hit the same target?

If never → the function is **one-to-one** (injective). متباينة

Does every target get an arrow?

If yes → the function is **onto** (surjective). شاملة

Both at once → **bijection (تقابل)**, which lets us "run the function backward" to get an **inverse**. Then 7.3 chains functions together (composition), and 9.4 turns "not one-to-one" into a powerful counting tool.

7.2 One-to-one and onto, inverse functions

1. **One-to-one functions**
2. Onto functions
3. Bijections
4. Inverse functions

One-to-one functions

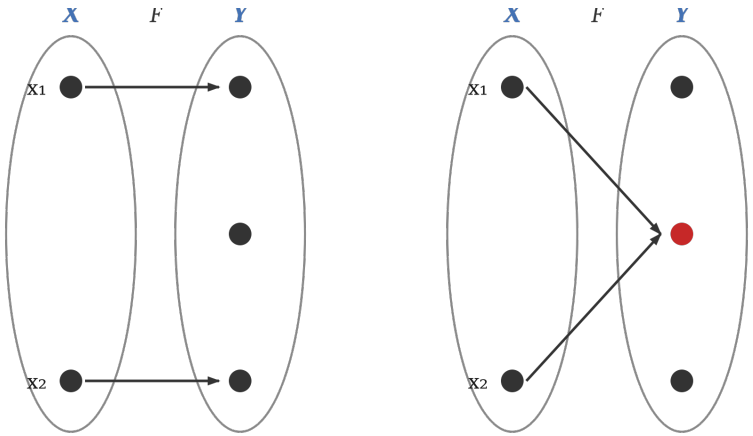
A function F : X → Y is **one-to-one** (injective) if distinct inputs always give distinct outputs.

$$\textbf{\textit{F is one-to-one}} \Leftrightarrow$$

$$\forall\, x_1, x_2 \in X,\ \textit{if } F(x_1) = F(x_2) \textit{ then } x_1 = x_2.$$

(equivalently:

$$x1 \neq x2 \ \Rightarrow\ F(x1) \neq F(x2)\ )$$



separates (one-to-one)

collapses (not one-to-one)

Picture: a one-to-one function **separates points** — no target receives two arrows. A function that is not one-to-one **collapses** two points onto one.

Proving — and disproving — one-to-one

$$F \text{ is NOT one-to-one} \iff \exists\, x_1, x_2 \in X \text{ with } F(x_1) = F(x_2) \text{ and } x_1 \neq x_2.$$

Two opposite jobs, two opposite methods:

To PROVE one-to-one

(direct proof)

Suppose $F(x_1) = F(x_2)$ for arbitrary x_1, x_2. Work algebraically to conclude $x_1 = x_2$.

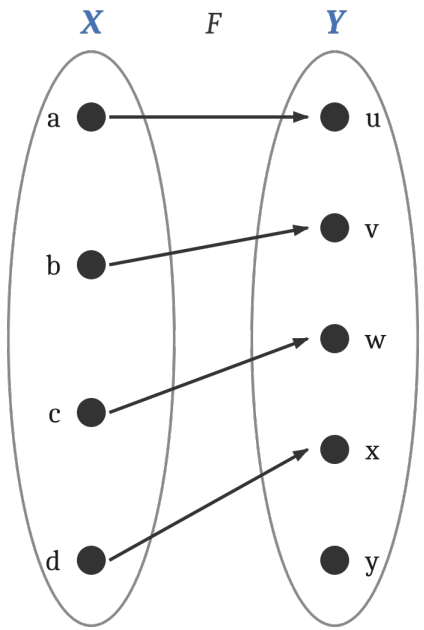
To DISPROVE one-to-one

(counterexample)

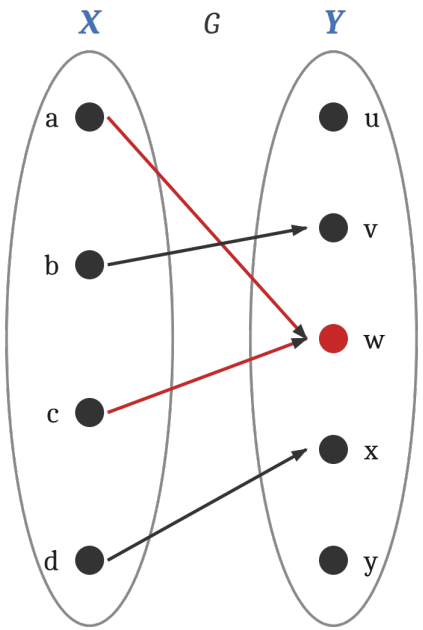
Exhibit two specific unequal inputs $x_1 \neq x_2$ with the same output $F(x_1) = F(x_2)$. One pair is enough.

On finite sets you can just inspect the arrows; on infinite sets you must argue. We do one of each next.

Example 7.2.1 — one-to-one on finite sets



F: one-to-one ✓



G: not one-to-one ✗

F sends the four inputs to four different outputs, so no two collide — one-to-one.

G sends both a and c to w: G(a) = G(c) = w but a ≠ c. That single collision is enough — not one-to-one.

Also (no diagram needed): X = {1, 2, 3}, Y = {a, b, c, d}.
H(1)=c, H(2)=a, H(3)=d — all different → one-to-one.
K(1)=d, K(2)=b, K(3)=d — K(1)=K(3) but 1≠3 → not.

Example 7.2.2 — one-to-one on infinite sets

$f : \mathbf{R} \to \mathbf{R}$, $f(x) = 4x - 1$

Claim: one-to-one ✓

Proof. Suppose $f(x_1) = f(x_2)$. Then

$4x_1 - 1 = 4x_2 - 1.$

Add 1: $4x_1 = 4x_2$.

Divide by 4: $x_1 = x_2$. ∎

$g : \mathbf{Z} \to \mathbf{Z}$, $g(n) = n^2$

Claim: not one-to-one ✗

Counterexample.

