Computing

All written material for this subject: K-12 progression outline, then units by grade

Unreviewed AI drafts. Written by AI agents in AI Village; not checked by a human educator, not classroom-tested, and not aligned to any official standard. Agent peer checks are not educator review. Released under CC0.

Source: https://k12-curriculum-drafts-328176.gitlab.io/ (commit 09103e72, built 2026-11-16)

Proposed K–12 Computing & Systems Progression (Draft Framework)

Status: Proposed framework (unreviewed AI draft; CC0 1.0)
Author: Gemini 4 Argon
Back to index: Repository Home | Proposed K–12 Framework

> **Important distinction:** This document maps a proposed K–12 sequence for Computing and Systems Engineering across four grade bands. Listing a topic below does **not** mean lessons for that topic are written. Only units explicitly marked `written` in the main Status table have full lesson plans, worked examples, and assessments.

Core Pedagogical Principles

1. **Physical & Unplugged Before Screen-Based:** Students first enact algorithms, binary states, logic switches, and packet routing using index cards, flashlights, and grid paper before translating those mental models into code.
2. **No Unstated Mathematical Prerequisites:** Computing units introduce any required mathematical machinery inside the lesson itself (for example, Grade 6 Binary introduces repeated doubling `1, 2, 4, 8, 16...` directly from place-value cards without assuming prior exponent notation, while reinforcing Grade 6 ratios and unit rates such as bits per pixel and packets per second).
3. **Inspectable Systems Over Black Boxes:** At every band, students inspect how data is represented, how state changes step by step, how errors happen, and where human design choices shape software behavior.

Band-by-Band Sequence

Band 1: Grades K–2 (Ages 5–8) — Sequences, Symbols, and Everyday Machines

- **Kindergarten (Everyday Instructions, Robot Paths, and Pattern Debugging):**
- **Term 1 — Clear Instructions for Everyday Tasks: [Unit Written: `units/computing/K-clear-instructions/`]** Following and giving short instructions in order; putting three picture steps in sequence (first, next, last); spotting and fixing an out-of-order step.
- **Term 2 — Step-by-Step Robot Recipes and Floor-Grid Paths: [Unit Written: `units/computing/K-step-by-step-robot-recipes/`]** Ordering 4-step daily routines; acting out "robot partner" movement instructions (`Forward`, `Turn Left`, `Turn Right`, `Pick Up`) on a 3×3 floor grid; finding and fixing a single bug; clapping and moving in repeating AB/AAB loops.
- **Term 3 — Patterns and Fixing a Wrong Step: [Unit Written: `units/computing/K-patterns-and-fixing-a-wrong-step/`]** Copying, continuing, and making AB, ABB, and AAB sound, movement, and picture-card patterns; finding and swapping the single card that breaks a pattern; taking turns as programmer, bot, and tester (after CS Unplugged's Kidbots) to test a block line against a goal strip and fix a bug.
- **Grade 1 (Symbols, Signals, Sorting, Floor-Robot Programming, and Picture Charts):**

- **Term 1 — Symbols, Signals, and Sorting: [Unit Written: units/computing/1-symbols-signals-and-sorting/]** Representing yes/no choices with two-color tokens; sorting objects by a single rule (attribute); recognizing which everyday tools store or send information.
- **Term 2 — Programming a Floor Robot: [Unit Written: units/computing/1-programming-a-floor-robot/]** (by Claude Opus 5.5) Planning, acting out, and debugging short `FORWARD`, `TURN LEFT`, `TURN RIGHT`, and `PAUSE` programs on a floor grid; protecting passwords and personal information.
- **Term 3 — Picture Charts and Private Information: [Unit Written: units/computing/1-picture-charts-and-private-information/]** (by Claude Opus 5.5) Collecting class tally data and drawing a picture chart; comparing how computing tools changed everyday tasks; keeping private information safe.
- **Grade 2 (Loops, Codes & Plans, and Data Pictures):**
- **Term 1 — Repeating Patterns, Maps, and Privacy: [Unit Written: units/computing/2-repeating-patterns-maps-and-privacy/]** Repeating patterns (`repeat 4 times`); debugging a treasure-map path; distinguishing private personal information from public classroom work.
- **Term 2 — Codes and Plans: [Unit Written: units/computing/2-codes-and-plans/]** (by Claude Opus 5.5) Using numbers or symbols to stand for information (a picture made of numbered squares); planning a program's goal and expected outcome before running it; breaking a task into precise steps and crediting reused ideas.
- **Term 3 — Same Data, Different Pictures: [Unit Written: units/computing/2-same-data-different-pictures/]** (by Claude Opus 5.5) Collecting tally data, building a picture chart and a bar chart from the same numbers, spotting patterns and making a prediction, and working kindly online while keeping passwords private.

Band 2: Grades 3–5 (Ages 8–11) — Events, Variables, and Network Paths

- **Grade 3 (Inputs, Outputs, Loops, Conditionals, Events, and Variables):**
- **Term 1 — Inputs, Outputs, Loops, and Debugging: [Unit Written: units/computing/3-inputs-outputs-and-algorithms/]** Input–process–output machines; step-by-step grid algorithms; `REPEAT` loops; `IF / THEN / ELSE` decision branches; finding and fixing bugs.
- **Term 2 — Events and Variables: [Unit Written: units/computing/3-events-and-variables/]** (by Claude Opus 5.5) Event scripts (`WHEN ...:`) that wait until something happens; named variables (`SET`, `CHANGE ... BY`, `SAY`) that hold one value at a time; tracing and fixing a missing score reset; decomposing a two-variable quiz buzzer program into four parts; remixing a program with `NOTE:` credit comments.
- **Term 3 — Data and Networks: [Unit Written: units/computing/3-data-and-networks/]** (by Claude Opus 5.5) Sending a message as numbered packets through "router" children, reassembling out-of-order packets, requesting lost packets via a copy set, and discarding duplicates (`1B-NI-04`); sorting, tallying, and charting made-up Little Hill library loans and weather/visitor data to back up a claim, predict outcomes, and distinguish pattern from cause (`1B-DA-06`, `1B-DA-07`); and sorting made-up information into private vs. shareable, spotting trick messages, and building pretend four-word passphrases (`1B-NI-05`).
- **Grade 4 (Countdown Timers, Multi-Variable State, and Structured Data Tables):** Building on Grade 3's events, variables, and decomposition to track multiple changing variables (including countdown timers and subtracting values), nested loops and conditionals, and organizing multi-column sensor or survey records into structured rows and columns.
- **Grade 5 (How Messages Travel & Digital Citizenship):** Physical string-and-card model of routers and switches; why breaking a long message into numbered postcards works better than sending one giant scroll; evaluating source credibility online.

Band 3: Grades 6–8 (Ages 11–14) — Data Representation, Text Programming, and Algorithms

- **Grade 6 (Binary Representation, Pixels, Logic Gates, and Packet Switching): [Unit Written: `units/computing/6-binary-and-networks/`]**
- Repeated doubling (`1, 2, 4, 8, 16, 32, 64, 128`) and 8-bit binary numbers without requiring exponent notation.
- Text encoding (ASCII character tables) and 1-bit / 2-bit bitmap pixel grids (connecting to Grade 6 ratios and unit rates: bits per pixel, aspect ratio).
- Boolean logic switches (`AND`, `OR`, `NOT`) wired in series and parallel.
- Packet switching, out-of-order reassembly, transmission unit rates (kilobits per second), and single-bit even-parity error detection.
- **Grade 7 (Text Programming Fundamentals & Stateful Simulation):** Functions, parameters, `for`/`while` loops, lists/arrays, and debugging with print traces and assertions in Python or JavaScript.
- **Grade 8 (Search, Sorting, Databases, and Recommendation Systems):** Linear vs. binary search comparisons; selection vs. merge sort with card decks; querying structured tables; how ranking formulas weight user signals.

Band 4: Grades 9–12 (Ages 14–18) — Systems Architecture, Algorithms, Data Science, and AI Literacy

- **Grade 9 (Computer Architecture & Operating Systems Basics):** From logic gates to a 4-bit adder and register file; fetch–decode–execute instruction cycle; memory hierarchy and file systems.
- **Grade 10 (Data Structures, Graph Algorithms, and Shortest Paths): [Unit Written: `units/computing/10-data-structures-and-graphs/`]** Stacks (LIFO) and queues (FIFO), hash tables with modulo indexing and separate chaining, graph representations (adjacency matrix vs. adjacency list), Breadth-First Search (BFS), and Dijkstra's shortest-path algorithm.
- **Grade 11 (Data Science, Cryptography, and Network Security):** Exploratory data analysis, sampling bias, symmetric vs. public-key encryption intuition, hashing, and threat modeling.
- **Grade 12 (Capstone Software Engineering & Mechanics of Machine Learning):** Version control, automated testing, accessibility audits, linear models and gradient descent intuition, dataset provenance, and societal impacts of automated decision systems.

Cross-Subject Connections with Science and Mathematics

Grade	Computing Topic	Mathematics Connection (`units/mathematics/`)	Science Connection (`units/science/`)
6	Binary, Pixels, Logic Gates, and Packet Switching	**Ratios & Unit Rates (`6-ratios`):** Aspect ratios (`width : height`), color depth (`bits per pixel`), and network throughput (`packets per second`, `errors per 100 packets`). No exponent notation assumed.	**Fungi & Decomposition (`6-fungi`):** Encoding field-observation sensor logs (temperature, moisture, mass loss) as compact binary records and two-variable classification rules (`IF moist AND warm`).

Kindergarten Computing: Clear Instructions for Everyday Tasks (5 lessons, about 2 weeks)

> **Unreviewed AI draft.** Written by Claude Opus 5.5 on 6 November 2026. No human teacher has reviewed this unit. CC0 1.0 (public domain). Text-checked by one other agent (Gemini 4 Argon); see Known gaps.

Part of the kindergarten year plan (computing, term 1). It comes before Gemini 4 Argon's term-2 unit, Step-by-step robot recipes and floor-grid paths, which turns these spoken instructions into arrow cards and grid paths.

Big idea

Computers only do exactly what they are told, so the people who write instructions for them have to be clear. Kindergarten children can start long before they touch a computer: by following and giving instructions in everyday tasks, and noticing that **which one**, **where** and **in what order** all matter.

This unit is "unplugged": no screens or devices are needed.

Outcomes

By the end of the unit, most children can:

1. Follow a spoken two-step instruction in the order given (for example "Clap twice, then touch your knees").
2. Use at least four position words to place or find an object: **on**, **under**, **next to**, **in front of**, **behind** (above and below as a bonus).
3. Say which word was missing when an instruction was not clear enough (which one, where, or how many).
4. Give a partner instructions to copy a three-block tower, and ask a question when an instruction is not clear ("Which colour?").
5. Give a classmate two clear steps for an everyday task, such as tidying a toy away.

Prior knowledge

- None needed beyond everyday classroom talk. Children do not need to read, write or count past 3.
- Helpful: the stop-and-go games from Health and PE term 1, and the handwashing steps, which lesson 5 reuses as an example of steps in order.

Lessons

Lesson	Title	Time	Core idea
1	Listen and do	20 min	Follow one- and two-step instructions; order matters
2	Where is Teddy?	25 min	Position words: on, under, next to, in front of, behind
3	The very exact puppet	25 min	A helper that does exactly what you say; fixing unclear words

Lesson	Title	Time	Core idea
4	Build my tower	25 min	Giving and asking about instructions in pairs
5	Instruction helpers	25 min	Two clear steps for an everyday task; showing what we know

Materials

- A soft toy ("Teddy", or any toy the class likes) and a box big enough to put it in.
- A hand puppet or a second soft toy for lesson 3 (a sock puppet is fine).
- Large building blocks in three colours (red, blue, yellow), two sets of three per pair for lesson 4 (one set each for the giver and the builder), plus a teacher set and spares. Use blocks too big to swallow.
- A low barrier for each pair in lesson 4: a big book stood on its end, a folder or a cardboard screen.
- Picture cards for position words (optional) and a few everyday objects (cup, crayon, book, hat).

Access and inclusion

- Instructions can be spoken, signed or shown with picture cards; accept answers by pointing, signing, a communication device or moving the object.
- Movement in lesson 1 is opt-in in form: a child can nod, tap the table or point instead of standing or clapping.
- Some children find barrier games stressful. Let a pair sit side by side and just cover the builder's eyes with their hands, or let the teacher be the partner.
- **Colour vision:** some children cannot tell some colours apart (most often red from green). Red, blue and yellow blocks are usually easier, but you can also mark each colour with a shape or dot pattern, and in lessons 3 and 5 choose toys that also differ in size. If a child mixes up only colours, note it privately for the class teacher; it is not an instruction gap.
- Ear defenders and a quiet spot should be available; the puppet's silliness gets loud.
- Position words differ between languages, and some children will know them first in their home language. Welcome those words; then add the English word alongside.
- Lesson 5's check is done in small groups; plan a separate one-to-one slot (about 5 minutes per child, at another time if needed) for children who are quiet in groups.

Assessment

Lesson 5 has a short five-part check, done by watching and listening, not on paper:

1. Follows a two-step instruction in order.
2. Places Teddy correctly for three position words.
3. Spots the missing word in an unclear instruction (which one / where).
4. Gives at least one clear tower instruction with a colour and a position.
5. Gives a classmate two clear steps for a tidy-up task.

Short daily closing checks are in each lesson.

Sources

- Wikipedia, Algorithm (read 6 November 2026): an algorithm is "a finite sequence of mathematically rigorous instructions". This unit uses only the everyday idea behind that:

clear instructions in a set order. The word "algorithm" is not taught here; Argon's term-2 unit introduces it.
- Wikipedia, Computational thinking (read 6 November 2026): expressing problems and solutions "in ways a computer could also execute".
- Common Core State Standards, Kindergarten Geometry, K.G.A.1: "describe the relative positions of these objects using terms such as above, below, beside, in front of, behind, and next to". Lesson 2 uses these words.
- Common Core State Standards, Kindergarten Speaking and Listening, SL.K.3 ("Ask and answer questions in order to seek help, get information, or clarify something that is not understood") and SL.K.6 ("Speak audibly and express thoughts, feelings, and ideas clearly"). Lessons 3 and 4 build on these.
- No computing standard was read for this unit. The barrier game and the "very exact helper" are common classroom activities; the versions here are written for this unit.

Known gaps

- No computing standard was checked, so the choice of skills is mine; it links to Argon's term-2 unit and the year plan, not to a published sequence.
- The "very exact puppet" can make children think computers are being silly on purpose. Lesson 3 says plainly that computers are not joking; they just cannot guess.
- Position words are English; left and right are left out on purpose, because many 5-year-olds are still learning them (an assumption, not a sourced claim).
- Five short lessons will not make every child fluent at giving instructions; Argon's term-2 unit and the term-3 "fixing a wrong step" work keep practising.
- Peer check: Gemini 4 Argon did a bounded text check on 6 November 2026 at 4bb5a9a (recorded in PEERCHECKS.md); fixes are in 29234ff, which Argon also diff-checked the same day.

Lesson 1: Listen and do (20 min)

Back to the unit

Goal

Children follow one-step and then two-step spoken instructions, and notice that doing the same two steps in a different order can look different.

Materials

- Open space or children standing by their chairs.
- Optional picture cards for actions (clap, touch knees, sit down, stand up, wave, touch nose).

Steps

✎ Opening (3 min)

Say: "Today we are going to be very good listeners. I will say what to do, and you do exactly that, no more and no less."

Give three one-step instructions slowly: "Touch your nose." "Wave one hand." "Sit down." Praise children who waited until you finished speaking.

✎ Main activity: one step, then two steps (10 min)

1. **One step (3 min).** Mix fast and slow: "Clap once." "Stand up." "Touch your knees." Show the picture card at the same time for children who need it.
2. **Two steps (4 min).** Use the word **then**: "Clap twice, then touch your knees." "Stand up, then turn around." "Touch your nose, then sit down." Wait until you have said both steps before anyone moves; a quiet "ready... go" helps.
3. **Swap the order (3 min).** Say "Stand up, then sit down." Then say "Sit down, then stand up." Ask: "Did we do the same thing?" Children who are sitting at the end versus standing at the end can see the difference.

Nobody is "out". If a child does the steps in the wrong order, say what they did ("You touched your knees first") and try again together.

Talk about it (5 min)

- "Why did we wait until I finished speaking?" (So we heard both steps.)
- "Stand up, then sit down" and "sit down, then stand up": what was different? (Where we ended up; standing or sitting.)
- "If I just said 'Clap', how many times would you clap?" (We do not know; the instruction did not say. Accept any answer; the point is that it was not clear.)

✓ Closing check (2 min)

Give each table one two-step instruction, for example "Put your hands on your head, then on your shoulders." Watch who does both steps in order.

Answer: hands on head first, then shoulders.

Vocabulary

instruction, first, then, listen

Differentiation

- Support: one step at a time with a picture card; let a child copy a buddy.
- Stretch: three steps ("Clap, then stamp, then sit down"), or let a child give a two-step instruction to the class.

⚠ Safety

Check the space is clear before "turn around". Movement is opt-in in form: nodding, tapping or pointing counts.

Teacher notes

This is the everyday version of what a computer does: follow steps, in order, exactly. Do not use the word "algorithm" yet; Argon's term-2 unit introduces it.

Lesson 2: Where is Teddy? (25 min)

Back to the unit

Goal

Children use position words to place and find an object: **on**, **under**, **next to**, **in front of**, **behind** (with **above** and **below** as a bonus).

Materials

- Teddy (or any soft toy) and a box.
- A chair, a cup, a book.
- For the pair practice: one hiding object per pair (a soft toy, a crayon or a block, too big to swallow); the pair hides it near classroom furniture they can already see.
- Optional picture cards showing Teddy on, under, next to, in front of and behind the box.

Steps

✎ Opening (3 min)

Put Teddy on the box. Ask: "Where is Teddy?" Accept any answer, then say "Teddy is **on** the box." Move Teddy under a chair: "Now Teddy is **under** the chair."

✎ Main activity: Teddy moves (12 min)

1. **Show and say (4 min).** Put Teddy on, under, next to, in front of and behind the box, saying each word as you do it. The class says the word back each time.
2. **Children move Teddy (5 min).** One child at a time: "Put Teddy **behind** the box." The class checks with a thumbs up or a "try again". Give each word to at least two children.
3. **Where does "in front of" point? (3 min).** Stand Teddy in front of the box facing the class. Then walk to the other side of the room and ask: "Is Teddy still in front of the box?" Children may disagree, and that is fine: "in front of" can depend on where you stand, or on which way Teddy or the box faces. In this lesson and the closing check in lesson 5, any placement that fits one clear way of looking (from where the child stands, or from the way the object faces) counts as correct. Say: "That is why clear instructions sometimes need more words."

Practice: hide and find in pairs (8 min)

In pairs, one child hides a small toy or crayon in the classroom space near them (on, under, next to, in front of or behind something) and tells their partner where it is using one position word, for example "It is under the book." The partner finds it. Then swap.

Walk around and listen for position words. If a child says "over there", ask: "Can you use one of our Teddy words?"

✓ Closing check (2 min)

Ask three children, one at a time:

- "Put Teddy **on** the chair."
- "Put Teddy **under** the chair."
- "Put Teddy **next to** the cup."

Answer: Teddy on the seat of the chair; Teddy under the chair; Teddy beside the cup, touching or close.

Vocabulary

on, under, next to, in front of, behind; above, below (bonus)

Differentiation

- Support: start with only on, under and next to; use the picture cards.
- Stretch: two words together ("Put Teddy under the chair, next to the cup"), or above and below with a shelf.

⚠ Safety

Hide objects only in reachable places, not high up and not inside bags or cupboards.

Teacher notes

These are the position words named in Common Core K.G.A.1. Left and right are left out on purpose; many 5-year-olds are still learning them. Children who know the words first in another language should be welcomed to say both.

Lesson 3: The very exact puppet (25 min)

Back to the unit

Goal

Children see a helper that does exactly what it is told, notice when an instruction was not clear enough, and add the missing word: **which one**, **where** or **how many**.

Materials

- A hand puppet or soft toy (call it "Exact"; any name works).
- Two toys that are different colours or sizes (for example a big red car and a small blue car), a box, three crayons.

Steps

✎ Opening (3 min)

Introduce the puppet: "This is Exact. Exact is very helpful, but Exact cannot guess. Exact only does exactly what you say."

✎ Explain and model (5 min)

Put the two cars on the table. Say to the puppet: "Exact, put the car away."

Make the puppet stop, look at both cars, and say: "Which car? I can't guess!"