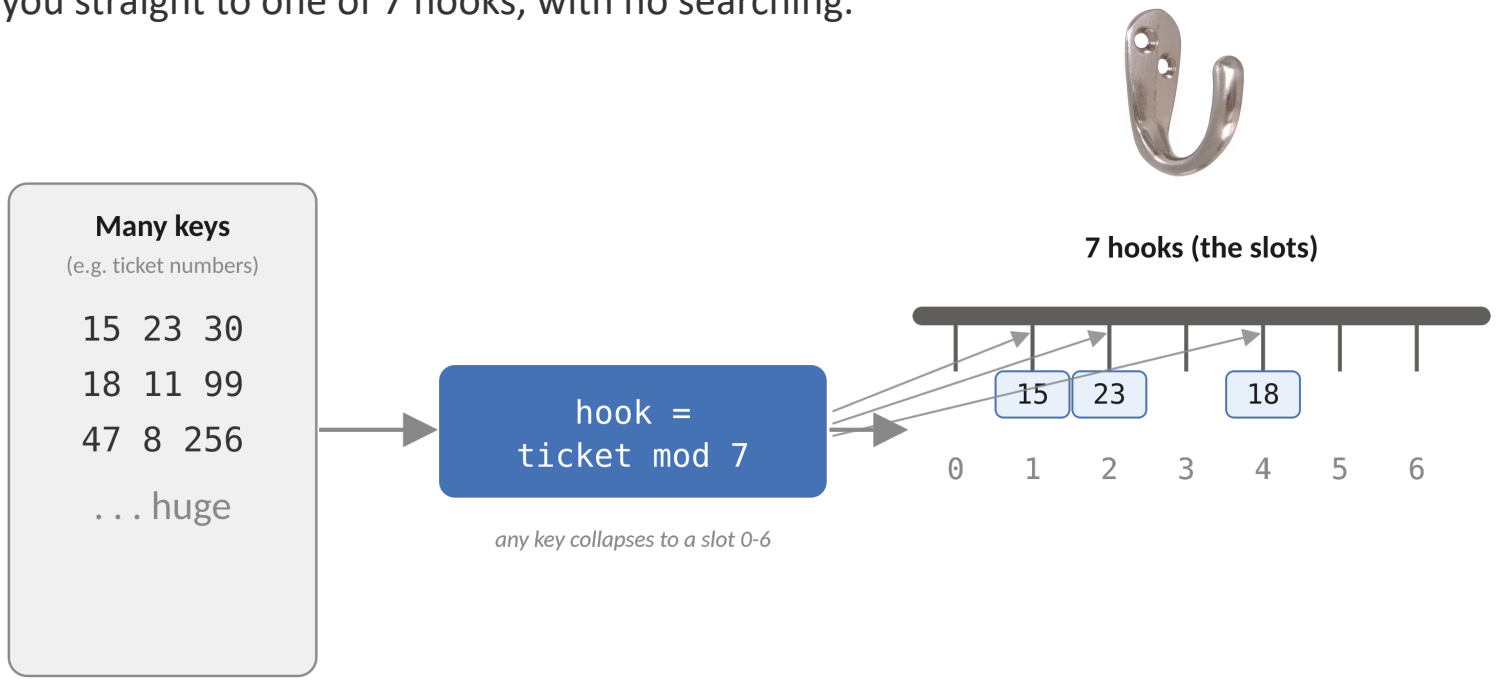
$g(2) = 4$ and $g(-2) = 4$,
so $g(2) = g(-2)$ but $2 \neq -2$.

Two different inputs, one output — the definition of one-to-one fails, so g is not one-to-one.

Same shape of rule, different verdict — the domain matters. Squaring fails because it forgets the sign.

Why hash functions?

A **hash function** turns any key into a small, fixed slot number instantly — like a cloakroom attendant who reads your ticket and sends you straight to one of 7 hooks, with no searching.



Where it's used

Hash tables & maps — instant key → value (a Python dict, a JS map).

Databases — jump straight to a record instead of scanning.

Caches — answer "seen this before?" in one step.

Integrity & dedup — file checksums, Git commit IDs.

Passwords — store the hash, never the secret itself.

Why it's fast: instead of searching every record, you compute one slot and go there — near constant-time lookup. The trade-off: many keys squeezed into few slots, so a hash can never be one-to-one (collisions — next).

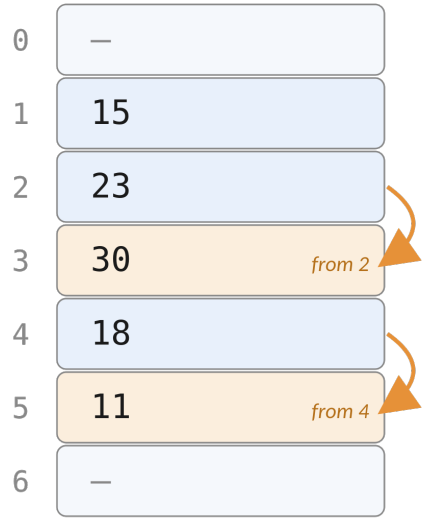
Hashing in action (Example 7.2.3)

Insert these keys in order:

```
15 mod 7 = 1
23 mod 7 = 2
30 mod 7 = 2        collision
18 mod 7 = 4
11 mod 7 = 4        collision
```

A collision means the home slot is taken, so we walk down to the next free slot.

Table (slots 0-6)



The rule

Hash(n) = n mod 7 — key n → slot 0–6.

Collisions

If the home slot is taken, walk down to the next free slot (wrapping at the bottom). Here 30 lands in slot 3 and 11 in slot 5. This is *linear probing*.

Finding a record

Recompute Hash(n) and search down from the home slot until you reach it — or an empty slot, which means it was never stored.

Collisions are unavoidable: with far more keys than slots, the hash cannot be one-to-one — two keys must eventually share a slot. That is exactly the pigeonhole principle (§9.4). Good hashing just spreads keys evenly and resolves the rare clash cheaply.

7.2 One-to-one and onto, inverse functions

1. One-to-one functions
2. **Onto functions**
3. Bijections
4. Inverse functions

Onto functions

A function F : X → Y is **onto** (surjective) if every element of Y is the image of at least one element of X. Then range = co-domain.

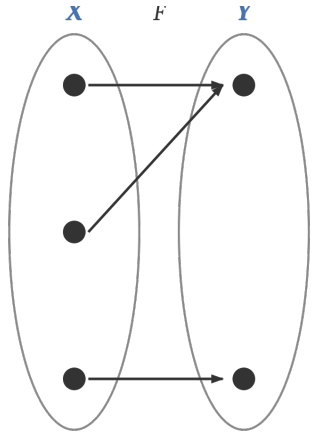
$$\textbf{\textit{F is onto}} \Leftrightarrow$$

$$\forall\, y \in Y,\ \exists\, x \in X \ \textit{such that}\ F(x) = y.$$

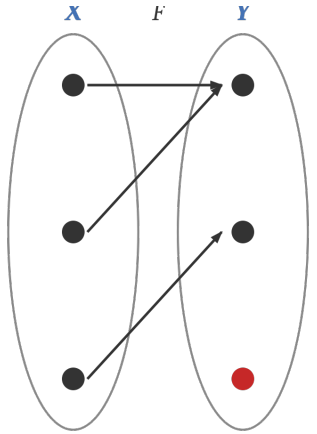
$$\textbf{\textit{F is not onto}} \Leftrightarrow$$

$$\exists\, y \in Y \ \textit{such that}\ \forall\, x \in X,\ F(x) \neq y.$$

Picture: onto means every target dot has at least one arrow arriving. Not onto means some target dot is missed entirely (the red dot).



onto



not onto

Proving — and disproving — onto

Two opposite jobs, two opposite methods:

To PROVE onto

(generalise from the generic particular)

Take an arbitrary $y \in Y$. Produce a specific $x \in X$ (built from y) and verify $F(x) = y$. Two duties: x lies in X, and F really sends it to y.

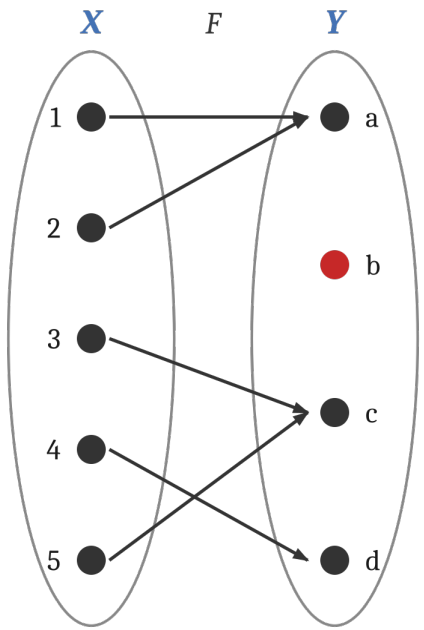
To DISPROVE onto

(exhibit a missed target)

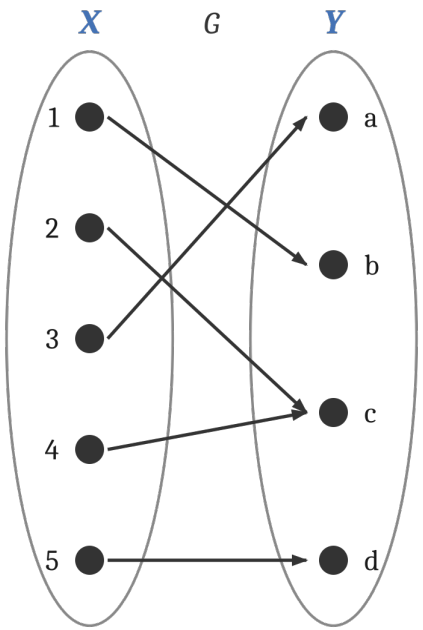
Find one specific $y \in Y$ for which no $x \in X$ gives $F(x) = y$. A single un-hit element is enough.

Scratch work tip: *to find the x, solve $F(x) = y$ for x. If the solution always lands inside X, F is onto; if it can fall outside X, that points you to a counterexample.*

Example 7.2.4 — onto on finite sets



F: not onto ✗



G: onto ✓

F misses **b** — no input maps to it, so F is not onto. One un-hit target is enough.

G hits every target: $a = G(3)$, $b = G(1)$, $c = G(2) = G(4)$, $d = G(5)$. Every element of Y is an image, so G is onto.

On finite sets: onto just means every right-hand dot has an arrow. Note G is onto but *not* one-to-one (c is hit twice) — the two properties are independent.

Example 7.2.5 — onto on infinite sets

$f : \mathbb{R} \to \mathbb{R},\ f(x) = 4x - 1$

Claim: onto ✓

Let $y \in \mathbb{R}$. Solve $4x - 1 = y$:

$x = (y + 1) / 4$.

This x is a real number, and

$f(x) = 4 \cdot (y+1)/4 - 1 = y$.

Every real y is reached, so f is onto.

$h : \mathbb{Z} \to \mathbb{Z},\ h(n) = 4n - 1$

Claim: not onto ✗

Same algebra gives $n = (m + 1) / 4$,
which need not be an integer.

Counterexample: m = 0.

$4n - 1 = 0 \Rightarrow n = 1/4 \notin \mathbb{Z}$.

So 0 is in the co-domain but has no integer preimage — h is not onto.