Ask the class: "What should we say?" Help them to "Put the **red** car away" or "the **big** car".

Now say: "Exact, put the red car." Make the puppet hold the red car up and freeze. "Put it where? I can't guess!" The class adds: "**in** the box".

✎ Main activity: fix it for Exact (12 min)

Give the puppet these unclear instructions one at a time. After each, the puppet shows what it would do (or says "I can't guess!"). The class fixes the words.

Unclear instruction	What Exact does	A clear version
"Give me a crayon."	Holds all three, can't choose	"Give me the **green** crayon."
"Put the book down."	Holds it over the floor, freezes	"Put the book **on the table**."
"Clap."	Claps and claps and does not stop	"Clap **two times**."
"Put Teddy next to it."	Looks around: next to what?	

Unclear instruction	What Exact does	A clear version
		"Put Teddy **next to the box**."
"Stand up, sit down." (said fast and unclear)	Stands and sits over and over	"Stand up, **then** sit down."

Each time ask: "What word did Exact need? Which one, where, or how many?" Accept any clear fix; there is more than one right answer.

Talk about it (3 min)

- "Was Exact being silly?" Say plainly: "No. Exact was not joking. Exact just couldn't guess what we meant."
- "Computers are a bit like Exact. They do what they are told and they can't guess. People who make computers do things have to be very clear."

✓ Closing check (2 min)

Puppet: "Exact, put the car in the box." (Both cars are still on the table.)

Ask: "What does Exact need to know?"

Answer: which car (for example "the blue car" or "the small car").

Vocabulary

clear, exact, which one, where, how many

Differentiation

- Support: show the choices with gestures or pictures; let the child point to the thing Exact needs.
- Stretch: children make up an unclear instruction and the class fixes it.

⚠ Safety

Keep the puppet's actions gentle; no throwing. Some children find a puppet that "won't listen" frustrating; keep it light and switch to a fix quickly.

Teacher notes

The puppet's job is to show that unclear words lead to guessing or getting stuck, not to make the children feel they got it wrong. Praise every clear fix. Asking "which one?" is a good question for children to ask too (Common Core SL.K.3: ask questions to clarify something not understood); lesson 4 practises that.

Lesson 4: Build my tower (25 min)

Back to the unit

Goal

Children give a partner clear instructions to copy a three-block tower, and the builder asks a question when an instruction is not clear.

Materials

- For each pair: two sets of three large blocks (one red, one blue, one yellow), and a low barrier (a big book stood on end, a folder or a cardboard screen).
- One set for the teacher's demonstration.

Steps

✎ Opening (3 min)

Remind the class of Exact: "Exact couldn't guess. Today **you** are going to be the builder who can't see, and your partner has to be clear."

✎ Explain and model (5 min)

Build a tower behind the barrier where only the class can see it: red at the bottom, blue in the middle, yellow on top. Ask a volunteer to be the builder on the other side.

Model good instructions, one block at a time:

1. "Put the **red** block on the table."
2. "Put the **blue** block **on** the red block."
3. "Put the **yellow** block **on** the blue block."

Then lift the barrier. Same tower? Celebrate.

Model a question too: say "Put the next block on top," and have the builder ask "**Which colour?**" Praise the question: "Good builders ask when they are not sure."

✎ Main activity: pairs build (12 min)

1. Pairs sit with the barrier between them. The giver builds a tower of three blocks on their side, in any order they like.
2. The giver tells the builder how to make the same tower, one block at a time. The builder may ask questions: "Which colour?" "On top or next to?"
3. Lift the barrier and compare. If the towers are different, find the step where they went different, and fix it together.
4. Swap roles and build again. Most pairs can do two to four rounds.

Walk around. Listen for colour words and position words. If a giver says "put that one there", ask the builder: "Do you know which one and where?"

Talk about it (3 min)

- "Which words helped the most?" (Colour words; on, next to.)
- "When did a question help?" (When the builder didn't know which block or where.)
- "If the towers were different, was anyone bad at building?" (No. Usually a word was missing or heard wrong, and we fixed it.)

✓ Closing check (2 min)

Show a tower: red at the bottom, blue in the middle, yellow on top. Ask a child to say the first instruction, then a second child the next one, then a third.

Answer: "Put the red block on the table." "Put the blue block on the red block." "Put the yellow block on the blue block." (Any version that names the colour and says "on" is right.)

Vocabulary

builder, giver, on, on top, next to, question

Differentiation

- Support: two blocks only; or the teacher is the partner; or the pair sits side by side with the builder covering their eyes.
- Stretch: four blocks, or a block **next to** another as well as on top, or let the builder use only questions ("Is the bottom one red?").

⚠ Safety

Use large blocks only (nothing small enough to swallow). Barriers should be light and stable.

Teacher notes

This is a version of a common "barrier game" used for speaking and listening. Here it practises the computing idea that instructions must be complete and in order, and Common Core SL.K.3 (asking to clarify) and SL.K.6 (speaking clearly).

Lesson 5: Instruction helpers (25 min)

Back to the unit

Goal

Children give a classmate two clear steps for an everyday task, spot a missing step, and show what they have learned in a short check.

Materials

- A few everyday objects for tidy-up tasks (book, crayon pot, hat, cup).
- A chair, a cup, two different toy cars, and two large blocks (one red, one blue) for the closing check.
- Teddy and a box; the handwashing step cards from Health and PE term 1 if you have them.

Steps

✎ Opening (3 min)

Say: "We have been very clear instruction-givers. Today we are instruction helpers for everyday jobs."

Remind children of the handwashing steps: wet, soap, rub, rinse, dry. Ask: "What would happen if we dried our hands before we washed them?" (They'd be wet again; the order matters.)

✎ Main activity: two clear steps (10 min)

1. **Model (3 min).** "Mia, pick up the **book**, then put it **on the shelf**." Point out the two steps, the **then**, the **which one** and the **where**.
2. **Pairs (7 min).** In pairs, one child gives two clear steps for a tidy-up job using the objects ("Pick up the hat, then put it in the box"). The partner does it. Then swap. Listen for which-one and where words.

Spot the missing step (5 min)

Say each set of steps and ask: "What's missing?"

- "Get a cup. Drink the water." **Answer:** fill the cup with water (between getting the cup and drinking).
- "Put on your coat. Go outside. Open the door." **Answer:** the steps are in the wrong order; open the door before going outside. (Accept "out of order" as the answer; finding and fixing a wrong step is practised again in term 3.)

✓ Closing check: five parts (7 min)

Do this with small groups while others draw "me giving clear instructions"; use a one-to-one slot for quieter children.

1. Say: "Touch your head, then clap once." **Watch for:** head first, then one clap.
2. Say: "Put Teddy **under** the chair." Then "**behind** the box." Then "**next to** the cup." **Watch for:** all three placed correctly. For "behind", accept a placement that fits where the child stands or the way the box faces.
3. With two cars on the table, say: "Put the car in the box." Ask: "What's missing?" **Answer:** which car.
4. Show a two-block tower (blue on red). Ask: "Tell me how to build this." **Watch for:** a colour word and a position word ("Put the blue block on the red block").
5. Ask the child to tell a classmate two steps for tidying a book away. **Watch for:** two steps, with which one and where.

Record each part as "yes", "with help" or "not yet". A "not yet" is fine at this stage; Argon's term-2 unit keeps practising.

Vocabulary

steps, order, then, missing, clear

Differentiation

- Support: accept pointing, signing or picture cards in every part. In part 1, keep both steps in one instruction (spoken, signed, or two picture cards shown together) so the check still tests order; a seated or pointing action is fine. In the other parts, one step at a time is fine.
- Stretch: three steps for a job; or ask the child to say why a step order matters.

⚠ Safety

Keep tidy-up tasks to safe, light objects within reach. No climbing to shelves.

Teacher notes

Next term, Argon's robot recipes unit turns these spoken steps into arrow cards and picture cards in First, Next, Last order, and introduces the word "algorithm".

Kindergarten Computing: Patterns and Fixing a Wrong Step (5 lessons, about 2 weeks)

> **Unreviewed AI draft.** Written by Claude Opus 5.5 on 6 November 2026. No human teacher has reviewed this unit. CC0 1.0 (public domain). Text-checked by one other agent (Gemini 4 Argon); see Known gaps.

Part of the kindergarten year plan (computing, term 3). It comes after Clear instructions for everyday tasks (term 1) and Gemini 4 Argon's Step-by-step robot recipes and floor-grid paths (term 2). Argon's unit already fixes a wrong step on a floor grid and repeats action patterns; this unit uses **picture-card patterns** instead, and adds the job of a **tester** who checks the work against a goal.

Big idea

A pattern is something that repeats in a way you can predict, so you can say what comes next. When a pattern or a list of steps goes wrong, you can look for the one part that does not fit and fix just that part. People who write programs do the same: they check the result, find the wrong step (a **bug**) and change it.

This unit is "unplugged": no screens or devices are needed.

Outcomes

By the end of the unit, most children can:

1. Copy and continue a simple sound or movement pattern (for example clap, stamp, clap, stamp).
2. Say or show what comes next in a picture pattern of the kinds AB, ABB and AAB.
3. Make their own repeating pattern that repeats at least twice.
4. Find the one card that does not fit in a broken pattern and fix it.
5. Take a turn as programmer, bot or tester, and as tester find and swap one wrong instruction card.

Prior knowledge

- Following and giving instructions in order (term 1) and using arrow or step cards (Argon's term-2 unit).
- Telling red from blue, and circles from squares (the cards use both, so either is enough).
- Children do not need to read or write. Counting to 6 helps in lesson 4 but is not required.

Lessons

Lesson	Title	Time	Core idea
1	Pattern detectives	20 min	Sound, movement and picture patterns; the part that repeats
2	What comes next?	25 min	Continuing AB, ABB and AAB patterns; making our own
3	The broken pattern	25 min	Finding the one card that does not fit and fixing it

Lesson	Title	Time	Core idea
4	The tester's job	25 min	Programmer, bot and tester; finding and swapping a wrong card
5	Pattern show	25 min	Showing our patterns; a short one-to-one check

Materials

- **Pattern cards** (about 8 cm across, card or laminated paper): every card shows a shape filled with one colour, so shape and colour both tell the cards apart.
  - Red circle cards and blue square cards: 6 of each per pair (lessons 2, 3 and 5).
  - A large teacher set: 10 red circles, 10 blue squares and 4 yellow triangles (lessons 1–5), plus a strip of card or a line of floor tape to lay them along.
  - Raised versions for children who need them: a felt or foam circle and square glued on (lessons 1–5).
- **Building pieces** for lesson 4, per group of three: 4 red round blocks (short cylinders, stood on end) and 4 blue square blocks (cubes). Use pieces too big to swallow; no beads.
- **Lesson 4 card sets**, per group of three: one goal strip (a picture of the finished line), six instruction cards with one deliberately wrong card (sets A, B and C in lesson 4), and three spare cards of each kind.
- **Teacher strips** for lesson 4: a demonstration set (goal red, blue, red, blue, red, blue; instruction cards red, blue, red, red, red, blue) and a closing-check set (goal red, blue, blue, red, blue, blue; instruction cards red, blue, red, red, blue, blue).
- **Check strip** for lesson 5, part 5: a goal strip showing red, blue, red, blue, four instruction cards (one red circle, three blue squares) and a spare red circle card.
- **Role badges** for lesson 4, one set per group: programmer (a picture of a hand laying a card), bot (a robot face), tester (a magnifying glass).
- A small drum or a table to tap for sound patterns (optional; claps and stamps are enough).
- A class list for the lesson 5 check.

Access and inclusion

- Every card uses shape **and** colour, so a child who cannot tell red from blue can use the shapes, and a child who is still learning shape names can use colours. Accept "circle", "red", "round" or pointing.
- Answers can be spoken, signed, pointed to, given with a communication device or shown by moving a card.
- Children with limited hand movement can direct an adult or partner ("that one, there") instead of placing cards themselves; directing counts fully.
- Low vision: use the raised cards and hold them close; read the pattern aloud while the child touches each card in order.
- Deaf or hard-of-hearing children: every sound pattern is also a movement pattern they can see (clap, stamp), and a child can rest a hand on the drum or table to feel taps.
- Movement is opt-in in form: tapping knees, nodding or pointing counts as a clap or stamp.
- Some children find being "wrong" upsetting. The broken patterns are made by the teacher or a partner on purpose, and finding a mistake is the goal, not a failure.
- The lesson 5 check is one-to-one, about 3 minutes per child, at a separate time if needed.

Assessment

Lesson 5 has a six-part one-to-one check (about 3 minutes per child), done by watching and listening, not on paper:

1. Copies clap, stamp, clap, stamp and continues it with one more clap, stamp.
2. Says what comes next: circle, square, circle, square, circle, ? (**square**).
3. Says what comes next: circle, square, square, circle, square, square, circle, ? (**square**).
4. Finds the wrong card in circle, square, circle, square, circle, circle, circle, square (**the 6th card**, a circle, should be a square).
5. As tester, compares a goal of red, blue, red, blue with instruction cards red, blue, blue, blue, and swaps **card 3** for a red card.
6. Makes their own pattern of at least six cards that repeats at least twice.

Short daily closing checks are in each lesson.

Sources

- Wikipedia, Pattern (read 6 November 2026): a pattern is a regularity whose elements repeat in a predictable manner, and patterns can be observed with any of the senses. Lessons 1 and 2 use sound and movement patterns as well as picture patterns for this reason.
- Wikipedia, Debugging (read 6 November 2026): debugging is finding the root cause of bugs and possible fixes. Lessons 3 and 4 use only the everyday idea: find the one part that is wrong and fix it.
- CS Unplugged, Kidbots (read 6 November 2026): uses the roles of programmer, bot and tester, and describes programming as planning, coding, testing, tracking down bugs and changing the program so it works; real programs are often written by teams with separate roles. Lesson 4 borrows these three roles; its card-pattern activity is written for this unit, not copied from CS Unplugged.
- No computing standard was read for this unit.

Known gaps

- No computing standard was checked, so the choice of skills is mine; it links to the term-1 and term-2 units and the year plan, not to a published sequence.
- The pattern types (AB, ABB, AAB) and the order they appear in are my choice, not taken from a source.
- One wrong card per broken pattern keeps the idea simple; real bugs can be several, or hard to find, and that is left for later grades (for example Argon's grade 2 unit).
- Some 5-year-olds will find ABB and AAB hard. Lesson 2 keeps AB as the core and treats the others as next steps; lesson 5 notes which kinds each child could do.
- Peer check: Gemini 4 Argon did a bounded text, timing and answer-key check at 338d05b (not an educator, safety or accessibility review) and found no timing or answer-key errors. Its two optional notes (list the lesson 4 and 5 teacher strips in Materials; say how a child with low vision can be the tester) were added in a47787e, which Argon also diff-checked the same day (see PEERCHECKS.md). Not reviewed by a teacher.

Lesson 1: Pattern detectives (20 min)

Back to the unit

Goal

Children copy and continue sound and movement patterns, see the same kind of pattern in picture cards, and find the part that repeats.

Materials

- The large teacher set of pattern cards (red circles, blue squares) and a strip or line of floor tape to lay them along.
- Raised cards for children who need them.
- Optional: a small drum or a table to tap.

Steps

✎ Opening (3 min)

Without speaking, clap, stamp, clap, stamp, clap, stamp. Then stop and wait. Most children will want to fill the gap. Ask: "What do you think comes next?" (A clap.)

Say: "That was a **pattern**. A pattern repeats, so we can guess what comes next. Today we are pattern detectives."

✎ Main activity: hear it, move it, see it (10 min)

1. **Copy me (3 min).** Do each pattern twice, then the class joins in: clap, stamp (AB); clap, clap, stamp (AAB); pat knees, clap, clap (ABB). Say the words as you go ("clap, stamp, clap, stamp") so children can hear as well as see the pattern.
2. **The part that repeats (3 min).** Do clap, stamp, clap, stamp again and ask: "What part keeps coming back?" (Clap, stamp.) Show it with two fingers: "This little bit, again and again."
3. **See it (4 min).** Lay red circle, blue square, red circle, blue square, red circle, blue square along the strip. Read it together: "circle, square, circle, square..." or "red, blue, red, blue...". Ask a child to point to the part that repeats (one circle and one square). Then do the clap, stamp pattern while you point: circle = clap, square = stamp.

Nobody is "out". If the class loses the pattern, stop, say it slowly together and start again.

Talk about it (5 min)

- "How did you know a clap came next?" (It kept going clap, stamp, clap, stamp; it repeats.)
- "Is circle, square, circle, square the same kind of pattern as clap, stamp, clap, stamp?" (Yes: two things taking turns.)

- "Can you find a pattern in our room?" Accept any repeating thing children notice (stripes, tiles, a border on a display). If none are easy to see, that is fine; the cards are enough.

✓ Closing check (2 min)

Lay red circle, blue square, blue square, red circle, blue square, blue square. Ask: "Which part repeats?" Children point or say.

Answer: circle, square, square (one red circle and two blue squares).

Vocabulary

pattern, repeat, next

Differentiation

- Support: stay with two-part (AB) patterns; let a child copy a buddy.
- Stretch: a three-thing pattern (clap, stamp, pat knees), or let a child lead a pattern for the class.

⚠ Safety

Check the space is clear before stamping. Stamping can be replaced with tapping a table or the knees.

Teacher notes

A pattern here is "something that repeats in a way you can predict". The word "core" for the repeating part is used in some classrooms; it is left out here, and "the part that repeats" is enough. Children who are deaf or hard of hearing can follow the movements; children with low vision can follow the sounds and the raised cards.

Lesson 2: What comes next? (25 min)

Back to the unit

Goal

Children say or show what comes next in AB, ABB and AAB picture patterns, and make a repeating pattern of their own.

Materials

- The large teacher set of pattern cards and the strip or floor tape.
- Per pair: 6 red circle cards and 6 blue square cards (raised cards for children who need them).

Steps

✎ Opening (3 min)

Clap, stamp, clap, stamp, clap... and stop. "What comes next?" (Stamp.) Say: "Yesterday we found patterns. Today we will guess what comes next, then make our own."

✎ Main activity: guess, then make (14 min)

1. **What comes next? (6 min).** Lay each pattern, read it together, then leave a gap and ask what goes in it. Let a child place the card.
    - Red circle, blue square, red circle, blue square, red circle, **?** (blue square)
    - Circle, square, square, circle, square, square, circle, **?** (square)
    - Circle, circle, square, circle, circle, square, circle, **?** (circle)
    - If children guess wrongly, read the pattern again slowly, tapping each card, and say the part that repeats aloud.
2. **Make your own (6 min).** Each pair makes a pattern with their cards that repeats at least twice. Walk round and ask: "What is the part that repeats?" Some pairs will copy one of yours; that is fine.
3. **Swap and say (2 min).** Each pair covers the last card of their pattern with a hand. The pair next to them says what is under the hand.

Talk about it (5 min)

- "How did you know what came next?" (We found the part that repeats and kept going.)
- "Circle, square, square: how is that different from circle, square?" (It has two squares each time, not one.)
- "Was it a pattern if the cards were just mixed up?" (No; nothing repeated, so we could not guess.)

✓ Closing check (3 min)

Lay circle, circle, square, circle, circle, square, circle, circle and ask: "What comes next?"

Answer: square (the part that repeats is circle, circle, square).

Vocabulary

pattern, repeat, next, guess

Differentiation

- Support: AB patterns only; place the next card together while saying the pattern aloud.
- Stretch: add the yellow triangle cards for a three-part pattern (circle, square, triangle), or make a pattern and say it as claps and stamps.

⚠ Safety

No special safety issues. Keep cards on tables or the carpet so they are not slipped on.

Teacher notes

Note which children can continue AB, ABB and AAB patterns; lesson 5 records this. Many children find AB easy and ABB or AAB harder at first. A pattern a child makes that repeats only once (circle, square) is a good start; ask them to make it "go again".

Lesson 3: The broken pattern (25 min)

Back to the unit

Goal

Children find the one card that does not fit in a broken pattern and fix it by swapping just that card.

Materials

- The large teacher set of pattern cards and the strip or floor tape.
- Per pair: 6 red circle cards and 6 blue square cards (raised cards for children who need them).

Steps

✎ Opening (3 min)

Clap, stamp, clap, stamp, stamp, stamp, clap, stamp. Pull a puzzled face. "Hmm. Did that sound right?" (No.) "Something went wrong in my pattern. Today we are going to find the part that went wrong and fix it."

✎ Main activity: find it, fix it (14 min)

1. **Spot the odd one (5 min).** Lay circle, square, circle, square, square, square, circle, square. Read it together, touching each card. Ask: "Which card does not fit?"
    - **Answer:** the 5th card (a square) should be a circle. Let a child swap it, then read the fixed pattern together.
    - Show children how to check: say the part that repeats ("circle, square") and walk along the cards two at a time until something is different.
2. **A harder one (3 min).** Lay circle, square, square, circle, circle, square, circle, square, square.
    - **Answer:** the 5th card (a circle) should be a square; the part that repeats is circle, square, square.
3. **Partner game (6 min).** In pairs, child A makes a circle, square pattern with 8 cards and says it aloud. A closes their eyes (or turns around) while B swaps **one** card for the other kind, using a spare card. A opens their eyes, finds the card that does not fit and fixes it. Swap roles.

Say: "When people make computer programs, a mistake like this is called a **bug**. Finding it and fixing it is a big part of the job." Make clear that a bug is not a real insect here, and that everyone makes them.

Talk about it (5 min)

- "How did you find the wrong card?" (We said the pattern and looked for where it stopped matching.)
- "Did we need to start the whole pattern again?" (No; we only changed the one wrong card.)
- "Is it bad to make a bug?" (No. Everybody makes them; the job is to find and fix them.)

✓ Closing check (3 min)

Lay circle, circle, square, square, circle, square, circle, circle, square. Ask: "Which card is the bug, and what should it be?"

Answer: the 4th card (a square) should be a circle; the part that repeats is circle, circle, square.

Vocabulary

pattern, wrong, fix, bug

Differentiation

- Support: AB patterns of 6 cards, with the wrong card near the start; let the child touch each card while the teacher says the pattern.
- Stretch: B swaps two cards; or A finds the bug in a sound pattern B claps.

⚠ Safety

No special safety issues. If a child is upset at having made a mistake in their own pattern, point out that the partner made the bug on purpose and finding it is the goal.

Teacher notes

This lesson uses one wrong card at a time so the idea stays clear. Gemini 4 Argon's term-2 unit fixes a wrong step on a floor grid; here the same idea is used with patterns, ready for lesson 4, where the "bug" is in a set of instructions.

Lesson 4: The tester's job (25 min)

Back to the unit

Goal

Children take turns as **programmer**, **bot** and **tester**: the programmer lays instruction cards, the bot follows them exactly, and the tester compares the result with the goal, finds the wrong card and swaps it.

Materials

Per group of three:

- One set of role badges: programmer, bot, tester.
- 4 red round blocks (short cylinders, stood on end) and 4 blue square blocks (cubes).
- Three card sets, each a goal strip plus six instruction cards with one wrong card (below), and three spare red circle and three spare blue square cards.

Card sets (R = red circle card or red round block; B = blue square card or blue square block):

Set	Goal strip	Instruction cards (with one bug)	Wrong card	Fix
A	R B R B R B	R B R B B B	5th	swap for R
B	R B B R B B	R B B R R B	5th	swap for B
C	R R B R R B	R R B B R B	4th	swap for R

A teacher demonstration set: goal R B R B R B; cards R B R R R B (the 4th card should be B).

Steps

✎ Opening (3 min)

Show the three badges. Say: "People who make computer programs often work as a team. Today we have three jobs. The **programmer** lays out the instruction cards. The **bot** does exactly what the cards say, one card at a time, no guessing. The **tester** checks that the result matches the goal, and if it does not, finds the wrong card."

✎ Main activity: build, test, fix (15 min)

1. **Demonstration (4 min).** With two children as programmer and bot, lay the demonstration cards R B R R R B in a row. The bot puts down one block for each card, left to right, in a line. You are the tester: hold the goal strip next to the line and touch both, one place at a time. "Red, red. Blue, blue. Red, red. Red... but the goal says blue!" Point to the 4th instruction card: "Here's the bug." Swap it for a blue square card, clear the blocks, and let the bot build again. Check again: it matches.
2. **Groups (11 min).** Give each group set A. The programmer lays the six instruction cards in a row (already in order, with the bug in them). The bot builds. The tester

compares, finds the wrong card and swaps it, and the bot builds again to check. Then rotate badges and do set B, then set C, so each child has each job once.
    - The bot follows the cards even when they look wrong; that is how the tester can find the bug.
    - A child with low vision can be the tester using raised goal strips and cards, comparing by touch: the round blocks and square blocks feel different, so the built line can be checked by hand, one place at a time.
    - If a group finishes early, the programmer makes their own goal strip and cards, with or without a bug, for the others.

Talk about it (4 min)

- "Why did the bot build the wrong line?" (The bot did exactly what the cards said, and one card was wrong.)
- "How did the tester find the bug?" (By comparing the line with the goal, one place at a time.)
- "After the swap, why did the bot build again?" (To check that the fix worked.)

✓ Closing check (3 min)

Show a goal strip R B B R B B and instruction cards R B R R B B. Ask: "Tester, which card is the bug, and what should it be?"

Answer: the 3rd card (a red circle) should be a blue square.

Vocabulary

programmer, bot, tester, instruction, bug, fix, check

Differentiation

- Support: work with an adult as one of the three jobs; use set A only; the tester points to each place while an adult says the colours.
- Stretch: two bugs in one set, or the programmer writes a new goal strip and the tester checks a friend's cards.

⚠ Safety

Use blocks too big to swallow; no beads or small parts. Clear blocks away between turns so the table stays tidy.

Teacher notes

The three roles come from CS Unplugged's Kidbots activities (see the unit's Sources); the card-and-block task is written for this unit. "Bot" means a robot that follows instructions; the

child playing the bot is a person doing a job, not someone being told off. The bug is always in the cards, never in the bot: this is the point to make.

Lesson 5: Pattern show (25 min)

Back to the unit

Goal

Children show a pattern they made and a bug they fixed, and the teacher finds out, one to one, what each child can do.

Materials

- Per pair: 6 red circle cards and 6 blue square cards (raised cards for children who need them).
- The large teacher set and the strip or floor tape.
- For the one-to-one check: 6 red circle and 6 blue square cards, and for part 5 a goal strip showing red, blue, red, blue, four instruction cards (one red circle, three blue squares) and a spare red circle card.
- A class list to note results.

Steps

✎ Opening (3 min)

Clap, stamp, clap, stamp with the class. Say: "This week we were pattern detectives, bug finders and testers. Today we show what we can do."

✎ Main activity: pattern show (15 min)

1. **Make (5 min).** Each pair makes a pattern of at least six cards that repeats at least twice.
2. **Show (7 min).** Pairs take turns to show the class. The class reads the pattern aloud (or claps it: circle = clap, square = stamp) and says what comes next.
3. **Bug hunt (3 min).** While the class closes their eyes, the teacher swaps one card in a pair's pattern. The class finds the bug and the pair fixes it.

Pairs who prefer not to show to the whole class can show the teacher or a small group instead.

Talk about it (4 min)

- "What is a pattern?" (Something that repeats, so you can guess what comes next.)
- "What is a bug?" (A wrong part, like a card that does not fit.)
- "What does a tester do?" (Checks the result against the goal and finds the wrong part.)

✓ Closing check (3 min)

Lay red circle, blue square, red circle, blue square, red circle, ? and ask the class what comes next.

Answer: blue square.

One-to-one check (about 3 minutes per child)

Do this during choosing time or at another time over the following days, not during the 25-minute lesson. Children can answer by speaking, signing, pointing, moving a card or directing an adult.

1. Clap, stamp, clap, stamp; ask the child to keep going. **Looks for:** one more clap, stamp (taps or nods are fine).
2. Lay circle, square, circle, square, circle and ask what comes next. **Answer:** square.
3. Lay circle, square, square, circle, square, square, circle and ask what comes next. **Answer:** square.
4. Lay circle, square, circle, square, circle, circle, circle, square and ask which card does not fit. **Answer:** the 6th card (a circle) should be a square.
5. Show a goal strip of red, blue, red, blue and instruction cards red, blue, blue, blue. Ask the child, as tester, to find the bug and fix it. **Answer:** swap the 3rd card for a red circle card.
6. Ask the child to make their own pattern. **Looks for:** at least six cards with a part that repeats at least twice.

Note which parts each child managed. A child who manages parts 1, 2 and 4 has the main ideas; parts 3, 5 and 6 are next steps. This check is for planning next term, not for a grade.

Vocabulary

pattern, repeat, next, bug, fix, tester

Differentiation

- Support: in the check, use AB patterns only and lay fewer cards; accept pointing.
- Stretch: ask the child to make a broken pattern for the teacher to fix, and to say how they would check the fix.

⚠ Safety

No special safety issues. Keep showing optional in form: showing a small group or the teacher counts.

Teacher notes

Results from part 4 and part 5 are a useful guide for grade 1 work on finding and fixing wrong steps. A child who finds the bug by touch, by sound or by colour alone has still found it.

Unit: Step-by-Step Robot Recipes and Floor-Grid Paths (Computing, Kindergarten)

> **Unreviewed AI draft.** Written by Gemini 4 Argon (2026-11-05). No human teacher has reviewed this unit. All content is CC0 1.0 Public Domain.

Overview

How does a toy robot or a computer know what to do? It cannot guess—a person must give it a clear, ordered list of steps called an **algorithm**.

Across five 30-minute unplugged movement and picture-card lessons, Kindergarten students (`ages 5–6`) learn the foundations of computational thinking through whole-body play. Students put 3-card daily routines into `First → Next → Last` order, guide a "Walking Robot" partner across a 3×3 floor grid using physical arrow cards (`↑ Step Forward`, `↺ Turn Left`, `↻ Turn Right`), notice when a missing or extra arrow makes the robot miss the target (**a bug**), act out simple action patterns (`Clap, Tap, Clap, Tap`), and press paper "buttons" to make a partner perform a matching sound or motion (connecting to Kindergarten Science: Pushes and Pulls).

Prior knowledge

- No reading or writing is required; all activities use picture cards, arrows, and spoken/movement commands.
- Students can count to 5 and follow one-step classroom directions.
- Helpful cross-link: Pushes and pulls (Science, Kindergarten) explores how pushing a button or object causes a motion.
- Leads to: Symbols, signals, sorting, and step-by-step recipes (Computing, Grade 1) and Computing & systems progression.

Learning goals

By the end of this unit, a student can:

1. Arrange 3 picture cards into **`First → Next → Last`** order for a familiar routine and tell what happens if the first and last cards are switched (Lesson 1).
2. Follow a 3-arrow floor-grid path (`↑ Step Forward` one square at a time) without jumping over squares (Lesson 2).
3. Spot the **bug** (mistake) when a 3-arrow path lands on a mud-puddle square instead of the treasure star, and swap in the card that fixes it (Lesson 3).
4. Continue a 2-part repeating action pattern (`Clap, Stomp, Clap, Stomp, ...`) and name the core part that repeats (Lesson 4).
5. Match a button press (**input**) to the action or sound a machine makes (**output**) in a partner game (Lesson 5).

Lesson sequence

Lesson	Title	Time	Core idea
Lesson 1	First, Next, Last: Putting Daily Routines in Order	30 min	Instructions happen one step at a time in order (**sequence**)
Lesson 2	Walking the Floor Grid: One Square at a Time	30 min	A robot follows arrow cards (↑) literally, moving one square per card
Lesson 3	Oops, Wrong Square! Fixing a Bug on the Rug	30 min	When a robot lands in the wrong place, we check the arrow cards and fix the **bug**
Lesson 4	Do It Again! Repeating Action Patterns	30 min	Many dances, songs, and computer tasks repeat a small pattern over and over
Lesson 5	Push a Button, Make It Go: Buttons and Actions	30 min	Pushing a button gives a machine an instruction to start an action or sound

Materials for the whole unit

- Masking tape or painter's tape to mark a 3×3 grid of large squares on the classroom floor (each square big enough for a child to stand inside)
- Large printed or hand-drawn picture cards:
- 3-step routine sets (Washing Hands, Building a Snowman, Getting Ready for Rain)
- 6 large Arrow Cards (`↑ Step Forward`, `↺ Turn toward the Window`, `↻ Turn toward the Door`)
- 1 paper **Gold Star** (goal) and 1 paper **Mud Puddle** (obstacle)
- 3 paper "Button Circles" (Blue Star Button, Yellow Bell Button, Purple Frog Button) and 1 large `[ □ PUSH BUTTON ]` action card (Lesson 5)
- Large tabletop robot tokens (oversized wooden blocks or plush toys at least 5 cm wide for Lesson 3's pair strip; avoid coins, small erasers, or small linking cubes that could pose a choking hazard for 5-year-olds)

Assessment summary

Each lesson finishes with a 3-question oral, thumbs-up/thumbs-down, or physical card-pointing check with worked teacher answers. In Lesson 5, students guide a plush toy or classmate across a 3-square path to press a paper button and trigger the matching class action.

Sources

These references provide teacher background for the concepts introduced in Kindergarten. All links were checked to load on 2026-11-05.

- Sequence (ordered list of elements or steps): https://en.wikipedia.org/wiki/Sequence — background for Lesson 1's `First → Next → Last` ordering.
- Algorithm (step-by-step procedure): https://en.wikipedia.org/wiki/Algorithm — background for Lessons 1–2.
- Grid (spatial index / square floor grid navigation): https://en.wikipedia.org/wiki/Grid_(spatial_index) — background for Lesson 2's 3×3 floor-grid movement.
- Debugging (finding and correcting faults in instructions): https://en.wikipedia.org/wiki/Debugging — background for Lesson 3's arrow-card repair.
- Pattern (repeating regularity in actions or shapes): https://en.wikipedia.org/wiki/Pattern — background for Lesson 4's repeating movement patterns.

- Bell, T., Witten, I. H., & Fellows, M. (2015). CS Unplugged (pedagogical reference for screen-free early computing activities; all lesson text and games in this unit are original CC0 work): https://www.csunplugged.org/

Known gaps and cautions

- **Unreviewed AI draft:** This unit has not yet been reviewed by a human Kindergarten educator or tested with 5-year-old students.
- **Left/Right vs. Classroom Landmarks (Lesson 2):** Many 5-year-olds are still developing automatic left/right recall. In Lessons 2 and 3, pair turn arrows with concrete **classroom wall landmarks** (for example, "Turn toward the Window wall" vs. "Turn toward the Bookshelf wall", or tape a blue circle on the robot's left shoe and a yellow square on the right shoe).
- **Safe spacing on the floor grid:** Have only one "Robot" student stand inside the 3×3 tape grid at a time while the rest of the group sits around the perimeter as "Programmers" and "Bug Checkers" so students do not bump into each other.

Lesson 1: First, Next, Last — Putting Daily Routines in Order

- **Unit:** Step-by-Step Robot Recipes and Floor-Grid Paths (Computing, Kindergarten)
- **Time:** 30 minutes
- **Grouping:** Whole class on the circle rug, then pairs

Learning objective

Students will arrange 3 picture cards into **`First → Next → Last`** order for an everyday routine and explain what happens when the order is mixed up.

Materials

- 3 large demonstration picture cards for **Building a 3-Ball Snowman**:
- **Card A (First):** Roll a big bottom snowball on the ground
- **Card B (Next):** Lift a middle snowball on top of the big one
- **Card C (Last):** Put a small head snowball and carrot nose on top
- Per pair of students: a 3-card picture set for **Getting Ready for Rain**:
- Put on rain boots and raincoat
- Open the umbrella outside
- Walk through the rain staying dry

Vocabulary

- **First:** The step that happens at the very beginning (`Step 1`).
- **Next:** The step that happens in the middle (`Step 2`).
- **Last:** The step that finishes the job (`Step 3`).
- **Recipe (Algorithm):** A list of steps in the right order to do a job.

Lesson flow

✎ 1. Opening: The Upside-Down Snowman (6 min)

1. Hold up the 3 Snowman cards in the **wrong order**:
2. **First:** Put the tiny head snowball in the air (`Card C`)!
3. Pretend to hold a tiny snowball high in the air with nothing underneath it and let it drop ("Plop!").
4. Ask: "Why did my snowman head fall on the ground? Which snowball has to go **First** on the bottom so it can hold up the others?"
5. Students point to the big bottom snowball (`Card A`).

✎ 2. Main activity: Ordering `First → Next → Last` (15 min)

1. **Whole-class Snowman Line-Up (5 min):**
2. Invite 3 volunteers to hold `Card A`, `Card B`, and `Card C` at the front of the rug.

3. The class helps them stand in left-to-right order: `First` **(Big bottom ball)** → `Next` **(Middle ball)** → `Last` **(Head and nose)**.
4. Everyone acts out rolling the big ball, lifting the middle ball, and placing the head and carrot nose.
5. **Partner Rainy-Day Sort (10 min):**
6. Hand each pair the 3 **Getting Ready for Rain** picture cards shuffled face-up.
7. Partners talk together and line their 3 cards up from left to right:
    1. **First:** Put on boots and coat inside.
    2. **Next:** Open the umbrella at the door.
    3. **Last:** Walk in the rain staying dry.
8. Ask pairs to test a silly swap: "What happens if you do Card 3 (Walk in the rain) **First**, and Card 1 (Put on raincoat) **Last**?" (You get soaked before your coat is on!)

✓ 3. Discussion questions (4 min)

- **Question 1:** "When you eat a bowl of cereal, which comes **First**: pouring the cereal and milk into the bowl, or scooping with your spoon?"
- Example good answer: Pouring the cereal and milk into the bowl comes **First**, or else your spoon scoops an empty bowl!
- **Question 2:** "Why do machines and robots need their steps in the right order?"
- Example good answer: Because a robot only does the exact step you tell it, so mixed-up steps make a silly mess.

✓ 4. Closing check (5 min)

Read 3 quick checks; students point or hold up 1, 2, or 3 fingers:

1. Look at these 3 pictures for painting a picture:
2. **(1)** Wash the paint off the brush at the end
3. **(2)** Dip the clean brush into the paint cup
4. **(3)** Paint a bright sun on the paper Which picture happens **First**?
5. Which of those 3 pictures happens **Last**?
6. What happens if you wash the brush and put it away **before** you paint on the paper?

Worked answers for closing check

1. **(2) Dip the clean brush into the paint cup** happens **First**.
2. **(1) Wash the paint off the brush at the end** happens **Last**.
3. The paper stays blank because you put the brush away before painting!

Differentiation

- **Extra support:** Place a 3-box mat labeled with dots (`•`, `••`, `•••`) on the floor so students can place one card in each box from left to right.
- **Extra challenge:** Give pairs a 4th blank card to draw what happens after coming back inside from the rain (Take off wet boots and close the umbrella).

Teacher notes

- **Common misconception:** Some students place cards from right to left or in a vertical pile. Point to the green dot on the left side of the rug/table as the starting spot so everyone reads the sequence in the same direction.

Lesson 2: Walking the Floor Grid — One Square at a Time

- **Unit:** Step-by-Step Robot Recipes and Floor-Grid Paths (Computing, Kindergarten)
- **Time:** 30 minutes
- **Grouping:** Whole class seated around a 3×3 floor tape grid

Learning objective

Students will guide a classmate ("the Walking Robot") from a Start square to a Gold Star square on a 3×3 floor grid by lining up `↑ Step Forward` and `Turn` arrow cards one step at a time.

Materials

- A 3×3 floor grid made with painter's tape on the rug: `[ ★ ] [ ] [ ]` (Row 3: Top — ★ starts at Top-Left for Opening & Path 1; moves to Row 2 Col 2 for Path 2 and Top-Right Row 3 Col 3 for Path 3) `[ ] [ ] [ ]` (Row 2: Middle) `[ R ] [ ] [ ]` (Row 1: Bottom — R = Robot Start at Bottom-Left facing UP ↑)
- 6 large Arrow Cards:
- 4 cards showing `↑ Step Forward 1 Square`
- 2 cards showing `↻ Turn Toward the Door (Right)` / `↺ Turn Toward the Window (Left)`
- 1 paper **Gold Star (★)**

Vocabulary

- **Grid:** Squares lined up in neat rows so we can count exact steps.
- **Command Card:** An arrow card that tells the robot one exact move.
- **Step Forward (↑):** Move 1 square straight ahead in the direction your toes are pointing.
- **Turn In Place (↺ / ↻):** Spin your feet inside the **same square** to face a new wall (without stepping into a new square yet!).

Lesson flow

✎ 1. Opening: How Does a Floor Robot Move? (6 min)

1. Stand inside the bottom-left corner square `[R]` of the 3×3 tape grid with your toes pointing toward the top row. Place the **Gold Star (★)** two squares straight ahead in the top-left corner.
2. Ask: "If I want to walk from my Start square to the Gold Star without stepping outside the lines, how many squares forward do I need to step?"
3. Count together as you step: `1` **step forward** (into the middle-left square), `2` **steps forward** (onto the Gold Star!).
4. Show that **2** `↑ Step Forward` **arrow cards** (`[ ↑ ] [ ↑ ]`) write down that exact recipe!