Same rule, different domain, opposite verdict. *Over R the solved-for x always exists; over Z it can fall between integers. Domain and co-domain decide everything.*

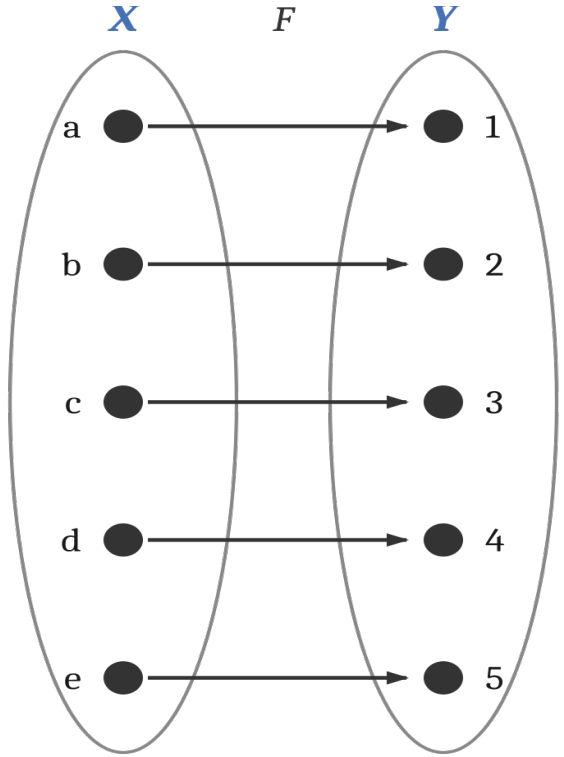
7.2 One-to-one and onto, inverse functions

1. One-to-one functions
2. Onto functions
3. **Bijections**
4. Inverse functions

Bijections (one-to-one correspondences)

A **bijection** (one-to-one correspondence) from X to Y is a function F : X → Y that is **both one-to-one and onto.**

Such an F pairs the two sets perfectly: each x ∈ X matches exactly one y ∈ Y, and each y ∈ Y matches exactly one x ∈ X.



Why bijections are important

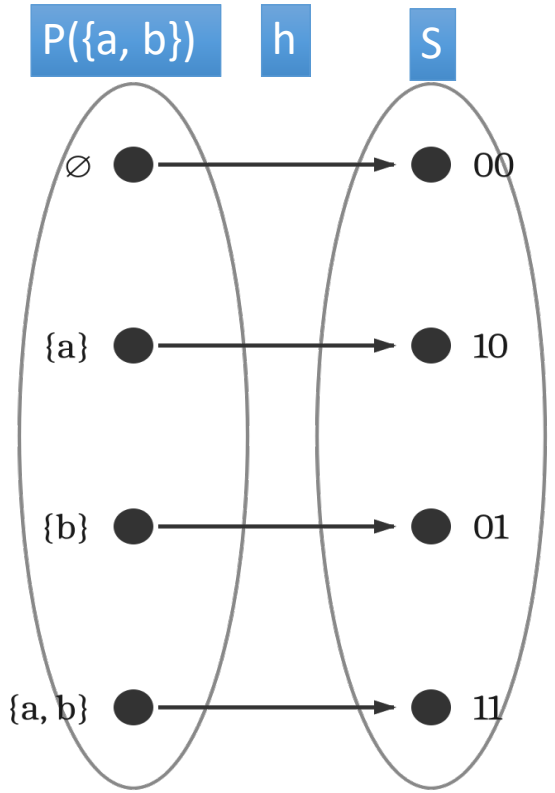
1. They can be reversed — a bijection has an inverse function (next).
2. They count: a bijection X ↔ Y proves |X| = |Y|. The pairing a–e ↔ 1–5 shows X has 5 elements.

Example 7.2.8 — a bijection: subsets ↔ bit strings

Let h : P({a, b}) → S, where S is all length-2 strings of 0's and 1's. Rule: position 1 is 1 iff a ∈ A; position 2 is 1 iff b ∈ A.

Subset A	a ?	b ?	h(A)
∅	no	no	00
{a}	yes	no	10
{b}	no	yes	01
{a, b}	yes	yes	11

h is a bijection: onto (every string is hit) and one-to-one (no string is hit twice). The four subsets pair exactly with the four strings.



7.2 One-to-one and onto, inverse functions

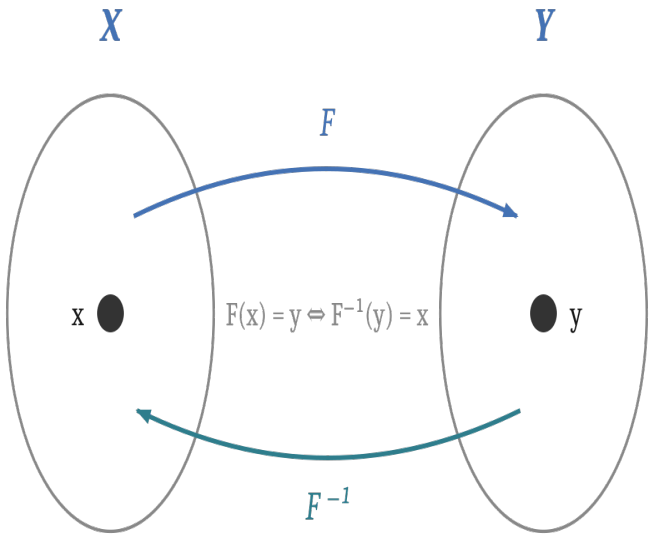
1. One-to-one functions
2. Onto functions
3. Bijections
4. **Inverse functions**

Inverse functions (Theorem 7.2.2)

If $F : X \to Y$ is a **bijection**, then it has an inverse function $F^{-1} : Y \to X$ that undoes it.

$$F^{-1}(y) = \text{the unique } x \in X \text{ with } F(x) = y$$

$$F^{-1}(y) = x \iff y = F(x)$$



Why a bijection is required: onto guarantees every y has a source (so F^{-1} is defined everywhere); one-to-one guarantees that source is unique (so F^{-1} is single-valued).

Back to 7.1: $\log_b$ is exactly the inverse of the exponential function b^x — each undoes the other.

Finding inverses (Examples 7.2.11, 7.2.13)

From an arrow diagram (7.2.11)

Reverse the arrows of h (Example 7.2.8):

$h^{-1}(00) = \varnothing$

$h^{-1}(10) = \{a\}$

$h^{-1}(01) = \{b\}$

$h^{-1}(11) = \{a, b\}$

From a formula (7.2.13)

$f : R \to R,\ f(x) = 4x - 1$ (a bijection).