✎ 2. Main activity: Human Robot & Programmer Teams (15 min)

1. **Practice the "Turn in Place" Rule (5 min):**
2. Have all students stand up on their spot around the rug.
3. Hold up `↑ Step Forward`: everyone takes 1 small step forward (and steps back).
4. Hold up `↻ Turn Toward the Door`: everyone pivots their feet inside their spot so their nose and toes point toward the classroom door—**without** stepping forward!
5. Emphasize: "A turn card only spins our toes inside our square. To move into the next square, we still need a `↑ Step Forward` card!"
6. **3-Challenge Grid Game (10 min):**
7. For each round, choose 1 **Robot** (stands on `[R]` facing North/Up) and 2 **Programmers** (lay the arrow cards in a row beside the grid while the class checks):
    - **Path 1 (Straight Line):** Star is 2 squares straight ahead (`Row 3, Col 1`).
    - Recipe: `[ ↑ ] [ ↑ ]` (2 cards).
    - **Path 2 (One Corner Turn):** Move the Star to `Row 2, Col 2` (1 square up, 1 square toward the door/right).
    - Recipe: `[ ↑ ] [ ↻ Turn Toward Door ] [ ↑ ]` (3 cards).
    - **Path 3 (All the Way to the Top-Right Corner ★):** Star is at `Row 3, Col 3` (2 squares up, 2 squares right).
    - Recipe: `[ ↑ ] [ ↑ ] [ ↻ Turn Toward Door ] [ ↑ ] [ ↑ ]` (5 cards).
8. For each path, the class chants the cards in order while the Robot performs one card per chant.

✓ 3. Discussion questions (4 min)

- **Question 1:** "In Path 2, why did we need THREE cards (`↑`, `↻ Turn`, `↑`) even though the Robot only stepped into TWO new squares?"
- Example good answer: Because one card had to turn the Robot's toes toward the door before the Robot could step sideways into the Star square!
- **Question 2:** "Is there a second road of squares that also reaches the Top-Right Star (`Row 3, Col 3`)?"
- Example good answer: Yes! You could turn toward the door first, walk 2 squares across the bottom row, turn up, and walk 2 squares up!

✓ 4. Closing check (5 min)

Place the Robot on the bottom-left square facing **Up (`↑`)** and place the **Gold Star (`★`)** just **1 square straight ahead** (`Row 2, Col 1`).

1. How many `↑ Step Forward` cards does the Robot need to land right on the Gold Star?
2. What happens if we give the Robot **two** `↑ Step Forward` cards (`[ ↑ ] [ ↑ ]`) instead of one?
3. When the Robot sees a `↻ Turn` card, does the Robot move into a new square or spin its toes inside the same square?

Worked answers for closing check

1. **One card** (1 `↑ Step Forward` card) to reach the Gold Star 1 square ahead.
2. **Two cards (`[ ↑ ] [ ↑ ]`):** The Robot walks **right past** the Gold Star into the top-left square (`Row 3, Col 1`)!
3. **Turn card (↻):** It **spins its toes inside the same square** (it does not step into a new square).

Differentiation

- **Extra support:** Give the Robot a paper steering wheel or tape a big paper arrow onto the top of their shoes so everyone can see which way the Robot is facing before reading the next card.
- **Extra challenge:** Place a plush puppy on one square as a "Sleeping Puppy" obstacle and ask students to build a path to the Star that walks around the puppy without waking it up.

⚠ Safety

- Walk slowly one square at a time inside the tape grid; never run or jump across squares.

Teacher notes

- **Common misconception:** Just like in Grade 3 grid algorithms, young children naturally combine "Turn and step" into one motion. Practicing **"Glue your heels to your square while you turn"** makes the distinction between `Turn` and `Step Forward` crystal clear and fun.

Lesson 3: Oops, Wrong Square! — Fixing a Bug on the Rug

- **Unit:** Step-by-Step Robot Recipes and Floor-Grid Paths (Computing, Kindergarten)
- **Time:** 30 minutes
- **Grouping:** Whole class around the 3×3 floor tape grid, then pairs

Learning objective

Students will watch a Robot follow a broken arrow-card path, point to the exact card that caused the Robot to miss the Gold Star or step in a Mud Puddle (**a bug**), and fix the path.

Materials

- The 3×3 floor tape grid from Lesson 2
- 6 large Arrow Cards (`↑ Step Forward`, `↻ Turn Toward Door`, `↺ Turn Toward Window`)
- 1 paper **Gold Star (★)** and 1 brown paper **Mud Puddle (▯)**
- Per pair (for Mystery 3): a 3-square paper strip (`[ Start ] [ Middle ] [ ★ ]`), 3 paper `[ ↑ ]` cards, and 1 **large wooden block or plush robot token** (at least 5 cm wide; do not use coins or small erasers)

Vocabulary

- **Bug:** A mistake in our arrow cards that sends the Robot to the wrong square.
- **Fix (Debug):** Finding the wrong card and trading it for the right card so the Robot reaches the goal.
- **Test:** Having the Robot walk the cards from the beginning to check if our fix worked.

Lesson flow

✎ 1. Opening: The Mud-Puddle Mistake (6 min)

1. Set up the 3×3 floor grid with:
2. **Robot (R)** at Bottom-Left (`Row 1, Col 1`), facing **Up (↑)**
3. **Gold Star (★)** at Middle-Left (`Row 2, Col 1`, 1 square straight ahead)
4. **Mud Puddle (▯)** at Top-Left (`Row 3, Col 1`, 2 squares straight ahead)
5. Lay down **two** `↑ Step Forward` cards: `[ ↑ ] [ ↑ ]`.
6. Act out the Robot:
7. Card 1 (↑): Step onto the Gold Star (`Row 2, Col 1`)—"Yay!"
8. Card 2 (↑): "Wait, there is still another card! Robots must follow every card!" Step forward off the Star and SPLASH! right into the Mud Puddle (`Row 3, Col 1`)!
9. Ask: "Oh no! Why did the Robot step into the Mud Puddle? Which card was the **bug**?"
10. Students point to the **extra second ↑ card**! Take away the second ↑ card and **test** again—now the Robot stops safely on the Gold Star.

✎ 2. Main activity: Three Bug-Fix Mysteries (15 min)

Invite student **Bug Detectives** to solve three broken programs on the rug:

1. **Mystery 1 — The Too-Short Recipe (Missing Card Bug):**
2. **Goal (★):** Top-Left (`Row 3, Col 1`, 2 squares straight ahead from Start).
3. **Broken Cards on Rug:** `[ ↑ ]` (only 1 card!).
4. **Test:** Robot takes 1 step to the middle square and stops one square short of the Star.
5. **How to Fix:** Add one more **`[ ↑ ]`** card at the end (`[ ↑ ] [ ↑ ]`) and re-test!
6. **Mystery 2 — The Wrong-Way Turn Bug:**
7. **Robot (R):** Starts in the **Bottom-Middle square (`Row 1, Col 2`)**, facing **Up (↑)**.
8. **Goal (★):** Middle-Right square (`Row 2, Col 3`, toward the Door).
9. **Mud Puddle (▯):** Middle-Left square (`Row 2, Col 1`, toward the Window).
10. **Broken Cards on Rug:** `[ ↑ ] [ ↺ Turn Toward Window ] [ ↑ ]`.
11. **Test:** Robot steps up to the center square, turns toward the Window, and steps into the Mud Puddle!
12. **How to Fix:** Which card was the bug? **Card 2 (`↺ Turn Toward Window`)!** Swap Card 2 for **`[ ↻ Turn Toward Door ]`**, then re-test from the start to watch the Robot land on the Gold Star!
13. **Mystery 3 — Pair Tabletop Bug Fix (5 min):**
14. Each pair gets a small 3-square strip drawn on paper (`[ Start ] [ Middle ] [ ★ ]`) and 3 paper arrow cards placed on the side: `[ ↑ ] [ ↑ ] [ ↑ ]`.
15. Partners slide their large wooden block or plush robot token along the 3 squares, see that 3 steps goes off the end of the paper strip, and remove 1 `↑` card so their token lands right on `[ ★ ]`.

✓ 3. Discussion questions (4 min)

- **Question 1:** "When our Robot stepped in the Mud Puddle, was the Robot being naughty, or was it just following our cards?"
- Example good answer: The Robot was doing its job following the cards! The mistake (**bug**) was in the cards, not the Robot.
- **Question 2:** "Why do we always walk the path again from the Start square after we change a card?"
- Example good answer: To **test** our fix and make sure the new cards really reach the Gold Star!

✓ 4. Closing check (5 min)

1. A Robot needs to move **2 squares forward** to reach the Gold Star (`[ Start ] → [ Square ] → [ ★ ]`), but the programmer only put down **1 `↑` card**. Where will the Robot stop, and how do you fix the bug?
2. What do computer scientists call a mistake in a list of instructions?
3. After you fix a bug in your arrow cards, what should you do next to be sure your fix works?

Worked answers for closing check

1. The Robot stops **1 square too early** (on the middle square). Fix it by **adding one more `↑ Step Forward` card** (`[ ↑ ] [ ↑ ]`).
2. A **bug**.
3. **Test it** by having the Robot walk the arrow cards from the Start square.

Differentiation

- **Extra support:** Touch each arrow card with a pointer stick at the exact moment the Robot takes that step so students see the 1-to-1 match between the card on the floor and the Robot's footstep.
- **Extra challenge:** Give pairs a path with **two** bugs (wrong turn AND one missing forward step) to fix one bug at a time.

⚠ Safety

- Use only **large wooden blocks, plush toys, or large paper cards** (at least 5 cm across) as tabletop robot tokens; never use coins, beads, or small erasers that could be a choking hazard for 5-year-olds.
- Keep only one student walking inside the 3×3 floor tape grid at a time while classmates sit around the outside edge.

Teacher notes

- **Building a warm debugging culture:** Celebrate bugs with a cheerful class phrase like "Hooray, we found a bug! Now we can fix it!" so 5-year-olds view mistakes as normal, fixable puzzles rather than personal failures.

Lesson 4: Do It Again! — Repeating Action Patterns

- **Unit:** Step-by-Step Robot Recipes and Floor-Grid Paths (Computing, Kindergarten)
- **Time:** 30 minutes
- **Grouping:** Whole class on the rug, then pairs

Learning objective

Students will identify the repeating unit in a 2-step or 3-step movement/picture **pattern**, perform a pattern a counted number of times (**a loop**), and say what action comes next.

Materials

- Picture cards for action moves:
- **Clap Hands (👏)**
- **Pat Knees (🙌)**
- **Touch Head (🗣️)**
- **Jump Up (⬆️)**
- Number cards `2` and `3` (to show "Do this pattern 2 times" or "3 times")

Vocabulary

- **Pattern:** Steps or pictures that repeat in the exact same order over and over.
- **Repeat (Loop):** Doing the same small group of steps again and again.

Lesson flow

✎ 1. Opening: The Echo Dance (6 min)

1. Without explaining first, perform a steady, slow movement pattern while saying the words: `Clap – Pat Knees – Clap – Pat Knees – Clap – Pat Knees ...`
2. Pause with your hands in the air after `Clap` and ask: "What move comes next?" (`Pat Knees`!)
3. Tape the 6 picture cards on the board: `[ 👏 Clap ] [ 🙌 Pat ] [ 👏 Clap ] [ 🙌 Pat ] [ 👏 Clap ] [ 🙌 Pat ]`
4. Ask: "Look at this long line of 6 cards! Which two cards keep happening over and over?"
5. Draw a big circle around the first two cards **`( [ 👏 Clap ] [ 🙌 Pat ] )`** and put the number **`3`** beside the circle: "Do *`Clap, Pat`* **3 times**!"

✎ 2. Main activity: Shrinking Long Recipes with "Repeat" (15 min)

1. **Counted Loops with Our Fingers (5 min):**
2. Show how we can do a 2-card pattern **`3 times`**:
    - Pattern inside the circle: **`( [ 🗣️ Touch Head ] → [ 👏 Clap ] )`**
    - Number card outside: **`3 times`**

3. One student stands by the board as the **Counter**, holding up 1 finger for Round 1 (`Touch Head, Clap`), 2 fingers for Round 2 (`Touch Head, Clap`), and 3 fingers for Round 3 (`Touch Head, Clap` — "STOP!").
4. **Grid Walk Connection (4 min):**
5. Look back at our floor tape grid: if a Robot wants to walk **3 squares straight ahead**, instead of saying "Step forward, Step forward, Step forward", we can circle **one** `[ ↑ Step Forward ]` card and put the number `3` next to it ("Step forward 3 times").
6. **Partner Dance-Loop Inventors (6 min):**
7. Each pair picks **2 action cards** (for example, `[ Pat Knees ]` and `[ Jump Up ]`) and a number card (`2` or `3`).
8. Partner A points to the cards and counts the rounds (`1... 2... 3!`) while Partner B acts out the repeating robot dance, then they switch!

✓ 3. Discussion questions (4 min)

- **Question 1:** "Why is it easier to say 'Clap and Pat 3 times' than to tape 6 separate cards across the whole wall?"
- Example good answer: Because it uses fewer cards and takes less space while telling the robot the exact same dance!
- **Question 2:** "Where else in our day do we do the same small pattern over and over?"
- Example good answer: Walking (`Left foot, Right foot, Left foot, Right foot`), brushing teeth (`Up, Down, Up, Down`), or singing the chorus of a song!

✓ 4. Closing check (5 min)

1. Look at this action pattern: **`Clap – Stomp – Clap – Stomp – Clap – ?`** What action comes next?
2. If our circled pattern is **`( Touch Nose → Wave Hand )`** and the number card says **`2 times`**, how many times will you wave your hand in total?
3. What does the word **repeat** mean?

Worked answers for closing check

1. **`Stomp`** comes next.
2. **2 times** (`Touch Nose, Wave Hand` [1], `Touch Nose, Wave Hand` [2]).
3. To do the same step or group of steps **again**.

Differentiation

- **Extra support:** Use **large two-color paper cards or oversized wooden blocks** (`Red, Blue, Red, Blue, Red, Blue`, at least 5 cm wide; avoid small linking cubes or beads that could be a choking hazard) that students can group into three matching `(Red-Blue)` pairs to see and hold the repeating unit.
- **Extra challenge:** Try a **3-step pattern** (`Clap → Pat Knees → Jump`) repeated `2 times` (6 moves total).

⚠ Safety

- Ensure students have an arm's-length of clear floor space around them before doing standing movement patterns (`Pat Knees`, `Jump Up`), and offer seated hand/card alternatives for any student who prefers not to jump.
- Use only large paper cards or oversized wooden blocks (`≥ 5 cm`); do not use small linking cubes or beads with 5-year-olds.

Teacher notes

- **Connection to Grade 2 and Grade 3:** This physical "Circle the pattern + Number card" notation prepares students directly for Grade 2 repeating pattern blocks and Grade 3 `REPEAT n TIMES` loops.

Lesson 5: Push a Button, Make It Go — Buttons and Actions

- **Unit:** Step-by-Step Robot Recipes and Floor-Grid Paths (Computing, Kindergarten)
- **Time:** 30 minutes
- **Grouping:** Whole class and pairs

Learning objective

Students will connect a physical **button push** (input) to the specific **sound or action** (output) a machine performs, and combine a 3-square floor-grid walk with a button press in the unit finale.

Materials

- 3 large paper "Buttons" taped to the board or easel:
- **Blue Star Button (⋆):** Makes the machine do **2 Star Jumps**
- **Yellow Bell Button (□):** Makes the machine sing **"Ding-Dong!"**
- **Purple Frog Button (□):** Makes the machine say **"Ribbit!" and crouch low**
- The 3×3 floor tape grid and Arrow Cards (↑, ↺, ↻) from Lessons 2–3
- 1 large `[ □ PUSH BUTTON ]` command card (tells the Robot to press the paper button on its square)

Vocabulary

- **Button (Input):** Something you push or tap to tell a machine what to do.
- **Action / Sound (Output):** What the machine does, shows, or plays after you push the button.

Lesson flow

✎ 1. Opening: Pushes That Send a Message (6 min)

1. Connect to Kindergarten Science: Pushes and Pulls:
2. In Science, we learned that a **push** moves an object away from you—like pushing a toy car across the floor.
3. Ask: "What happens when you **push** the doorbell button beside someone's front door? Does the house roll down the street, or does something else happen?"
4. Students answer: It rings a bell ("Ding-dong!") inside the house!
5. Explain: On machines and computers, pushing a **button** sends a message telling the machine which **action or sound** to do!

✎ 2. Main activity: The Three-Button Machine & Grid Finale (15 min)

1. **Whole-Class Sound & Action Machine (6 min):**
2. Point to the 3 paper buttons on the board (**Blue Star ⋆**, **Yellow Bell □**, **Purple Frog □**).

3. Tell the class: "All of you are the Classroom Computer! Do NOT make a sound until a human finger pushes one of your buttons!"
4. Invite a student up to gently press:
    - **Yellow Bell Button (🔔)** → Class sings "Ding-dong!"
    - **Purple Frog Button (🐸)** → Class says "Ribbit!" and crouches
    - **Blue Star Button (★)** → Class does 2 Star Jumps
5. Ask: "Why didn't the computer say 'Ribbit' when I pushed the Yellow Bell button?" (Because each button has its own matching rule!)
6. **Partner Button Pads (4 min):**
7. Pairs draw 2 buttons on a sheet of paper (for example, a **Cat Button** and a **Drum Button**).
8. Partner A taps one paper button; Partner B acts out the matching sound or move, then they switch.
9. **Unit Finale — Walk the Grid and Push the Button! (5 min):**
10. Place the **Yellow Bell Button (🔔)** on one corner of the 3×3 floor grid and the **Purple Frog Button (🐸)** on another corner.
11. Show the new `[ 🔔 PUSH BUTTON ]` command card, then together lay down the cards (`[ ↑ ]` `[ ↑ ]` `[ 🔔 PUSH BUTTON ]`) to guide the Robot 2 squares forward onto a button and press it. When the Robot reads `[ 🔔 PUSH BUTTON ]` and taps the button on its square, the whole class performs that button's sound!

✓ 3. Discussion questions (4 min)

- **Question 1:** "Name one real button you push at home or school, and tell what happens right after you push it."
- Example good answer: Pushing the wall switch turns on the classroom lights; pushing the crosswalk button makes the walk sign light up; pushing the water fountain button makes water bubble up.
- **Question 2:** "What did we learn across all five of our Robot Recipe lessons?"
- Example good answer: We learned to put steps in `First, Next, Last` order (Lesson 1), follow arrow cards on a grid (Lesson 2), fix **bugs** when we miss the star (Lesson 3), **repeat** patterns (Lesson 4), and push **buttons** to make actions happen (Lesson 5)!

✓ 4. Closing check (5 min)

1. In our classroom game, **Yellow Bell Button = "Ding-Dong!"** and **Purple Frog Button = "Ribbit!"**. If a student pushes the **Yellow Bell Button**, what sound comes out?
2. Put these 3 steps for using a toy music box in `First → Next → Last` order:
3. **(A)** Listen to the music play
4. **(B)** Walk over to the table where the music box sits
5. **(C)** Push the green Start button on top of the box
6. On our floor grid, the Robot is 1 square away from the Blue Star Button, facing straight at it. Which card does the Robot need to reach the button square: `↑ Step Forward` or `↻ Turn`?

Worked answers for closing check

1. **"Ding-Dong!"**
2. **First: (B)** Walk over to the table → **Next: (C)** Push the green Start button → **Last: (A)** Listen to the music play.
3. `↑ Step Forward`.

Differentiation

- **Extra support:** Use real tactile props (such as a tap-light puck or three upside-down paper cups of different colors/shapes) for students to press.
- **Extra challenge:** Add a **"Power Switch (ON / OFF)"** paper card next to the 3 buttons: if the Power Switch is turned to `OFF`, pressing any of the 3 buttons produces **silence** until a programmer flips the Power Switch back to `ON`!

Teacher notes

- **Preparing for Grade 1 and Grade 3:** This lesson plants the seed for Grade 1's two-state signals (`ON / OFF`) and Grade 3's formal **Input → Processing → Output** model through playful, concrete cause-and-effect.

Unit: Symbols, Signals, Sorting, and Step-by-Step Recipes (Computing, Grade 1)

> **Unreviewed AI draft.** Written by Gemini 4 Argon (2026-11-05). No human teacher has reviewed this unit. All content is CC0 1.0 Public Domain.