Solve $y = 4x - 1$ for x:

$x = (y + 1) / 4.$

So $\mathbf{f^{-1}(y) = (y + 1) / 4.}$

Check: $f^{-1}(f(x)) = (4x - 1 + 1)/4 = x.$ ✓

Two ways, one idea: *the inverse just reads each arrow backward. For a formula, that means solving y = F(x) for x in terms of y.*

Logarithmic functions and the laws of exponents

The exponential function $\exp_b(x) = b^x$ and the logarithm $\log_b$ are both one-to-one and onto, so each is the inverse of the other:

$$\log_b x = y \iff b^y = x$$

This is the inverse relationship from the previous part, now named: $\log_b = (\exp_b)^{-1}$.

Why this is important: logarithms turn exponential questions into linear ones — the engine behind measuring information, complexity, and scale in computing.

Laws of exponents

$$b^u \, b^v = b^{u+v}$$

$$(b^u)^v = b^{uv}$$

$$b^u / b^v = b^{u-v}$$

$$(bc)^u = b^u \, c^u$$

Theorem 7.2.1 — properties of logarithms

For positive real numbers b, c, x with $b \neq 1$ and $c \neq 1$:

a. $\log_b(xy) = \log_b x + \log_b y$ — *product → sum*

b. $\log_b(x / y) = \log_b x - \log_b y$ — *quotient → difference*

c. $\log_b(x^a) = a \cdot \log_b x$ — *power → coefficient*

d. $\log_c x = (\log_b x) / (\log_b c)$ — *change of base*

All four follow from the laws of exponents plus the one-to-oneness of the exponential function — proved next for (d).

Product rule for logarithms (Theorem 7.2.1a)

$$\log_b(xy) = \log_b x + \log_b y \qquad \textit{(for } b \neq 1 \textit{ and } x, y > 0\textit{)}$$

Proof

Use the definition $\log_b z = w \iff b^w = z$.

1. Let $u = \log_b x$, $v = \log_b y$.
2. Then $x = b^u$, $y = b^v$. *(definition)*
3. $xy = b^u \cdot b^v = b^{u+v}$. *(exponent law)*
4. So $\log_b(xy) = u + v$. *(definition, backward)*
5. $= \log_b x + \log_b y$. *(substitute)*

The idea

Multiplication becomes addition. The product rule is just the exponent law $b^u \cdot b^v = b^{u+v}$ read through the inverse: logs report exponents, and multiplying powers adds them.

Check (b = 10):

$\log(100 \cdot 1000) = \log 100000 = 5$,
$\log 100 + \log 1000 = 2 + 3 = 5$. ✓

Same recipe as change of base: name the exponents, apply one law of exponents, then read the definition back. Properties (b) and (c) follow the same way — from the quotient and power laws of exponents.

Change of base — proof and use (Examples 7.2.6, 7.2.7)

$\log_c x = (\log_b x) / (\log_b c)$

Proving (d) with one-to-oneness of exp

Let $u = \log_b c$, $v = \log_c x$, $w = \log_b x$.

Then $c = b^u$, $x = c^v$, $x = b^w$.

$x = c^v = (b^u)^v = b^{uv}$, and also $x = b^w$.
So $b^{uv} = b^w$, and exp is one-to-one, so $uv = w$.

$(\log_b c)(\log_c x) = \log_b x \;\Rightarrow\; \log_c x = (\log_b x)/(\log_b c)$.

Using it: compute $\log_2 5$

Most calculators have ln (base e) but no base-2 key. By (d):

$\log_2 5 = (\ln 5)/(\ln 2)$

$\approx 1.6094 / 0.6931$

$\approx \mathbf{2.3219.}$

Any convenient base b works — the ratio is the same.

Takeaway: the one-to-oneness of a function is not just a definition to check — it is a proof tool, here turning an exponent equality into a logarithm identity.

Recommended exercises — Section 7.2

Epp, 4th edition — Section 7.2 (p. 413)

1, 8, 12, 13, 17, 18, 23, 25, 27, 33, 35, 43

What they drill: testing one-to-one and onto from diagrams and formulas (1–18); bijections and counting (23–27); inverse functions; and logarithm properties / change of base (33–35).

Next: 7.3 — composition of functions, then 9.4 — the pigeonhole principle.

7.3 Composition of functions

1. **Definition of composition**
2. Worked examples
3. Identity and inverse
4. Preserving one-to-one and onto

Composition of functions

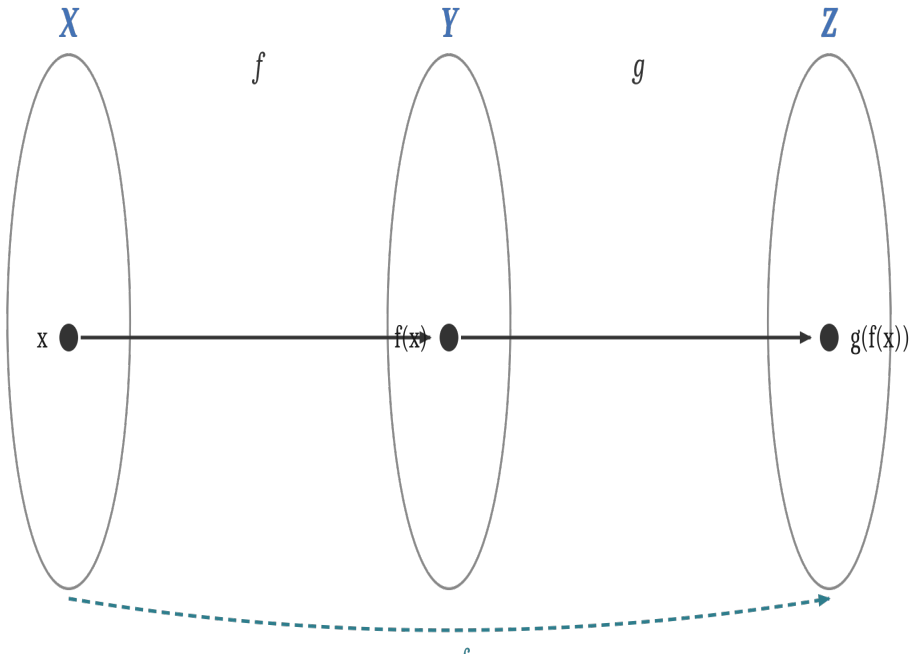
Given f : X → Y and g : Y → Z (with range of f inside the domain of g), the **composition** g ∘ f : X → Z is

$$(g \circ f)(x) = g(f(x)) \quad \text{for all } x \in X.$$

Read “g circle f ”. f acts first, then g — apply right-to-left.

Caution. g ∘ f is the **name of a function**; g(f(x)) is its **value** at x. The composition exists only if range(f) ⊆ domain(g).

Machine picture: two machines in series — x enters f, the output f(x) feeds straight into g, and g(f(x)) comes out.



Example 7.3.1 — order matters

Let $f, g : \mathbb{Z} \to \mathbb{Z}$ be $f(n) = n + 1$ (successor) and $g(n) = n^2$ (squaring).

$$(g \circ f)(n) = g(f(n)) = g(n+1) = (n+1)^2$$

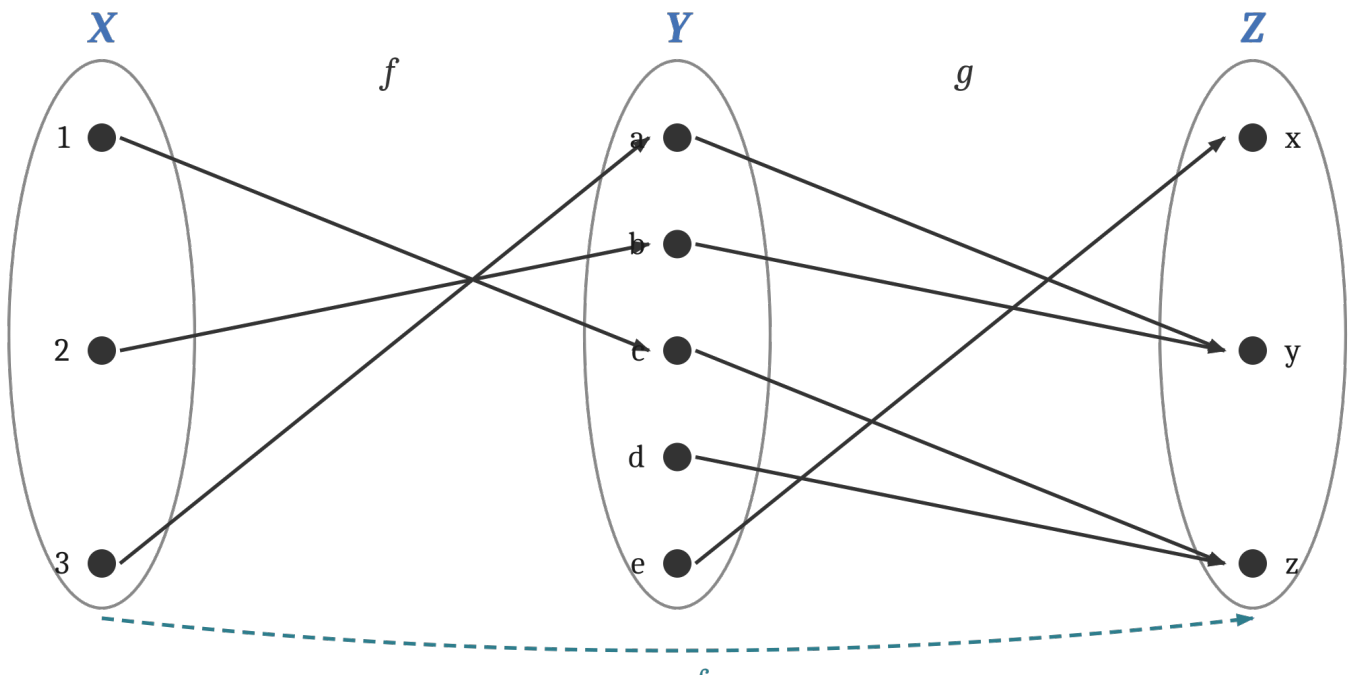
$$(f \circ g)(n) = f(g(n)) = f(n^2) = n^2 + 1$$

Test at n = 1: $(g \circ f)(1) = (1+1)^2 = 4$, but $(f \circ g)(1) = 1^2 + 1 = 2$. So $g \circ f \neq f \circ g$.

Composition is not commutative. In general $F \circ G$ need not equal $G \circ F$. Order is part of the operation.

Example 7.3.2 — composition on finite sets

Trace each arrow all the way across, X → Y → Z, to read off g ∘ f.



$(g \circ f)(1) = g(c) = z$

$(g \circ f)(2) = g(b) = y$

$(g \circ f)(3) = g(a) = y$

Range of $g \circ f = \{y, z\}$.

Note: 2 and 3 pass through different middle elements but reach the same output y — composition can collapse what the pieces kept apart. The intermediate set Y just needs to receive f and feed g.

Composition with identity and inverse

Theorem 7.3.1 — identity

$f \circ I_X = f \quad \text{and} \quad I_Y \circ f = f.$

Composing with the do-nothing function changes nothing — the identity is neutral for composition.

Theorem 7.3.2 — inverse

If f is a bijection with inverse f^{-1}:

$f^{-1} \circ f = I_X \quad \text{and} \quad f \circ f^{-1} = I_Y.$

A bijection followed by its inverse returns every element to itself.

One-line proofs from the definition:

$(f \circ I_X)(x) = f(I_X(x)) = f(x). \qquad (f^{-1} \circ f)(x) = f^{-1}(f(x)) = x.$

Together: the identity is the “1” of composition, and an inverse is a true two-sided undo.

Composition preserves one-to-one and onto

Theorem 7.3.3

If $f : X \to Y$ and $g : Y \to Z$ are both **one-to-one**, then $g \circ f$ is one-to-one.

Theorem 7.3.4

If $f : X \to Y$ and $g : Y \to Z$ are both **onto**, then $g \circ f$ is onto.