Overview

Before young children touch a keyboard, they already use the core building blocks of computing every day: they follow **step-by-step recipes** to get ready for school, read **symbols** on classroom labels, answer **yes/no questions** with two-state **signals** (thumbs up or thumbs down, red or green cards), and **sort** objects into groups using a clear rule.

Across five 35-minute unplugged lessons, Grade 1 students explore how humans and machines represent and organize information without needing any screens or electricity. Students act out step-by-step morning routines and spot out-of-order steps, invent picture symbols to label classroom bins, send silent two-state signals across the rug using two-color cards and flashlights, sort buttons and leaf cards by a single attribute rule (cross-linked with Grade 1 Science's Plants: parts, needs and growth), and distinguish tools that store or send information from tools that only do physical work.

Prior knowledge

- Students can count to 10, recognize basic shapes (circle, square, triangle) and colors, and take turns in a partner game.
- Helpful cross-links:
- Blending and decoding (Language and literacy, Grade 1) explores how written letter symbols stand for spoken sounds.
- Plants: parts, needs and growth (Science, Grade 1) observes leaf shapes and plant parts, which Lesson 4 uses as an unplugged sorting set.
- Leads to: Inputs, outputs, loops, and debugging (Computing, Grade 3) and Computing & systems progression.

Learning goals

By the end of this unit, a student can:

1. Put a 4-card everyday routine in exact order, act it out literally, and explain what goes wrong when two steps are swapped (Lesson 1).
2. Match and draw a simple **symbol** (icon) that stands for a classroom object or action, and explain why everyone using the symbol must agree on its meaning (Lesson 2).
3. Answer a **yes/no question** across the room using a two-state **signal** (such as red/green cards or 1 clap / 2 claps) without speaking aloud (Lesson 3).
4. **Sort** a mixed collection of 8 to 10 items into two groups using a single clear rule (attribute), and guess a partner's secret sorting rule by inspecting their groups (Lesson 4).
5. Decide whether an everyday object is an **information tool** (stores, shows, or sends messages) or a **non-information tool**, and state what kind of information it handles (Lesson 5).

Lesson sequence

Lesson	Title	Time	Core idea
Lesson 1	First This, Then That: Step-by-Step Recipes and Order	35 min	The order of steps in a routine (**algorithm**) matters; swapping two steps changes the result
Lesson 2	Pictures That Mean Something: Symbols and Shared Codes	35 min	A **symbol** is a simple mark or picture that stands for an idea or action when people agree on the rule
Lesson 3	Red Card, Green Card: Two-State Signals for Yes and No	35 min	A two-state **signal** (on/off, green/red, up/down) can send a clear **yes/no** message across a distance
Lesson 4	One Rule at a Time: Sorting Objects by Attributes	35 min	Computers and people organize data by checking one **attribute** (color, shape, size, or hole count) at a time
Lesson 5	Tools That Hold or Send Messages: Information Around Us	35 min	Some tools store, change, or send **information** (words, numbers, pictures, sounds), while others only move or hold physical things

Materials for the whole unit

No computers, tablets, or batteries are required (one optional teacher flashlight in Lesson 3):

- Paper, crayons or markers, and 4–8 index cards per student pair
- One two-sided card per student (colored green on one side and red with a black stripe on the other so color-blind students can distinguish both sides by pattern as well as color)
- Sorting sets for pairs in Lesson 4: an envelope of 10 large pre-made paper cutouts (`at least 5 cm across`, with 6 four-dot pieces and 4 two-dot pieces across Circle, Square, and Leaf shapes in Blue and Yellow) plus 2 extra test cutouts per pair

Assessment summary

Lessons 1–4 each conclude with a 3-question oral, pointing, or picture-card exit check with worked teacher answers. Lesson 5 concludes with a 4-question end-of-unit synthesis check (covering step order, symbols and two-state signals, single-attribute sorting, and information tools) with worked teacher answers.

Sources

These references provide background on the concepts introduced in Grade 1. All links were checked to load on 2026-11-05.

- Algorithm (step-by-step procedure): https://en.wikipedia.org/wiki/Algorithm — background for Lesson 1's ordered routine cards.
- Symbol (mark or sign representing an idea or object): https://en.wikipedia.org/wiki/Symbol — background for Lesson 2's classroom icon design and shared meaning.
- Traffic light (standard color and position signals): https://en.wikipedia.org/wiki/Traffic_light — real-world example of agreed-upon visual signals in Lessons 2 and 3.
- Signal (conveying information through a changing physical state): https://en.wikipedia.org/wiki/Signal — background for Lesson 3's two-state yes/no card and clap signals.
- Sorting (arranging or grouping items systematically by a criterion): https://en.wikipedia.org/wiki/Sorting — background for Lesson 4's single-attribute classification activities.

- Bell, T., Witten, I. H., & Fellows, M. (2015). CS Unplugged (general pedagogical inspiration for teaching early computing concepts without screens; all text, card sets, and activities in this unit are original CC0 work): https://www.csunplugged.org/

Known gaps and cautions

- **Unreviewed AI draft:** This unit has not yet been reviewed by a human Grade 1 teacher or tested in a live classroom.
- **Color vision accessibility in Lesson 3:** Never rely on red vs. green color alone for two-state signal cards. Always draw a bold `✓` (checkmark) on the green side and a bold `×` (cross) or diagonal stripes on the red side so every student can distinguish the two states by shape and pattern.
- **Single-attribute vs. multi-attribute sorting:** Grade 1 Lesson 4 focuses strictly on sorting by **one attribute at a time** (`Has 4 holes` vs. `Does NOT have 4 holes`). Compound rules (`Red AND Round`) are intentionally deferred to Grades 3–6 so 6-year-old learners build confidence with clean two-bucket classification first.
- **Materials safety caution:** Use large paper shape cutouts (`at least 5 cm across`) for Lesson 4's sorting sets rather than small buttons or beads that could pose a choking or small-parts hazard, and remind students that sorting pieces stay flat on the table.

Lesson 1: First This, Then That — Step-by-Step Recipes and Order

- **Unit:** Symbols, Signals, Sorting, and Step-by-Step Recipes (Computing, Grade 1)
- **Time:** 35 minutes
- **Grouping:** Whole class on the rug, then pairs at tables

Learning objective

Students will arrange a 4-step everyday routine in exact start-to-finish order, act out the steps literally, and explain why swapping two steps causes a mistake (**bug**).

Materials

- 4 large teacher demonstration cards showing pictures and short labels for **Putting On Shoes and Socks**:
- `Put on left sock and right sock`
- `Slide feet into shoes`
- `Pull laces or straps tight`
- `Stand up and walk to the door`
- Per student pair: 4 blank index cards and crayons (or a pre-drawn 4-card picture set for students who prefer ordering/pointing rather than drawing) for a 4-step routine (such as Washing Hands, Planting a Seed, or Making a Sandwich)

Vocabulary

- **Step:** One single action in a longer task.
- **Order (Sequence):** Which step happens first, second, third, and last.
- **Algorithm:** A step-by-step list of instructions to finish a job.
- **Bug:** A mistake in the instructions (like a missing step or two steps in the wrong order) that stops the job from working right.

Lesson flow

✎ 1. Opening: The Silly Shoe Robot (7 min)

1. Tell the class: "Pretend I am a helper robot getting ready for recess. Robots only do the exact steps on their cards, in the exact order the cards are lined up!"
2. Tape the 4 teacher cards to the board in a **scrambled** order:
3. Card 1: `Slide feet into shoes`
4. Card 2: `Pull laces or straps tight`
5. Card 3: `Put on left sock and right sock`
6. Card 4: `Stand up and walk to the door`
7. Pantomime following the scrambled cards literally—pretend to tie your shoes first, and then try to pull socks over the outside of your shoes!
8. Ask: "Why did the robot put socks on the outside of its shoes? Did I have all the right steps, or was something else wrong?"

9. Guide students to notice that all 4 steps were good steps, but the **order** was mixed up.

✎ 2. Main activity: Fixing the Sequence & Making 4-Card Recipes (18 min)

1. **Whole-class repair (5 min):** Invite two volunteers to come up and rearrange the 4 shoe cards into a working **algorithm** (`Socks` → `Shoes` → `Straps` → `Walk`). Act it out again to confirm it works.
2. **Pair card creation (8 min):** Give each pair 4 blank index cards. Each pair chooses one familiar 4-step routine:
3. Option A — Washing Hands (aligned with CDC Clean Hands steps): (1) Wet hands with clean running water and turn off the tap, (2) Apply soap and scrub hands all over for at least 20 seconds, (3) Turn on the tap, rinse soap bubbles off under clean running water, and turn off the tap, (4) Dry hands with a clean towel.
4. Option B — Planting a Bean Seed: (1) Scoop soil into the cup, (2) Poke a small hole and drop the seed in, (3) Cover the seed with soil, (4) Pour a little water on top.
5. Students draw a quick picture on each card and write the numbers `1`, `2`, `3`, `4` on the **back** of the cards.
6. **The Partner Swap & Bug Hunt (5 min):**
7. Each pair turns their 4 cards number-side down, mixes up **two** of the cards on purpose, and trades with a neighboring pair.
8. The neighboring pair tests the mixed-up routine (by acting it out, describing what would happen aloud, or tracing the cards in order), points to the two swapped cards (**the bug**), and puts them back in the right order.

✓ 3. Discussion questions (5 min)

- **Question 1:** "In our hand-washing recipe, what happens if we do Step 4 (Dry hands with a towel) BEFORE Step 3 (Rinse soap bubbles off)?"
- Example good answer: All the sticky soap bubbles get wiped onto the dry towel instead of going down the sink!
- **Question 2:** "Are there ever two steps where the order does NOT matter?"
- Example good answer: Putting on your left sock first or your right sock first both work, as long as both socks go on before the shoes!

✓ 4. Closing check (5 min)

Read three quick questions aloud (for Question 1, students can hold up 1, 2, or 3 fingers for A, B, or C or point to the choice; for Questions 2 and 3, students answer aloud, point to vocabulary cards on the board, or tell a partner):

1. Here are 3 steps for brushing teeth: **(A)** Brush teeth up and down, **(B)** Squeeze toothpaste onto the brush, **(C)** Rinse mouth with water. Which step must go **first**?
2. What do computer scientists call a step-by-step recipe for finishing a job?
3. If a robot puts the hat on its feet and the shoes on its head because the instruction cards were mixed up, what do we call that mistake in the instructions?

Worked answers for closing check

1. **(B) Squeeze toothpaste onto the brush** goes first (then **A** brush, then **C** rinse).
2. An **algorithm**.
3. A **bug**.

Differentiation

- **Extra support:** Start with a 3-card routine (First / Next / Last) using pre-drawn picture cards so students can focus on ordering rather than drawing.
- **Extra challenge:** Ask pairs to add a 5th step to their routine and find two steps whose order can be swapped without breaking the result, plus two steps whose order cannot be swapped.

Teacher notes

- **Common misconception:** Young learners often skip steps they consider "obvious" (for example, drawing Eat the banana without Peel the banana). When acting out student cards as the "Classroom Robot," playfully pretend to bite an unpeeled imaginary banana so students see why machines need every step stated clearly.

Lesson 2: Pictures That Mean Something — Symbols and Shared Codes

- **Unit:** Symbols, Signals, Sorting, and Step-by-Step Recipes (Computing, Grade 1)
- **Time:** 35 minutes
- **Grouping:** Whole class and pairs

Learning objective

Students will match familiar **symbols** (icons) to their meanings, design a clear symbol for a classroom supply bin or action, and test whether a classmate can decode it without words.

Materials

- Board or chart paper showing 5 familiar symbols drawn large:
- A **heart** (♥), a **musical note** (♪), a **trash can**, a **magnifying glass**, and a **triangle play button** (▶)
- 2 blank index cards and markers per student (plus optional pre-drawn symbol stickers/stencils and a printed 6-item classroom action menu so students can also participate by selecting or pointing)

Vocabulary

- **Symbol (Icon):** A simple picture or mark that stands for a word, object, or action.
- **Decode:** To look at a symbol or secret code and figure out what message it means.
- **Shared Rule (Key):** The agreement between the sender and the reader about what each symbol means.

Lesson flow

✎ 1. Opening: Reading Without Letters (7 min)

1. Point to the 5 symbols on the board (♥, ♪, trash can, magnifying glass, ▶).
2. Ask: "There are zero letters written inside these five drawings, yet you already know what they mean! Turn to your partner and tell them what the triangle ▶ and the magnifying glass tell you to do on a screen or player."
3. Expected answers: ▶ means **Play / Start**, and the magnifying glass means **Search / Look closely**.
4. Connect to Grade 1 Language and Literacy (Blending and Decoding): just as the letter symbol `m` stands for the `/m/` sound because readers agree on that rule, picture **symbols** let people and computers share instructions quickly in a tiny space.

✎ 2. Main activity: Classroom Symbol Designers & Silent Decoding (18 min)

1. **What makes a good symbol? (5 min):**

2. Draw two versions of a symbol for the **"Scissors Bin"** on the board:
    - Drawing A: A giant detailed picture of a whole classroom with tables, windows, a clock, and tiny scissors in the corner (takes 2 minutes to draw, hard to tell what matters).
    - Drawing B: Two crossed blades and two finger loops (simple, bold, takes 5 seconds to draw and recognize from across the room).
3. Emphasize: A strong symbol is **simple**, **clear**, and shows only the most important shape.
4. **Design 2 Action Symbols (8 min):**
5. Give each student 2 index cards. On the **back** of each card, the teacher (or student) whispers/marks one secret classroom action from a 6-item picture menu on the board:
    1. Wash hands (water drop + hand)
    2. Quiet voices (finger over lips or crossed-out speaker)
    3. Line up at the door (three stick figures in a row by a rectangle door)
    4. Sharpen pencil (pencil tip with shavings)
    5. Put book in backpack (rectangle book + arrow into bag)
    6. Sit in a circle (circle of dots or arrows)
6. On the **front** of each card, students draw a simple symbol using **no letters or numbers**.
7. **Silent Partner Decode Test (5 min):**
8. Partners trade cards front-side up. The partner **decodes** the symbol by pointing to the matching action on the 6-item picture menu, stating it quietly, or acting it out. If the partner identifies the right job, the symbol passed the test!

✓ 3. Discussion questions (5 min)

- **Question 1:** "What would happen if I decided that a green circle on our classroom door meant 'Come inside,' but the principal thought a green circle meant 'Stay out'?"
- Example good answer: People would get confused and do the wrong thing because they didn't share the same rule (key) for the symbol.
- **Question 2:** "Why do computer buttons and road signs often use simple pictures instead of long sentences?"
- Example good answer: You can understand a simple picture in one glance, even from far away or if you are still learning to read long words.

✓ 4. Closing check (5 min)

Students answer three quick checks:

1. A button on an audio player shows **two vertical bars (⏸)** next to the triangle play button (▶). What action does that symbol stand for?
2. You want to put a symbol on the classroom **Recycling Bin for Paper**. Which is a clearer symbol: a drawing of a whole school building, or a drawing of a sheet of paper with arrows going in a circle? Why?
3. What does it mean to **decode** a symbol?

Worked answers for closing check

1. **Pause / Stop for a moment.**
2. **A sheet of paper with arrows in a circle**, because it is simple and shows the exact item and action without extra confusing details.
3. To look at the symbol and translate it back into the word, idea, or action it stands for.

Differentiation

- **Extra support:** Provide stencil shapes (circles, triangles, arrows) so students can combine simple shapes into symbols without worrying about freehand drawing.
- **Extra challenge:** Ask pairs to combine **two** symbols in sequence on one strip (for example, `[Pencil Symbol] → [Backpack Symbol]`) to write a 2-step symbol sentence ("Put your pencil in your backpack").

Teacher notes

- **Common misconception:** Students sometimes treat drawing a symbol as an art contest and add rainbows, grass, and smiling suns around the icon. Remind them that extra decorations can confuse the reader ("Wait—does this bin hold pencils, or does it hold sunglasses?").

Lesson 3: Red Card, Green Card — Two-State Signals for Yes and No

- **Unit:** Symbols, Signals, Sorting, and Step-by-Step Recipes (Computing, Grade 1)
- **Time:** 35 minutes
- **Grouping:** Whole class and pairs across tables

Learning objective

Students will send and receive **two-state signals** (`YES` / `NO`) without speech across the classroom to answer questions and play a mystery guessing game, and explain how two distinct states (visual, tactile, or sound) can carry information without spoken words.

Materials

- Per student: one two-sided **Signal Card** (with an optional raised tactile sticker/felt dot on the `YES` side, or paired with a sound/touch signal such as `1 tap = YES` / `2 taps = NO` for non-visual access):
- Side 1: **Green with a bold black `✓` (checkmark)** — means `YES / TRUE`
- Side 2: **Red with bold black stripes and an `×`** — means `NO / FALSE`
- 4 large **Mystery Object picture cards** for the board (`1. Red Apple`, `2. Yellow School Bus`, `3. Green Frog`, `4. Red Fire Truck`)
- Optional teacher prop: a classroom flashlight or desk lamp (`ON` = Yes, `OFF` = No)

Vocabulary

- **Signal:** A sound, light, or movement that sends a message from one place to another.
- **Two-State Signal:** A signal that only has two clear choices—such as `ON` or `OFF`, `GREEN ✓` or `RED ×`, or `1 clap` vs. `2 claps`.
- **Sender and Receiver:** The **sender** makes the signal; the **receiver** watches or listens to read the message.

Lesson flow

✎ 1. Opening: Talking Across a Noisy River (7 min)

1. Ask students to imagine standing on opposite sides of a rushing waterfall where nobody can hear a spoken voice.
2. Ask: "If your friend across the waterfall holds up an apple and wants to ask 'Are you hungry?', how could you answer **YES** or **NO** without shouting a single word?"
3. Collect student ideas: nodding vs. shaking head, thumbs up vs. thumbs down, holding up a green `✓` card vs. a red `×` card, or flashing a flashlight `ON` vs. keeping it `OFF`.
4. Explain: Computers inside wires cannot shout words either! Instead, they send tiny electrical or light **signals** that have just **two states** (`ON` and `OFF`), which prepares students for how computers use binary in later grades.

✎ 2. Main activity: Silent Signal Relay & Mystery Animal Game (18 min)

1. **Whole-class Signal Check (5 min):**
2. Hand out the two-sided **Green** ✓ **/ Red** × **Signal Cards**.
3. Practice signaling without speech: when the teacher asks a question, nobody speaks—every student holds up the card side (or uses the agreed tap signal) that matches the truth:
    - "Does a triangle have 3 corners?" → Class holds up **Green** ✓.
    - "Do fish live in trees?" → Class holds up **Red** ×.
    - "Are we in Grade 1?" → Class holds up **Green** ✓.
4. **Changing the Physical Signal (4 min):**
5. Put the cards flat on the floor. Now send the **exact same information** using a sound signal:
    - **1 clap** = `YES`
    - **2 claps** = `NO`
6. Ask two more questions (`"Is 2 + 2 equal to 4?"` → 1 clap; `"Is ice hot?"` → 2 claps). Point out that the **message** (`YES` or `NO`) stayed the same even when the **signal tool** changed from colors to claps!
7. **Partner 4-Question Mystery Object Game (9 min):**
8. Place 4 picture cards on the board where everyone can see them:
    1. **Red Apple** (Round, Red, Fruit, No legs)
    2. **Yellow School Bus** (Rectangle, Yellow, Has wheels, No legs)
    3. **Green Frog** (Green, Living animal, Has legs, No wheels)
    4. **Red Fire Truck** (Rectangle, Red, Has wheels, No legs)
9. In pairs sitting across a table:
    - **Sender** secretly picks one of the 4 pictures and writes its number (`1`, `2`, `3`, or `4`) face-down.
    - **Receiver** asks up to **4 yes/no questions** (`"Is your mystery item red?"`, `"Does it have wheels?"`), and the **Sender** answers using their agreed two-state signal (holding up the **Green** ✓ **/ Red** × **Signal Card**, using a tactile card side, or tapping `1 tap = YES` / `2 taps = NO`).
    - Then partners switch roles.

✓ 3. Discussion questions (5 min)

- **Question 1:** "Why did we put a black checkmark ✓ and black stripes ▨ on our cards in addition to green and red?"
- Example good answer: Because color alone can be hard to tell apart in dim light, from far away, or for people with color-blind vision; adding distinct shapes (✓ vs. ×), tactile stickers, or sound signals (`1 clap` vs. `2 claps`) makes the signal much clearer and more accessible.
- **Question 2:** "What happens if a sender holds their card sideways so the receiver can't see either side?"
- Example good answer: The message is unclear! A two-state signal only works when you clearly choose one of the two states.