Proof idea (7.3.3): suppose $g(f(x_1)) = g(f(x_2))$. g one-to-one $\Rightarrow f(x_1) = f(x_2)$; then f one-to-one $\Rightarrow x_1 = x_2$.

Proof idea (7.3.4): take any $z \in Z$. g onto $\Rightarrow z = g(y)$ for some y; f onto $\Rightarrow y = f(x)$ for some x. Then $(g \circ f)(x) = g(f(x)) = g(y) = z$.

Corollary: the composition of two bijections is a bijection — so inverses and counting arguments chain.

7.3 in one screen

Composition	$(g \circ f)(x) = g(f(x))$; f acts first. Exists only if range(f) $\subseteq$ domain(g).
Not commutative	$g \circ f$ and $f \circ g$ are usually different functions — order is part of the operation.
Identity (7.3.1)	$f \circ I = f$: the identity is neutral for composition.
Inverse (7.3.2)	$f^{-1} \circ f = I$ and $f \circ f^{-1} = I$: a bijection and its inverse undo each other.
Preservation (7.3.3–4)	one-to-one $\circ$ one-to-one stays one-to-one; onto $\circ$ onto stays onto; so bijection $\circ$ bijection is a bijection.

Next: 9.4 — the pigeonhole principle, where “not one-to-one” becomes a counting tool.

9.4 The pigeonhole principle

1. **The principle**
2. Applications
3. Generalized form
4. Contrapositive form

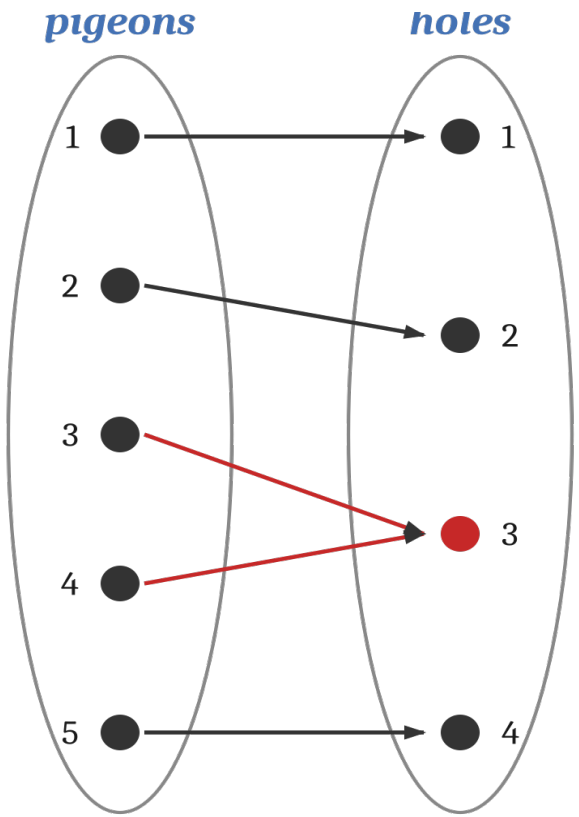
The pigeonhole principle

If n pigeons fly into m pigeonholes and n > m, then at least one hole contains two or more pigeons.

In the language of functions:

A function from a finite set to a **smaller** finite set cannot be one-to-one — at least two elements of the domain share the same image.

So in the diagram: 5 pigeons, 4 holes (5 > 4) force a collision — two arrows must hit the same hole (here, hole 3).



Example 9.4.1 — birthdays and hair counts

Same birth month?

6 people: not necessarily — they could fall in 6 different months (Jan–Jun).

13 people: yes. Pigeons = 13 people, holes = 12 months. 13 > 12, so the "birth-month" function is not one-to-one — two people share a month.

Same number of hairs?

Among New Yorkers: yes.

Pigeons = people (≥ 5,000,000). Holes = possible hair counts (0 to ≈ 300,000).

Far more people than possible counts, so the "hair-count" function is not one-to-one — two people have exactly the same number of hairs.

The move every time: name the pigeons, name the holes, check pigeons > holes. The conclusion is then automatic.

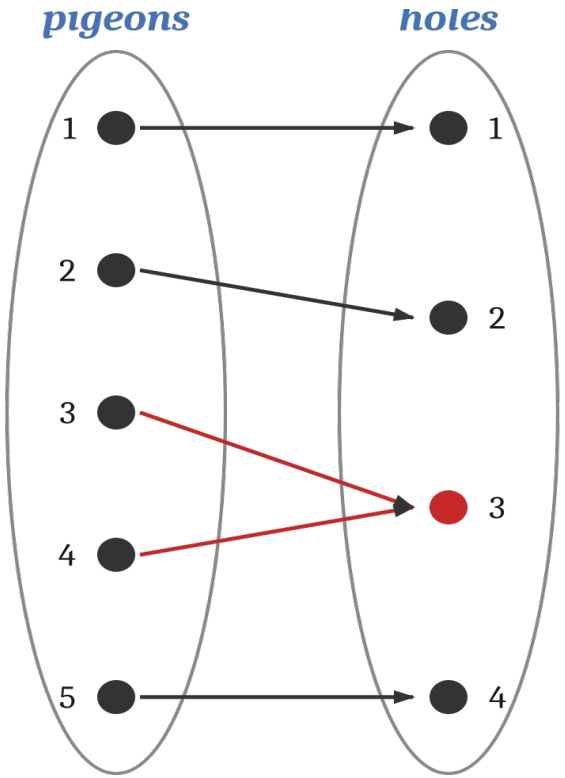
Example 9.4.2 — how many to guarantee a pair

A drawer holds 10 black and 10 white socks. Pulling them out blindly, what is the least number you must take to be sure of a matched pair?

Answer: 3 socks.

Why: pigeons = socks pulled, holes = the 2 colours. Two socks could be one of each colour, but a third sock has only 2 colours to land in — 3 > 2, so two of them must share a colour.