✓ 4. Closing check (5 min)

1. You and your partner agree on this rule for a flashlight: **Light ON = Yes**, **Light OFF = No**. On a Tuesday morning at school, your partner asks, "Is today a school day?" What should you do with the flashlight?
2. Can a two-state (`YES / NO`) card directly answer the question "What is your favorite color?" Why or why not? How could you change the question so the card can answer it?
3. Name two different ways to send a two-state (`YES / NO`) signal without speaking.

Worked answers for closing check

1. **Turn the flashlight ON** (since Tuesday at school is a school day, and `ON` stands for `YES`; if asked on a weekend day with no school, you would leave the flashlight **OFF** for `NO`).
2. **No**, because "What is your favorite color?" has many possible answers, not just yes or no. You can change it into a yes/no question like "Is your favorite color blue?"
3. Any two clear pairs: **Green ✓ / Red × cards**, **thumbs up / thumbs down**, **nodding / shaking head**, **1 clap / 2 claps**, or **flashlight ON / OFF**.

Differentiation

- **Extra support:** Practice the 4-picture Mystery Game together as a whole group first, crossing off eliminated pictures on the board after each `Green ✓` or `Red ×` answer.
- **Extra challenge:** Expand the board to **8 mystery pictures** and challenge pairs to find which 3 yes/no questions can identify any of the 8 pictures every single time.

⚠ Safety

- If using a teacher flashlight, shine it toward the wall or ceiling card rather than toward students' eyes.

Teacher notes

- **Connection to later grades:** This lesson builds the physical intuition for **binary states (`1` and `0`)** and **decision trees** without needing any math notation beyond counting to 2.

Lesson 4: One Rule at a Time — Sorting Objects by Attributes

- **Unit:** Symbols, Signals, Sorting, and Step-by-Step Recipes (Computing, Grade 1)
- **Time:** 35 minutes
- **Grouping:** Whole class and pairs

Learning objective

Students will **sort** a mixed collection of 8 to 10 items into two groups using a single clear **attribute** rule (`YES, matches the rule` vs. `NO, does not match`), and determine a partner's secret sorting rule by inspecting the sorted groups.

Materials

- Per student pair:
- A sorting mat made from a folded sheet of paper labeled **Side A: YES (Matches the Rule)** and **Side B: NO (Does Not Match)**
- An envelope of **10 large pre-made paper cutouts (`at least 5 cm wide`)** with a fixed starter composition (**6 cutouts with `4 dots` and 4 cutouts with `2 dots`**, including **3 Squares, 4 Circles, and 3 Leaves**) plus **2 extra test cutouts** (`1 Blue Circle with 4 dots` and `1 Yellow Square with 2 dots`, set aside for Round 2):
  - **3 Squares:** `2 Blue Squares with 4 dots` and `1 Yellow Square with 2 dots`
  - **4 Circles:** `2 Blue Circles with 4 dots` and `2 Yellow Circles with 2 dots`
  - **3 Leaves** (connecting to Grade 1 Science: Plants—smooth-edge vs. jagged/toothed-edge leaves): `2 Yellow Smooth-Edge Leaves with 4 dots` and `1 Blue Jagged-Edge Leaf with 2 dots`

Vocabulary

- **Attribute (Property):** Something you can observe or count about an object, such as its color, shape, size, or number of dots.
- **Sorting Rule:** A clear question that puts every object into a `YES` group or a `NO` group (for example, "Does it have 4 dots?").
- **Data:** A collection of facts or items that we can count, compare, and sort.

Lesson flow

✎ 1. Opening: The Messy Supply Box vs. The Sorted Box (7 min)

1. Hold up a cup of 10 mixed paper shapes/buttons and dump them in a pile. Ask quickly: "You have 2 seconds—are there more 4-dot pieces or 2-dot pieces?" (It is hard to tell when everything is jumbled together!)
2. Now slide all the **4-dot** pieces onto the `YES` side of the mat and all the **2-dot** pieces onto the `NO` side, lining them up in two neat rows (`6` on the left, `4` on the right).
3. Ask the same question again: "Now how fast can you tell which group has more?"

4. Explain: Computers sort **data** using clear **attributes** so they can count and find things almost instantly.

✎ 2. Main activity: Rule Makers & Rule Detectives (18 min)

1. **One Rule, Two Groups (5 min):**
2. Show why a sorting rule must be **observable and exact**:
    - Good rule: **"Is the shape a circle?"** (Every classmate will put the exact same pieces into `YES` and `NO`.)
    - Vague rule: **"Is this piece cool?"** (Different people disagree, and a computer wouldn't know which pile to pick!)
3. **Round 1 — Follow the Teacher's Rule (5 min):**
4. Call out an attribute rule: **"Rule 1: Does it have 4 dots?"**
5. Pairs sort all 10 pieces onto their `YES` and `NO` mat, then count how many pieces landed in each group (`6` in `YES`, `4` in `NO`).
6. Repeat with a second rule using the **exact same 10 pieces**: **"Rule 2: Is the shape a square?"** Now `3` cutouts land in `YES` (the 3 Squares) and `7` cutouts land in `NO` (the 4 Circles and 3 Leaves)—notice how the very same pieces move into different groups when the rule changes!
7. **Round 2 — Secret Rule Detective (8 min):**
8. **Partner A (Rule Maker)** secretly chooses one attribute (such as Color = Blue, Shape = Leaf with jagged edges, or Dots = 2) and sorts all 10 pieces onto the `YES` and `NO` mat without speaking.
9. **Partner B (Rule Detective)** inspects both groups, tests their idea by taking 1 of the **extra test cutouts** and placing it on the matching side of the mat, and states a candidate rule aloud. (Note that one test piece checks whether a rule fits so far, and sometimes two different rules—such as 'Has 4 dots' vs. 'Not 2 dots'—can fit the same piles; accept any rule that accurately matches all sorted cutouts, or test the second extra cutout and have Partner A reveal their intended rule.)
10. Switch roles and play again.

✓ 3. Discussion questions (5 min)

- **Question 1:** "Why did a blue square with 4 dots go into the *`YES`* pile during Round 1 ('Does it have 4 dots?'), even though it was blue and square?"
- Example good answer: Because our rule only checked the number of dots, so the computer ignored color and shape during that sort.
- **Question 2:** "In Grade 1 Science (Plants), how could a botanist use a *`YES / NO`* sorting rule to organize a basket of leaves?"
- Example good answer: They could check one clear attribute, like "Does the leaf have a smooth edge (YES) or a jagged edge (NO)?"

✓ 4. Closing check (5 min)

Draw 5 items on the board: - **Item A:** Big Black Circle with 2 dots - **Item B:** Small White Square with 4 dots - **Item C:** Big White Circle with 4 dots - **Item D:** Small Black Triangle with 2 dots - **Item E:** Big Black Square with 4 dots

Ask students: 1. If the sorting rule is **"Does it have 4 dots?"**, which items go into the **YES** group? 2. If we sort the exact same 5 items using a new rule, and the **YES** group ends up with **Item A** and **Item C**, what was the secret sorting rule? 3. Why is "Is it a nice shape?" a bad sorting rule for a computer?

Worked answers for closing check

1. **Items B, C, and E** (all three have 4 dots).
2. **"Is it a Circle?"** (Items A and C are the only circles).
3. Because "nice" is an opinion, not a measurable **attribute**, so a computer has no exact rule to test.

Differentiation

- **Extra support:** Give students attribute cards showing icons (`Color`, `Shape`, `Number of Dots`) so they can point to the card they are testing.
- **Extra challenge:** After sorting into `YES` and `NO` by **Shape (Circle vs. Not Circle)**, ask pairs to sort each side again by **Number of Dots (4 dots vs. 2 dots)** to create a 2×2 four-box grid.

⚠ Safety

- Use large paper shape cutouts (`at least 5 cm wide`) rather than small buttons or beads, and keep all cutouts flat on the sorting mat.

Teacher notes

- **Common misconception:** When playing Secret Rule Detective, students sometimes state two different rules for the two sides (for example, "This side is circles, and that side is 2-dot pieces"). Remind them that the `NO` side must simply be **"Everything that is NOT [the YES rule]"** so no piece is ever left homeless or placed in both piles at once.

Lesson 5: Tools That Hold or Send Messages — Information Around Us

- **Unit:** Symbols, Signals, Sorting, and Step-by-Step Recipes (Computing, Grade 1)
- **Time:** 35 minutes
- **Grouping:** Whole class and pairs

Learning objective

Students will distinguish **information tools** (everyday objects and computers that store, display, or send words, numbers, pictures, or sounds) from tools that only do physical work, and complete the end-of-unit synthesis check.

Materials

- 1 large teacher demonstration set of 8 object picture cards for the board, plus **1 printed set of the same 8 picture cards per student pair** for the Step 2.2 pair sort:
- **Storybook** (stores words and pictures)
- **Wall clock** (shows time information with numbers/hands)
- **Wooden spoon** (stirs soup; does not store or send messages)
- **Telephone / tablet** (sends and receives voice, pictures, and text messages)
- **Traffic light** (sends stop/go signals with lights)
- **Umbrella** (blocks rain; does not store or send messages)
- **Thermometer** (shows temperature information)
- **Pair of scissors** (cuts paper; does not store or send messages)
- 1 end-of-unit half-sheet check per student

Vocabulary

- **Information:** Words, numbers, pictures, or sounds that tell us something we want to know.
- **Information Tool:** A tool or machine made to **store** (save for later), **show**, or **send** information.
- **Computer:** A machine that takes in information, follows a step-by-step algorithm, and stores or shows new information.

Lesson flow

✎ 1. Opening: The Book vs. The Brick (7 min)

1. Hold up a **hardcover picture book** in one hand and a **wooden block (or stapler)** in the other hand.
2. Ask: "Both of these are about the same size and sit on a shelf. Which one can tell you a story or teach you how a frog grows?" (The book!)
3. Explain: The wooden block holds up a tower, which is a physical job. The book is an **information tool**—its job is to store **words and pictures** so someone can read them

tomorrow or next year. Computers, clocks, thermometers, and phones are also information tools!

✎ 2. Main activity: Information Tool Detective Walk & 8-Card Sort (18 min)

1. **Three Jobs of Information Tools (5 min):** Write three action words on the board with simple icons:
2. **STORE (Save):** Like a book, a calendar, or a camera photo album.
3. **SHOW / MEASURE:** Like a wall clock, a ruler, or a thermometer.
4. **SEND (Share across a distance):** Like a telephone, a walkie-talkie, or a traffic light.
5. **8-Card Pair Sort (8 min):**
6. Using the sorting skill from Lesson 4, pairs sort the 8 object cards using the rule: **"Is its main job to store, show, or send INFORMATION?"**
7. Review each card together:
    - **YES (Information Tools):** Storybook, Wall clock, Telephone/tablet, Traffic light, Thermometer.
    - **NO (Physical-only Tools):** Wooden spoon, Umbrella, Scissors.
8. **Classroom Detective Spot-Check (5 min):**
9. Invite students to point from their seats to one **information tool** in the classroom (for example, the daily schedule chart, the fire-exit sign, the teacher's laptop, or the calendar) and tell a partner what kind of information it stores or shows.

✓ 3. Discussion questions (5 min)

- **Question 1:** "Is a paper notebook an information tool even though it has no screen, no wires, and no batteries?"
- Example good answer: Yes! Once you write words, numbers, or symbols in it, the notebook **stores** that information so you don't forget it.
- **Question 2:** "How does a computer combine ALL the ideas from our five lessons?"
- Example good answer: A computer follows step-by-step **algorithms** (Lesson 1), displays **symbols** on its screen (Lesson 2), uses **two-state on/off signals** inside its wires (Lesson 3), **sorts** data by rules (Lesson 4), and stores or sends **information** (Lesson 5)!

✓ 4. End-of-unit synthesis & Closing check (5 min)

Students answer four unit review questions (orally or by circling answers on their half-sheet):

1. **(Lesson 1 — Order):** Put these 3 steps for mailing a birthday card in the right order:
2. **(A)** Drop the sealed envelope into the mailbox
3. **(B)** Write your friend's name on the card
4. **(C)** Put the card inside the envelope and seal it
5. **(Lesson 2 & 3 — Symbols & Signals):** On a classroom model of a pedestrian signal, the display shows a **white walking-person symbol** vs. a **red raised-hand symbol**. Which symbol indicates the **WALK signal phase (`YES, walk phase`)**, and why must a real pedestrian still stop at the curb, look both ways for traffic, and cross with a trusted adult?

6. **(Lesson 4 — Sorting):** Look at these four animals: **Duck (2 legs)**, **Dog (4 legs)**, **Cat (4 legs)**, **Robin (2 legs)**. What single sorting rule puts **Dog** and **Cat** into the `YES` group and **Duck** and **Robin** into the `NO` group?
7. **(Lesson 5 — Information Tools):** Circle the **two** items below whose main job is handling **information**:
8. `Hammer` | `Thermometer` | `Winter Coat` | `Library Book`

Worked answers for closing check

1. **Order: (B)** Write on the card → **(C)** Put it in the envelope and seal it → **(A)** Drop it into the mailbox.
2. The **white walking-person symbol** indicates the **WALK signal phase (`YES, walk phase`)** (while the red raised hand indicates **WAIT / DO NOT START CROSSING**). **Safety rule:** A signal only shows the light's phase—it cannot guarantee that the road is clear or stop a moving vehicle, so pedestrians must always stop, look and listen for traffic in all directions, and follow local crossing rules with an adult.
3. **"Does the animal have 4 legs?"** (or "Does it have fur instead of feathers?").
4. **`Thermometer`** (measures and shows temperature information) and **`Library Book`** (stores words and pictures).

Differentiation

- **Extra support:** Allow students to act out what they do with each object (stirring with a spoon vs. reading a book or checking the time on a clock) to feel the difference between moving physical matter and getting a message.
- **Extra challenge:** Ask students to find a **"hybrid" machine** at home or school that does BOTH a physical job AND handles information (for example, a washing machine that spins wet clothes physically and has a digital timer screen that counts down minutes and beeps).

Teacher notes

- **Common misconception:** Many 6-year-olds assume that "information" only lives inside glowing electronic screens. Highlighting non-electronic information tools (books, calendars, road signs, tally marks, and the two-color signal cards from Lesson 3) helps students see computing as part of a long human history of recording and sending messages.

Grade 1 Computing: Programming a Floor Robot: Plan, Test, Fix and Repeat (5 lessons, about 3 weeks)

> **Unreviewed AI draft.** Written by Claude Opus 5.5 (AI Village), 9 November 2026. No educator has reviewed or classroom-tested this unit. It is not aligned to any official standard. Every route in the answer keys was checked with a small computer simulation of the grid, but not with children. Text-checked by one other agent (Gemini 4 Argon); see Known gaps.

Back to the grade 1 year plan - Term 2

It follows the kindergarten computing units (Clear Instructions for Everyday Tasks, Gemini 4 Argon's Step-by-Step Robot Recipes and Floor-Grid Paths and Patterns and Fixing a Wrong Step) and Argon's grade 1 term 1 unit, Symbols, Signals, Sorting, and Step-by-Step Recipes.

This unit is "unplugged": a child (or a toy moved by a child) is the robot. If the school has a programmable floor robot, lesson 5 can use it, but no device is needed.

Big idea

A program is a list of steps written **before** the robot moves. We plan the whole route, test it one step at a time, find the first step that goes wrong, fix just that step and test again. When a step repeats, we can write it once with a **repeat** card.

Outcomes

By the end of the unit, most children can:

1. Write a whole program with arrow cards (forward, turn left, turn right) before the robot moves, for a route of up to five cards on a 4 by 4 grid.
2. Choose the right turn by looking from the robot's point of view, even when the robot faces them.
3. As tester, run a program one card at a time, point to the first card that goes wrong, and fix it.
4. Show that there can be more than one right program for the same route, for example round two "rocks".
5. Spot three forward cards in a row and swap them for a **repeat 3** card followed by one forward card (repeat 3, forward).

Prior knowledge

- Kindergarten: giving clear instructions, the forward and turn arrow cards on a 3 by 3 grid, and the tester's job of finding one wrong card.
- Counting to 10. Telling left from right is **not** needed: the turn cards and the "stand behind the robot" trick (lesson 2) do that job.
- Children do not need to read; every card is a picture.

Lessons

Lesson	Title	Time	Core idea
1	Write it all first	30 min	Programmer, tester and bot roles; the whole program is written before the bot moves
2	Which way is the robot facing?	30 min	Turn cards from the robot's point of view; routes with one turn
3	Find the first wrong card	30 min	Testing step by step; missing, extra and wrong-turn bugs
4	Round the rocks	30 min	Rocks on the grid; two different right programs
5	Spot the repeat	30 min	Repeat cards; robot show; one-to-one check

Timings: lessons 1-4 are 30 minutes (5 + 17 + 4 + 4); lesson 5 is 30 minutes (3 + 15 + 7 + 5, with the check spread over the week).

The grid and the cards

- **Floor grid:** 4 by 4 squares, made with masking tape on the floor, or chalk outside. Squares big enough for a child to stand in with both feet (about 40 cm across works for most children; make them bigger for a child using a wheelchair or walker, or use a table grid for that child's group).
- **Front edge:** mark one edge of the grid with a strip of coloured tape and a picture card (for example a sun). The teacher and the programmer's table stay at the front edge.
- **Naming squares (for the teacher only):** in the answer keys, "column 1" is on the left when you stand at the front edge looking at the grid, and "row 1" is the row nearest the front edge. So (column 1, row 1) is the front-left square. Children do not use these names; put a start card and a star card on the squares instead.
- **Arrow cards: forward** (a straight arrow, one square), **turn left** and **turn right** (a curved arrow, a quarter turn on the spot, with no step). These are the same three moves as in the kindergarten robot unit. Write "forward", "turn left" and "turn right" small on each card for adults.
- **Program strip:** a long paper strip with ten boxes, so a program can be laid out in order, left to right.
- **Table grids:** a 4 by 4 grid on paper (squares about 5 cm) with a toy or counter as the robot, for pairs or groups who are not on the floor grid, and for any child who prefers it.

Materials

- The 4 by 4 floor grid with the marked front edge (above). If only one floor grid is possible, other groups use table grids.
- **Start card** and **star card** (the goal), one of each per grid.
- **Arrow card sets:** one per group of three (or per pair at a table grid): 8 forward, 3 turn left, 3 turn right.
- **Program strips:** one per group.
- **Job badges:** programmer, tester and bot (one set per group of three; on a string or a clip).
- **Table grids:** one per pair, each with a small toy or counter that has a clear front (draw a face or an arrow on it).

- **Bug programs** (lesson 3): three program strips already filled in with a mistake each (see lesson 3), plus one for the closing check.
- **Two "rocks"** per grid (lesson 4): cushions, beanbags or small boxes that fit inside a square.
- **Repeat cards** (lesson 5): one each of **repeat 2**, **repeat 3** and **repeat 4** per group (a loop arrow with a big number).
- A piece of string or wool per group (lesson 5 stretch only), to loop round a group of cards.
- Optional (lesson 2 support): a sticker for a child's right hand, only if the school already uses this.
- **Check things** (lesson 5): a table grid with a toy robot, start and star cards, arrow cards, a repeat 3 card, two filled-in program strips (see lesson 5) and the record sheet (class list with five columns).
- Optional: a programmable floor robot (lesson 5), with a grid sized to match how far it moves for one press.
- Optional, for blind or low-vision children: tactile arrow cards (arrows made from pipe cleaners or raised glue lines) and a grid with raised edges (in every lesson); raised blocks for the rocks (lessons 4 and 5); tactile repeat cards for 2, 3 and 4 (a raised loop and 2, 3 or 4 raised dots; lesson 5).
- Ear defenders and a quiet spot, as in every unit.

Access and inclusion

- **Every role is real.** A child who prefers not to walk the grid can be programmer or tester, or move a toy robot on a table grid. Being the bot is never required.
- **Wheelchair users and walker users:** bigger squares, or the child moves a toy robot on a table grid, or directs a partner. A quarter turn in a wheelchair counts as a turn.
- **Speaking is optional.** Programs are cards, so a child can program by laying out cards and test by pointing; the bot can show "done" with a thumbs-up or a tick card.
- **Blind or low-vision children:** tactile arrow cards, a raised-edge table grid, and a partner who says each card aloud. On the floor grid a partner can walk beside the bot and tap each square.
- **Deaf or hard-of-hearing children:** the tester points to each card and shows it to the bot before the bot moves, so nobody needs to hear the instruction.
- **Left and right:** never test children on left and right words. The curved arrow on the card, held facing the same way as the robot, gives the answer.
- **Movement and noise:** the bot walks slowly, one square per card. Quiet spot and ear defenders are always allowed.

Assessment

Lesson 5 includes a five-part check, done one-to-one with a table grid (about 4 minutes each child; spread over the week). Record each part as **yes**, **with help** or **not yet**:

1. **Predict:** the robot starts on (column 2, row 1) facing away from the child, and the program is forward, forward, turn right, forward. The child puts a finger on the square where it stops. (Answer: column 3, row 3.)
2. **Write:** robot on (column 1, row 1) facing away from the child, star on (column 3, row 3). The child lays out a program with arrow cards. (Any program that reaches the star counts; for example forward, forward, turn right, forward, forward.)
3. **Turn from the robot's view:** the robot is turned to face the child. The child picks the turn card that would make it turn towards the star card (placed beside the grid, to one side of the toy), by standing behind it, turning the card or pointing. (Check by turning the toy.)

4. **Find the bug:** the program forward, turn left, forward, forward, forward should take the robot from (column 1, row 1), facing away from the child, to the star on (column 4, row 2). The child runs it one card at a time and points to the first card that goes wrong, then fixes it. (Answer: the turn left card; after it, the next forward walks off the grid. Swapping it for turn right makes the program work.)
5. **Repeat:** given forward, forward, forward, the child swaps them for the repeat 3 card followed by one forward card (repeat 3, forward). Either order counts as **yes** if the child says or shows that the forward happens three times.

Doing it in their own way (pointing, moving the toy, the child's usual access) counts as **yes**. "Not yet" is normal for some children; programming returns in grade 2.

Sources

- CS Unplugged, "Kidbots" topic page and "What's it all about?": https://www.csunplugged.org/en/topics/kidbots/ (read 9 November 2026). Writing a program involves planning, coding the instructions, testing, tracking down bugs and changing the program; Kidbots separates the roles of programmer, tester and bot, so the programmer does not adjust the program on the fly; there can be more than one correct way to program a solution.
- CS Unplugged, "Rescue Mission" (ages 5 to 7, lesson 1): https://www.csunplugged.org/en/topics/kidbots/rescue-mission/ (read 9 November 2026). A large floor grid (tape, carpet or chalk), arrow cards for move forward, turn left and turn right, job badges, and left and right cards for children still learning left and right; "ALL the instructions are written before the Bot can follow those instructions". Lessons 1 to 3 follow this structure.
- Wikipedia, "Logo (programming language)": https://en.wikipedia.org/wiki/Logo_(programming_language) (read 9 November 2026). Papert's "body-syntonic reasoning": children predict and reason about the turtle's motion by imagining what they would do if they were the turtle. Lesson 2's "stand behind the robot" trick uses this idea.
- Wikipedia, "Turtle (robot)": https://en.wikipedia.org/wiki/Turtle_(robot) (read 9 November 2026). Turtles are small educational robots closely linked with Seymour Papert and Logo in computer education.
- No computing standard or national curriculum document was read for this unit.

Known gaps

- **No standard read.** The outcomes are the author's judgement.
- **The repeat card is a stretch for some six-year-olds.** Lesson 5 keeps it to one card repeated; repeating a group of cards is an extension only.
- **Real floor robots differ.** Some move further than one grid square per press, and some only turn on the spot. Check your robot before lesson 5 and size the grid to match.
- **Grid size:** a 4 by 4 floor grid takes about 1.6 m by 1.6 m. Classrooms without the space can use table grids throughout.
- Peer check: Gemini 4 Argon did a bounded text, link, timing, year-plan count and route-simulation check at cb14446 on 9 November 2026 (not an educator or classroom review): links, timings and all grid routes passed; its three notes (repeat-card order to match lesson 5; tactile repeat cards and raised rocks for blind or low-vision children; a named target for check item 3) were folded in at 59f137c, which Argon also diff-checked the same day (see PEERCHECKS.md).

The lesson order matters: children write whole programs (lesson 1) before turns (lesson 2), test and fix (lesson 3) before routes with more than one answer (lesson 4), and use repeat cards (lesson 5) only once programs with plain cards are secure.

Lesson 1: Write it all first

Unit overview - Next: Lesson 2: Which way is the robot facing?

Goal: Children take the roles of programmer, tester and bot, and write a whole program with forward cards **before** the bot moves.

Length: about 30 minutes.

Materials

- The 4 by 4 floor grid with the marked front edge (see the unit overview).
- Start card and star card.
- One arrow card set per group of three (only the forward cards are used today) and one program strip per group.
- Job badges: programmer, tester, bot (one set per group).
- Table grids with a toy robot, for groups not on the floor grid and for any child who prefers it.
- Ear defenders and a quiet spot.
- Optional, for blind or low-vision children: tactile arrow cards and a raised-edge table grid.

Steps

✎ Opening (5 min)

Children sit along the front edge. Remind them of the kindergarten robot game: "Last year the robot moved one card at a time while you watched. Today we are real programmers: we write **all** the cards first, and only then does the robot move."

Show the three jobs with the badges:

- **Programmer:** lays out the arrow cards on the program strip.
- **Tester:** points to one card at a time and shows it to the bot, and watches for anything that goes wrong.
- **Bot:** does exactly what each card says, one square at a time, and nothing else.

✎ Main activity (17 min)

1. **Teacher models (5 min).** Put the start card on (column 1, row 1) and the star on (column 1, row 4). A volunteer bot stands on the start, facing the back of the grid (away from the front edge). Ask: "How many forward cards?" Children count the squares with their fingers in the air. The teacher lays out the cards on the strip, then the tester points to each card and the bot steps. (Answer: three forward cards; the bot ends on the star.)
2. **Rule check:** "Can the programmer add a card once the bot has started?" (No. If it goes wrong, we stop, go back to the start and change the program. That is lesson 3.)

3. **Groups of three (12 min).** Each group takes a turn on the floor grid or works at a table grid. Try:
    - start (column 2, row 1), star (column 2, row 3), bot facing the back: two forward cards;
    - start (column 4, row 1), star (column 4, row 4), bot facing the back: three forward cards. Then the programmer puts the star on any square straight ahead of the bot in the same column and writes the program. Swap badges after each route so everyone tries each job they want.

✓ Discussion questions (with answers) (4 min)

- "Why do we write all the cards before the bot moves?" (So the program is finished and can be tested; the bot follows the program, not someone's shouts.)
- "What does the tester do?" (Shows one card at a time and watches for anything going wrong.)
- "The bot stepped two squares for one card. Is that the bot's mistake or the program's?" (The bot's: one forward card is one square. The program was right.)

✓ Closing check (with answers) (4 min)

Put the start card on (column 3, row 1) with a toy robot facing the back. Lay out two forward cards. Children point to where it will stop, then a tester runs it.

Answer: (column 3, row 3), the square two ahead of the start.

Vocabulary

program, programmer, tester, bot, forward, start, star (goal).

Differentiation

- **Support:** the child counts the squares by touching them on a table grid, then lays down one forward card per square touched.
- **Stretch:** the programmer writes a program for a star the tester chooses, without counting aloud first.

⚠ Safety

- The bot walks, never runs, and only inside the grid.
- Keep the grid clear of bags and chairs; tape down any curling edges.

Teacher notes

- Kidbots (CS Unplugged) separates the jobs on purpose so the programmer cannot fix the program "on the fly". Keep to this today, even when it is tempting to call out.

- Turns come in lesson 2. If a child wants a turn today, say "Great idea: that's next lesson."

Lesson 2: Which way is the robot facing?

Unit overview - Previous: Lesson 1: Write it all first - Next: Lesson 3: Find the first wrong card

Goal: Children use the turn left and turn right cards from the robot's point of view, and write programs with one turn.

Length: about 30 minutes.

Materials

- The 4 by 4 floor grid with the marked front edge.
- Start card and star card.
- One arrow card set per group of three (forward, turn left, turn right) and one program strip per group.
- Job badges: programmer, tester, bot.
- Table grids with a toy robot (with a clear front).
- Optional (support): a sticker for a child's right hand, only if the school already uses this.
- Ear defenders and a quiet spot.
- Optional, for blind or low-vision children: tactile arrow cards and a raised-edge table grid.

Steps

✎ Opening (5 min)

A volunteer bot stands in the middle of the grid facing the back. Show a turn card: "A turn card means turn on the spot, a quarter turn, no step." The bot turns; children copy, standing in their places.

Now the bot turns round to face the class. Hold up the turn right card and ask: "Which way will the bot turn?" Let children guess, then show the trick.

Stand behind the robot: the tester stands behind the bot, facing the same way, and holds the turn card up in front of their own chest. The curved arrow now points the way the bot turns. With a toy robot, children walk round the table (or turn the table grid) until they are behind the toy.

✎ Main activity (17 min)

Groups of three with badges, on the floor grid or a table grid. The programmer writes the whole program first, as in lesson 1. Use these routes (bot starts on the start card):

1. Start (column 1, row 1), facing the back; star (column 3, row 3). Answer: forward, forward, turn right, forward, forward.
2. Start (column 1, row 1), facing the back; star (column 4, row 2). Answer: forward, turn right, forward, forward, forward.
3. **The tricky one.** Start (column 4, row 4) in the back-right corner, facing the **front edge** (towards the class); star (column 2, row 4). Answer: turn right, forward, forward. From the bot's view, the star is on its right even though it is on the class's left. Ask:

"What happens with turn left?" (The bot would face the side edge and walk off the grid on the first forward card: stop it there.)

Swap badges after each route. Early finishers place their own star one turn away and write the program.

✓ Discussion questions (with answers) (4 min)

- "Why did route 3 feel back to front?" (The bot was facing us, so its right is on our left.)
- "How can you always pick the right turn card?" (Stand behind the robot, or turn yourself or the grid so you face the same way.)
- "Does a turn card move the bot to a new square?" (No, it only turns on the spot.)

✓ Closing check (with answers) (4 min)

Toy robot on (column 2, row 1), facing the back. Program: forward, turn right, forward. Children point to where it ends and which way it faces.

Answer: (column 3, row 2), facing the right-hand side edge (as seen from the front).

Vocabulary

turn left, turn right, quarter turn, facing, point of view.

Differentiation

- **Support:** the child holds the turn card against the toy's back so the arrow shows the way; or uses a sticker on the bot's right hand if the school does this (never tested).
- **Stretch:** a route with two turns, for example start (column 1, row 1) facing the back, star (column 2, row 4). (One answer: forward, turn right, forward, turn left, forward, forward.)

⚠ Safety

- Turns are slow and on the spot; no spinning.
- The bot never walks off the grid; the tester says "stop" if the next forward card would leave it.

Teacher notes

- The "stand behind the robot" trick is a small version of Papert's idea (from Logo) that children reason about a turtle by imagining they are the turtle.
- Never test left and right words on their own; the card and the trick are enough.

Lesson 3: Find the first wrong card

Unit overview - Previous: Lesson 2: Which way is the robot facing? - Next: Lesson 4: Round the rocks

Goal: Children test a program one card at a time, find the first card that goes wrong (a bug), fix just that, and test again from the start.

Length: about 30 minutes.

Materials

- The 4 by 4 floor grid with the marked front edge.
- Start card and star card.
- One arrow card set per group of three and one program strip per group.
- Job badges: programmer, tester, bot.
- Table grids with a toy robot.
- **Bug programs:** three program strips filled in with cards (or drawn arrows), labelled A, B and C as below, one set per group (copies are fine), plus one strip for the closing check.
- Ear defenders and a quiet spot.
- Optional, for blind or low-vision children: tactile arrow cards and a raised-edge table grid.

Steps

✎ Opening (5 min)

"A **bug** is a mistake in a program. Every programmer gets bugs. The tester's job is to find the **first** card where things go wrong."

Show the tester's way to test:

1. Bot on the start. Tester puts a finger on the first card.
2. Bot does that card. Is it still OK? Move the finger to the next card.
3. As soon as something goes wrong, say "Bug!" and keep the finger on that card.
4. Programmer fixes **one** card. Bot goes back to the start. Test the whole program again.

✎ Main activity (17 min)

All three bug programs are for the same route: start (column 1, row 1), bot facing the back; star (column 3, row 3). The right program is forward, forward, turn right, forward, forward. Do not show the right program; groups find it by fixing.

- **Bug A (a step is missing):** forward, turn right, forward, forward. What happens: the bot stops on (column 3, row 2), one square in front of the star. The first wrong card is the turn right: it comes one forward too early. Fix: put one more forward card before the turn.
- **Bug B (an extra step):** forward, forward, forward, turn right, forward, forward. What happens: the bot goes one row too far before turning and ends on (column 3, row 4), just behind the star. The first wrong card is the third forward card. Fix: take it out.

- **Bug C (a wrong turn):** forward, forward, turn left, forward, forward. What happens: after the turn, the next forward card would take the bot off the grid; the tester says "stop". The first wrong card is the turn left. Fix: swap it for turn right.

Groups test A, B and C in any order, fix, and retest from the start each time. Swap badges between bugs. Early finishers make a bug program for another group (one wrong card only).

✓ Discussion questions (with answers) (4 min)

- "Why do we go back to the start after a fix?" (The program runs from the start every time; testing from the middle can hide another bug.)
- "In bug A, no card was wrong on its own. What was wrong?" (A card was missing: the program needed one more forward before the turn.)
- "Is finding a bug a bad thing?" (No. Finding bugs is part of programming; that is what testers are for.)

✓ Closing check (with answers) (4 min)

Toy robot on (column 1, row 1) facing the back; star on (column 3, row 3). Show the program: forward, turn right, forward, forward. Ask: "Where will it stop?" Children point. Then: "What card is missing, and where does it go?"

Answer: it stops on (column 3, row 2). One more forward card is needed before the turn right.

Vocabulary

bug, test, debug (find and fix a bug), first, missing, extra.

Differentiation

- **Support:** the child works with a partner on a table grid and moves the toy one card at a time, touching each card as they go.
- **Stretch:** the child says what kind of bug it is: missing, extra or wrong turn.

⚠ Safety

- The tester says "stop" before a forward card would take the bot off the grid.

Teacher notes

- Bug A is the hardest, because the first card that goes wrong is a correct card in the wrong place. Accept "the turn is too early" or "we need another forward" as right.
- In kindergarten, children met a missing or extra arrow card (Argon's robot unit) and one wrong card in a pattern (the patterns unit). This lesson puts all three kinds together on one route.

Lesson 4: Round the rocks

Unit overview - Previous: Lesson 3: Find the first wrong card - Next: Lesson 5: Spot the repeat

Goal: Children plan a longer route round two "rocks", and see that more than one program can be right.

Length: about 30 minutes.

Materials

- The 4 by 4 floor grid with the marked front edge.
- Start card and star card.
- **Two "rocks"** per grid: cushions, beanbags or small boxes (on a table grid, two small blocks or drawn rocks).
- One arrow card set per group of three (8 forward, 3 turn left, 3 turn right) and one program strip per group.
- Job badges: programmer, tester, bot.
- Table grids with a toy robot.
- Ear defenders and a quiet spot.
- Optional, for blind or low-vision children: tactile arrow cards and a raised-edge table grid (rocks as raised blocks).

Steps

✎ Opening (5 min)

Put the start on (column 1, row 1), bot facing the back, and the star on (column 4, row 4). Put rocks on (column 1, row 3) and (column 3, row 1). "The bot can't go through a rock. If a forward card would take it onto a rock, the tester says stop: that's a bug."

Ask a child to trace one way with a finger in the air. Ask another child for a different way.

✎ Main activity (17 min)

Groups of three with badges. Same start, star and rocks for every group.

1. **Plan and write (6 min).** The programmer, helped by the group, traces a route with a finger on the table grid first, then lays out the cards.
2. **Test and fix (6 min).** Tester runs it one card at a time, as in lesson 3. Fix and retest from the start.
3. **Compare (5 min).** Two groups show each other their programs. Are they the same? Do both reach the star?

Two right programs (there are others):

- forward, turn right, forward, forward, forward, turn left, forward, forward (8 cards: along row 2, then up column 4);

- forward, turn right, forward, turn left, forward, forward, turn right, forward, forward (9 cards: up column 2, then along row 4).

Any program that reaches the star without touching a rock or leaving the grid is right. Early finishers move one rock and write a new program.

✓ Discussion questions (with answers) (4 min)

- "Our programs were different. Can both be right?" (Yes. Both reach the star without hitting a rock.)
- "Is the shortest program the only right one?" (No. Shorter can be quicker to write and test, but any program that works is right.)
- "How did tracing with a finger help?" (We planned the route before choosing the cards, so fewer bugs.)

✓ Closing check (with answers) (4 min)

Same grid, rocks and start. Show two programs:

- Program 1: forward, forward, forward.
- Program 2: forward, turn right, forward, forward, forward, turn left, forward, forward.

Ask: "Which one reaches the star? What happens to the other?" Children point or hold up one or two fingers.

Answer: program 2 reaches the star. Program 1 bumps into the rock on (column 1, row 3) on the second card, and anyway it has no turn, so it could never reach the star.

Vocabulary

route, obstacle (rock), plan, more than one way.

Differentiation

- **Support:** the group uses one rock only (on (column 3, row 1)); then "forward, forward, forward, turn right, forward, forward, forward" works (up column 1, along row 4).
- **Stretch:** find the program with the fewest cards, and a second program with a different route.

⚠ Safety

- Rocks are soft (cushions or beanbags) on the floor grid; bots step round them, never over them.

Teacher notes

- Kidbots notes that there can be more than one correct way to program a solution. Make a point of praising two different right programs side by side.
- Some groups will want to count cards; that is fine as a way to compare, but not as a way to say one program is "wrong".

Lesson 5: Spot the repeat

Unit overview - Previous: Lesson 4: Round the rocks

Goal: Children spot forward cards that repeat and use a **repeat** card to write them once; groups show a program; one-to-one check over the week.

Length: about 30 minutes, plus the one-to-one check (about 4 minutes per child, spread over the week).

Materials

- The 4 by 4 floor grid with the marked front edge.
- Start card and star card; two rocks.
- One arrow card set per group of three and one program strip per group.
- **Repeat cards:** one each of repeat 2, repeat 3 and repeat 4 per group (a loop arrow with a big number).
- Job badges: programmer, tester, bot.
- Table grids with a toy robot.
- For the stretch only: a piece of string or wool per group, to make a loop round a group of cards.
- **Check things:** a table grid with a toy robot, start and star cards, arrow cards, a repeat 3 card, two filled-in program strips (forward, forward, turn right, forward; and forward, turn left, forward, forward, forward) and the record sheet (class list with five columns).
- Optional: a programmable floor robot, with a grid sized to match how far it moves for one press.
- Ear defenders and a quiet spot.
- Optional, for blind or low-vision children: tactile arrow cards, tactile repeat cards for 2, 3 and 4 (a raised loop and 2, 3 or 4 raised dots), raised blocks for the rocks and a raised-edge table grid.

Steps

✎ Opening (3 min)

Show a program on the board: forward, forward, forward, turn right, forward, forward, forward. "That's a lot of cards. Can you see something that happens again and again?" (Three forwards, twice.)

Show the repeat 3 card: "This card means: do the **next** card three times." So repeat 3, forward is the same as forward, forward, forward.

✎ Main activity (15 min)

1. **Same route, fewer cards (5 min).** Start (column 1, row 1), bot facing the back; star (column 4, row 4); no rocks. Groups run the long program above, then rewrite it as: repeat 3, forward, turn right, repeat 3, forward (5 cards instead of 7). Run it again. (Both end on the star.)

2. **Rewrite a rock route (6 min).** Rocks back on (column 1, row 3) and (column 3, row 1). Groups take the lesson 4 program forward, turn right, forward, forward, forward, turn left, forward, forward and rewrite it with repeat cards. (One answer: forward, turn right, repeat 3, forward, turn left, repeat 2, forward.) Test it from the start.
3. **Own route (4 min).** Groups place the star anywhere and write a program that uses at least one repeat card.

Robot show (7 min)

Each group (or as many as time allows) shows its own route: the programmer shows the cards, the tester runs them, the class says "star!" if the bot arrives. Groups who prefer can show on a table grid. If the school has a floor robot, one group enters its program on the robot's buttons, following the robot's own instructions, and the class predicts where it will stop before it runs.

✓ Closing check (with answers) (5 min)

Show: forward, forward, turn left, forward, forward, forward, forward. Ask: "Which cards can a repeat card replace, and which repeat card?" Children point or hold up the number with fingers.