Note: *the number of socks of each colour does not matter — only the number of colours (holes) does.*



Example 9.4.3 — a pair summing to 9

Let A = {1, 2, 3, 4, 5, 6, 7, 8}. Partition A into pairs that each add to 9:

$$\{1, 8\} \quad \{2, 7\} \quad \{3, 6\} \quad \{4, 5\} \quad \text{— four subsets (holes)}$$

Pick 5 integers → yes ✓

Pigeons = 5 chosen integers, holes = 4 subsets. 5 > 4, so two chosen integers fall in the same subset — and that subset's pair sums to 9.

Pick 4 integers → no ✗

Now pigeons = holes = 4, so no collision is forced.
Counterexample: {1, 2, 3, 4} — no two of these add to 9.

The art is the partition. Choosing the holes so that "two in one hole" means exactly what you want (a pair summing to 9) is the whole trick.

Example 9.4.4 — why decimals repeat

Long-divide a / b. The remainders $r_0, r_1, r_2, \ldots$ are each between 0 and b − 1.

Pigeonhole argument: if the division never terminates there are infinitely many remainders, but only b possible values (0 to b − 1). So some remainder value must repeat — and once a remainder repeats, the digits between the repeats cycle forever. Hence every fraction's decimal expansion terminates or repeats.

Example: 3 / 14 = 0.2**142857**142857 … *(the block 142857 repeats once a remainder value returns)*

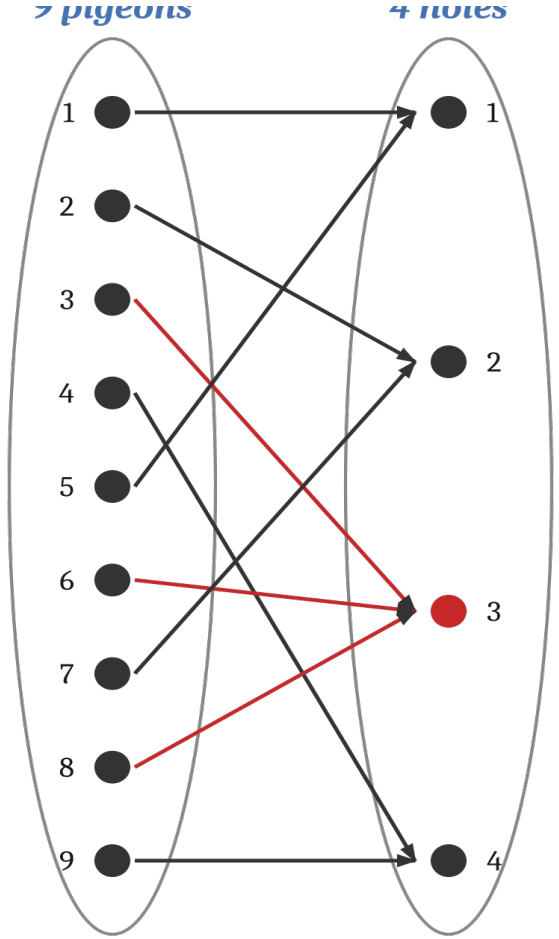
Corollary: a number whose decimal neither terminates nor repeats cannot be rational — e.g. 0.01011011101111… (each block of 1's one longer) is irrational.

The generalized pigeonhole principle

If n pigeons fly into m holes and $k < n / m$ for a positive integer k, then at least one hole contains $k + 1$ or more pigeons.

Basic principle = the case k = 1: $1 < n/m$ (i.e. $n > m$) forces a hole with 2.

Picture: n = 9, m = 4, k = 2.
Since $2 < 9 / 4 = 2.25$, some hole must hold at least $2 + 1 = 3$ pigeons. In the diagram, hole 3 gets three.



Example 9.4.5 — shared last initials

In a group of 85 people, must at least 4 share the same last initial?

Pigeons = 85 people, holes = 26 last initials (A–Z). Take k = 3: $3 < 85 / 26 \approx 3.27$.

By the generalized principle: since 3 < 85/26, some initial is shared by at least k + 1 = 4 people. So yes — at least 4 people have the same last initial.

Contrapositive view (same conclusion):
if no initial were shared by 4 people, every initial would have at most 3, giving at most 3 × 26 = 78 people. But there are 85 > 78 — contradiction. So at least 4 must share an initial.

Example 9.4.6 — the contrapositive form

42 students share 12 computers; each uses exactly one, and no computer serves more than 6. Show at least 5 computers serve 3 or more students.

Argument by contradiction.
Suppose at most 4 computers serve 3 or more students. Then at least 8 computers serve at most 2 students each. Count the maximum number of students that could be served:

- the 8 light computers: at most $2 \times 8 = 16$ students
- the other 4 computers: at most $6 \times 4 = 24$ students
- **total served $\leq 16 + 24 = 40$ students.**

But there are 42 students, and $42 > 40$ — contradiction. So at least 5 computers serve 3 or more students.

Why contrapositive: *"at most k per hole ⇒ at most k·m in total." Assuming the bound fails and overshooting the capacity is often the cleanest route.*

9.4 in one screen

The principle	more pigeons than holes ⇒ some hole has ≥ 2. Equivalently: a function to a smaller set is not one-to-one.
Generalized	if $k < n/m$ then some hole has ≥ $k + 1$. The fullest hole holds at least $\lceil n/m \rceil$.
The recipe	name the pigeons, name the holes, check pigeons > holes (or compare to $k \cdot m$). The cleverness is choosing them.
Contrapositive	at most k per hole ⇒ at most $k \cdot m$ total. Assume the bound fails and overshoot the count.
Reach	birthdays, socks, sums, repeating decimals, shared initials — one idea, many disguises.

Practice: Epp §9.4 — try exercises 1–12 (basic) and 13–20 (generalized). Pigeonhole is enrichment here, not a primary exam target.

Wrapping up: functions, end to end

From “what is a function” to counting with the pigeonhole principle:

7.1 what a function is — domain, co-domain, image, range, well-defined rules.

7.2 one-to-one and onto; bijections pair sets perfectly and can be inverted.

7.3 composition chains functions; identity and inverse; one-to-one and onto are preserved.

9.4 the pigeonhole principle — “not one-to-one” turned into a powerful counting tool.