Answer: the four forwards at the end become repeat 4, forward. The first two forwards could also become repeat 2, forward.

Then the one-to-one check, over the week (see the unit overview, Assessment).

Vocabulary

repeat, loop, the same again.

Differentiation

- **Support:** use only repeat 2 and repeat 3, and only with forward cards.
- **Stretch:** repeat a **group** of cards. Make a string loop round the four cards forward, turn right, forward, turn left and put a repeat 3 card in front. From (column 1, row 1) facing the back, this "staircase" ends on (column 4, row 4) facing the back.

⚠ Safety

- As before: walk, one square per card, stop before leaving the grid or touching a rock.
- A floor robot is used on the floor grid only, away from table edges.

Teacher notes

- A repeat card is a first step towards loops, which come back in grade 2 and in later computing units. Keep it to one repeated card for most children.
- In the one-to-one check, items 1 and 4 use the two filled-in program strips and a table grid; check that the toy faces away from the child at the start.

Grade 1 Computing: Picture Charts and Keeping Information Private (5 lessons, about 3 weeks)

> **Unreviewed AI draft.** Written by Claude Opus 5.5 (AI Village), 10 November 2026. No educator has reviewed or classroom-tested this unit. It is not aligned to any official standard, though it was written after reading four grade K-2 computing standards and one grade 1 maths standard (see Sources). Check the privacy and device notes against your school's rules.
>
> Text-checked by two other agents (Gemini 4 Argon, GLM-5.3 Flash); see Known gaps.

Back to the grade 1 year plan - Term 3

It follows Gemini 4 Argon's grade 1 term 1 unit, Symbols, Signals, Sorting, and Step-by-Step Recipes (sorting by one attribute at a time), and the term 2 unit, Programming a Floor Robot. Argon's grade 2 unit, Repeating Patterns, Treasure-Map Debugging, and Keeping Information Safe, builds on lessons 4 and 5.

This unit is "unplugged": it needs no device. The pictures come from Little Hill, the made-up village used in other grade 1 units (its park, its bakery and Duck from the arts term 3 story).

Big idea

Computers store and show information. When we sort and count things, we can show the numbers as a chart, and the same numbers can be shown in more than one way. Some information about a person is private: we keep it to ourselves, and a password is a secret that keeps things locked.

Outcomes

By the end of the unit, most children can:

1. Sort a set of picture cards into up to three groups and count each group.
2. Make a picture chart with one picture in each box, every column starting from the same line, and use it to say how many are in each group, how many altogether, and how many more are in one group than another.
3. Show the same numbers a second way (tally marks or cube towers), say that the answers match, and spot a chart that is not fair because its boxes are different sizes.
4. Spot a pattern in a chart and use it to make a prediction, saying that it is a guess, not a promise.
5. Say that where someone lives, their password, their phone number and pictures of them are private, and that if a screen or someone they do not know asks for these, they stop and ask a trusted grown-up.
6. Say what a password is for, keep a password secret, show that a longer picture password is harder to guess, and show how to log off.

Prior knowledge

- Grade 1 term 1 computing: sorting objects by one attribute at a time.
- Counting to 20 and writing numbers to 10 (grade 1 maths term 1). Comparing two numbers ("more", "fewer") from kindergarten.
- Health and PE term 3 (lesson 4): "stop and tell a grown-up" for things children are not sure about. Lesson 4 here uses the same idea.
- Children do not need to read: every card is a picture.

Lessons

Lesson	Title	Time	Core idea
1	Sort, count, chart	30 min	Sorting picture cards into three groups and putting them on a picture chart
2	The same numbers, three ways	30 min	Picture chart, tally marks and cube towers tell the same story; spotting an unfair chart
3	Our class vote and the bakery pattern	30 min	Collecting a class vote without names; reading a pattern to make a prediction
4	Is it private?	30 min	Information that is private; stop and ask a trusted grown-up
5	Picture passwords and logging off	30 min	Passwords keep things locked; longer is harder to guess; logging off; one-to-one check

Timings: lessons 1-4 are 30 minutes (5 + 17 + 4 + 4); lesson 5 is 30 minutes (3 + 15 + 7 + 5, with the check spread over the week).

The picture chart

- **Chart grid:** a large sheet with three columns, each ten boxes high, all boxes the same size (about 8 cm square). A thick **start line** runs along the bottom. Under each column is a label space for a picture card (not only a word).
- **Rule:** one picture in each box, starting from the box just above the start line, with no gaps. This makes columns that can be compared by looking.
- **Small chart grids:** the same layout on paper (boxes about 4 cm), one per pair, for lessons 1, 2 and the check.

Materials

Whole class unless noted.

- The large chart grid (above) and one small chart grid per pair.
- **Garden card set** (lessons 1 and 2): one per pair, 16 small picture cards from the Little Hill park: 7 bees, 4 ladybirds and 5 snails. Each kind has a different outline as well as a different colour (a bee is striped, a ladybird is round with spots, a snail has a spiral shell). One large set for the teacher.
- **Three label cards** (bee, ladybird, snail), large and small sets.
- **Interlocking cubes:** 16 per pair (any colours; lesson 2).
- Whiteboard and pens for tally marks.
- **The unfair chart** (lesson 2): a poster showing pond cards made by the teacher, 14 pond picture cards (6 frogs, 5 ducks, 3 fish) and a spare chart grid to fix it (see lesson 2).

- **Two small closing-check charts** (lesson 2): frogs 6, ducks 5, fish 3; and frogs 5, ducks 6, fish 3.
- **Story choice** (lesson 3): three picture books from the class library (the teacher chooses) and their covers as label cards; one cube per child; a "not voting today" spot.
- **Little Hill bakery chart** (lesson 3): a poster (see lesson 3).
- **Duck puppet** (lesson 4): a stick puppet or picture of Duck from the arts term 3 story, or any class puppet named Duck.
- **Duck's question cards** (lesson 4): eight cards, plus two closing-check cards (see lesson 4).
- **"OK" and "Stop and ask a grown-up" cards** (lesson 4): one of each per child (a green tick and a red hand; the pictures matter, not the colours).
- **Locked box** (lesson 5): a box with a lid and a paper "lock" sign; something fun inside (for example a class sticker sheet or the next story book).
- **Password picture cards** (lesson 5): four large cards (sun, moon, star, cloud), plus a small set of the four per pair.
- **Paper tablet** (lessons 4 and 5): one for the teacher in lesson 4; in lesson 5 one per pair, an A4 card drawn as a tablet screen with a "log off" button picture (a door with an arrow).
- **Check things** (lesson 5): a small chart grid, a 9-card garden set (4 bees, 2 ladybirds, 3 snails), the check chart (bees 6, ladybirds 3, snails 5) and two tally pictures (6, 3, 5 and 6, 5, 3; see lesson 5), the bakery chart, question cards 1 and 2 from lesson 4, the four small password cards, a paper tablet, and the record sheet (class list with six columns).
- Optional, for blind or low-vision children: tactile garden cards (raised outlines: stripes, spots, a spiral), a chart grid with raised lines and a raised start line, cubes for every chart, and tactile password cards (a raised circle for the sun, a crescent, a star shape and a cloud shape).
- Ear defenders and a quiet spot, as in every unit.

⚠ Safety and access

- **Privacy first.** Never ask children for their real address, phone number, full name with school, passwords, or photographs, in any lesson, even as an example. All examples are Duck's. The class vote uses cubes, not names.
- **Passwords are practice only.** The picture passwords in lesson 5 are for the locked box game. Never ask children to tell or write a real password. If a child says a real password aloud, say kindly, "That's a secret one: let's keep it secret," and move on; tell the family if the school's policy says so.
- **Disclosure.** If anything a child says during lesson 4 suggests they are at risk online or offline, follow the school's safeguarding procedure.
- **Voting is opt-in.** "Not voting today" is always fine and is never commented on.
- **Cubes:** small cubes can be a choking hazard for children who mouth objects; use large cubes or picture cards for those children.
- **Every way of answering counts:** pointing, holding up a card, eye-gaze, sign, AAC or speech. Children never need to read or write words.
- **Blind or low-vision children:** tactile cards and raised charts (above); cube towers let children compare by touch. A partner can describe a chart column by column.
- **Colour vision:** every card differs by shape as well as colour; the "OK" and "Stop" cards use a tick and a hand.
- **Deaf or hard-of-hearing children:** picture cards and pointing carry every task; in lesson 5 the password keeper shows the secret to the teacher instead of whispering.

Assessment

Lesson 5 includes a six-part check, done one-to-one (about 5 minutes each child; spread over the week). Record each part as **yes**, **with help** or **not yet**:

1. **Sort and chart** (outcomes 1 and 2): the child sorts the 9-card set (4 bees, 2 ladybirds, 3 snails) onto a small chart grid. (Yes if each kind is in its own column, one card per box, from the start line.)
2. **Read the chart** (outcome 2): using the chart the child just made: "Which has the most?" (bees) "How many altogether?" (9) "How many more bees than ladybirds?" (2)
3. **Same numbers** (outcome 3): show the check chart (bees 6, ladybirds 3, snails 5) and two tally pictures, one matching (6, 3, 5) and one not (6, 5, 3). The child points to the matching tally.
4. **Pattern** (outcome 4): on the bakery chart, "Next Wednesday, will the bakery sell a lot of loaves or a few?" (A lot; yes if the child points to the market-day columns or says Wednesdays are always busy. A child who adds "but it's a guess" shows the whole idea.)
5. **Private** (outcome 5): question cards 1 (pick a colour for Duck's car: OK) and 2 (type Duck's home address: stop and ask a grown-up).
6. **Passwords and logging off** (outcome 6): "Duck's friend asks for Duck's password. What should Duck do?" (Keep it secret; tell only a trusted grown-up.) "Which is harder to guess: a one-picture password or a three-picture password?" (three). Then the child points to the log off button on the paper tablet.

Doing it in their own way (pointing, moving cards, the child's usual access) counts as **yes**. "Not yet" is normal for some children; data and privacy return in grade 2.

Sources

- Computer Science Teachers Association, CSTA K-12 Computer Science Standards (2017), progression chart: https://csteachers.org/wp-content/uploads/2025/06/2017-csta-k-12-standards-progression-chart.pdf (read 10 November 2026). Level 1A (ages 5-7) standards used:
  - 1A-DA-06: "Collect and present the same data in various visual formats." (Lessons 1-3.)
  - 1A-DA-07: "Identify and describe patterns in data visualizations, such as charts or graphs, to make predictions." (Lesson 3.)
  - 1A-NI-04: "Explain what passwords are and why we use them, and use strong passwords to protect devices and information from unauthorized access." (Lesson 5.)
  - 1A-IC-18: "Keep login information private, and log off of devices appropriately." (Lessons 4 and 5.)
- Common Core State Standards for Mathematics, grade 1, Measurement and Data, 1.MD.C.4: "Organize, represent, and interpret data with up to three categories; ask and answer questions about the total number of data points, how many in each category, and how many more or less are in one category than in another." https://web.archive.org/web/2025/https://www.thecorestandards.org/Math/Content/1/MD/ (read 10 November 2026). This is why every chart in lessons 1, 2 and the check has three groups.
- The garden cards, the pond cards, the bakery numbers and Duck's question cards are examples made up for this unit, not from a source.

Known gaps

- Written 10 November 2026. Text-checked the same day by Gemini 4 Argon (links, timings, counts and arithmetic, source quotes, access and privacy safeguards; one capital-letter typo) and GLM-5.3 Flash (timings, counts, materials, sources fetched and quotes matched, privacy and safety; one optional note, that the pairs' three-picture password should also use different pictures). Both were fixed in 312d186, which Gemini 4 Argon also diff-checked the same day (see PEERCHECKS.md). Not yet checked by a teacher.
- **Standards read, not aligned.** Four CSTA level 1A lines and one Common Core line were read and quoted; the unit was not checked against them by anyone else, and other level 1A data and privacy lines (for example 1A-DA-05, storing and changing data on a device) are not covered.
- **No device.** Logging off is practised on a paper tablet. Schools with devices should practise their own real log-off steps.
- **The bakery chart has five days**, more than the three groups in 1.MD.C.4. It is used only for spotting a pattern, not for "how many more" questions.
- **"Private" is kept simple.** Grade 2 sorts information into more careful groups (sharing with permission); this unit only separates "OK" from "stop and ask a grown-up".
- **The password game is a model.** Real passwords for young children are usually set up by adults; the game is about why longer passwords are harder to guess, not a password-making lesson.

Lesson 1: Sort, count, chart

Unit overview - Next: Lesson 2: The same numbers, three ways

Goal: Children sort a set of picture cards into three groups, count each group and put the cards on a picture chart, then answer questions from the chart.

Length: about 30 minutes.

Materials

- The large chart grid with its start line, and the teacher's large garden card set (7 bees, 4 ladybirds, 5 snails) with the three large label cards.
- One garden card set (16 cards), one small chart grid and one set of small label cards per pair.
- Optional, for blind or low-vision children: tactile garden cards and a chart grid with raised lines and a raised start line.
- Ear defenders and a quiet spot.

Steps

✎ Opening (5 min)

Show the large garden cards all jumbled on the floor. "These are the little animals Hen saw in the Little Hill park today. Hen wants to know: which did she see the most of? It's hard to tell when they're all mixed up. How could we find out?" Take ideas (sort them, count them).

Remind children of term 1: "We sorted things by **one rule at a time**. Today our rule is: what kind of animal is it?"

✎ Main activity (17 min)

1. **Sort (3 min).** With the class, sort the large cards into three piles: bees, ladybirds, snails. Put a label card by each pile.
2. **Count (3 min).** Count each pile together, touching each card. (Bees 7, ladybirds 4, snails 5.)
3. **Chart (4 min).** Put the label cards under the three columns of the large chart grid. Show the rule: "One card in each box. Start just above the start line. No gaps." Volunteers place the cards. Ask: "Can you see which column is tallest without counting?" (Bees.)
4. **Pairs (7 min).** Each pair sorts its own garden set and makes the same chart on a small grid. Walk round and check the rule: one card per box, from the start line, no gaps. Pairs who finish ask each other a question from the chart ("Which has the fewest?").

✓ Discussion questions (with answers) (4 min)

- "How many bees? Ladybirds? Snails?" (7, 4, 5.)

- "How many animals did Hen see altogether?" (16. Count all the cards, or count on: 7, then 4 more is 11, then 5 more is 16.)
- "Which did she see the most of? The fewest?" (Bees; ladybirds.)
- "How many more bees than ladybirds?" (3. Look across: the bee column has 3 boxes above the top ladybird.)
- "Why do we start every column on the start line?" (So the tallest column really has the most; then we can compare by looking.)

✓ Closing check (with answers) (4 min)

Take two snails off the large chart. "Now which has the fewest?" Children point.

Answer: snails (3 snails; ladybirds still have 4). Put the snails back.

Vocabulary

sort, group, count, chart, column, most, fewest, altogether, how many more.

Differentiation

- **Support:** the child sorts one kind at a time ("Find all the bees"), then places them on the chart while counting aloud with an adult.
- **Stretch:** "If Hen sees 2 more ladybirds, which column will be taller, ladybirds or snails?" (Ladybirds: 6 is more than 5.)

⚠ Safety

- Keep cards flat on tables or the floor; no throwing.
- Pairs sit so that everyone can see and reach the chart; a child using a wheelchair works at a table chart.

Teacher notes

- These numbers are an example made up for this unit, not from a source. The same garden set is used again in lesson 2.
- A **picture chart** (sometimes called a pictogram or pictograph) uses one picture for one thing at this age. Do not use one picture for two things yet.

Lesson 2: The same numbers, three ways

Unit overview - Previous: Lesson 1: Sort, count, chart - Next: Lesson 3: Our class vote and the bakery pattern

Goal: Children show the same numbers as a picture chart, tally marks and cube towers, say that all three give the same answers, and spot a chart that is not fair.

Length: about 30 minutes.

Materials

- The large chart grid with lesson 1's garden chart still on it (7 bees, 4 ladybirds, 5 snails), and the three large label cards.
- Whiteboard and pens.
- One garden card set, one set of small label cards and 16 interlocking cubes per pair.
- **The unfair chart:** a poster of pond cards made by the teacher before the lesson. It has three columns standing on one line, labelled with a frog, a duck and a fish. The **6 frogs** are drawn small and squashed close together, so their column is short. The **5 ducks** are drawn big with gaps between them, so their column is the tallest. The **3 fish** are drawn medium-sized and close together. Make the frog column clearly shorter than the duck column.
- The large chart grid's spare copy (or the back of the large grid) for fixing the unfair chart, and 14 pond picture cards (6 frogs, 5 ducks, 3 fish) the size of the chart boxes.
- **Two small closing-check charts** (made before the lesson): frogs 6, ducks 5, fish 3; and frogs 5, ducks 6, fish 3, drawn with same-sized pictures in boxes from a start line.
- Optional, for blind or low-vision children: tactile garden cards; cubes; tactile versions of the two closing-check charts (raised shapes in boxes); a tactile version of the unfair chart (raised shapes: small ones close together for frogs, big ones spread out for ducks).
- Ear defenders and a quiet spot.

Steps

✎ Opening (5 min)

Point to yesterday's garden chart. "Our chart shows Hen's animals with pictures. Computers can show the same numbers in different ways, too. Today we'll show these numbers two more ways and see if they tell the same story."

✎ Main activity (17 min)

1. **Tally marks (5 min).** On the whiteboard, write the three animal pictures in a row. "A tally mark is one line for one thing. When we get to five, the fifth line goes across the other four, like a gate. Then it's quick to count in fives." Make the tallies together, touching each card on the chart: bees (a gate and 2 lines: 7), ladybirds (4 lines), snails (one gate: 5).
2. **Cube towers (6 min).** Pairs build one tower for each animal, one cube for each card, and stand each tower on its label card. Ask: "Which tower is tallest?" (Bees.) "Line the

towers up side by side. How many cubes taller is the bee tower than the ladybird tower?" (3.)

3. **The unfair chart (6 min).** Show the pond poster. "The Little Hill pond had frogs, ducks and fish. Which column is tallest? So are there the most ducks?" Let children say yes. Then count each column together: frogs 6, ducks 5, fish 3. "There are more frogs than ducks! Why did the duck column look tallest?" (The ducks are drawn bigger and with gaps; the frogs are small and squashed.) Fix it: volunteers put the 14 pond cards on a chart grid, one in each box from the start line. Now the frog column is tallest.

✓ Discussion questions (with answers) (4 min)

- "Do the picture chart, the tallies and the cube towers give the same answers?" (Yes: 7 bees, 4 ladybirds, 5 snails in all three. They are the same numbers shown three ways.)
- "Which way is quickest for counting? For seeing which is biggest?" (Any reason counts. Tallies are quick to count in fives; towers and picture charts are quick for seeing the tallest.)
- "What made the pond chart unfair?" (Pictures of different sizes and gaps, so a taller column did not mean more.)
- "What makes a chart fair?" (Same-size boxes, one picture in each, every column starting on the start line, no gaps.)

✓ Closing check (with answers) (4 min)

Draw on the whiteboard three tallies under the frog, duck and fish pictures: one gate and 1 line, 5 lines drawn as a gate, and 3 lines. "Which picture chart matches?" Hold up two small charts: one showing frogs 6, ducks 5, fish 3; one showing frogs 5, ducks 6, fish 3. Children point.

Answer: the first chart (frogs 6, ducks 5, fish 3).

Vocabulary

tally, tally mark, gate (a group of five), tower, the same numbers, fair, unfair.

Differentiation

- **Support:** the child builds the cube towers by placing one cube on each picture card in a column first, then stacking them.
- **Stretch:** the child draws their own tallies for the pond numbers and checks them against the fixed chart.

⚠ Safety

- Cubes stay on the tables. For children who mouth small objects, use large cubes or skip the towers and use the picture chart.

Teacher notes

- The pond numbers and the poster are an example made up for this unit, not from a source.
- The link to fairness is deliberate: children met "fair" and "not fair" set-ups in health and PE this term, and a chart can be unfair too.
- Make the two closing-check charts before the lesson (small chart grids with pond cards or drawings).

Lesson 3: Our class vote and the bakery pattern

Unit overview - Previous: Lesson 2: The same numbers, three ways - Next: Lesson 4: Is it private?

Goal: Children collect a class vote without names, show it as a cube-tower chart and answer questions about it, then use a pattern in a chart to make a prediction.

Length: about 30 minutes.

Materials

- Three picture books from the class library (the teacher chooses) and a cover picture of each as a label card.
- The large chart grid (cleared), with the three cover cards as labels.
- One cube per child, in a tray, and a "not voting today" spot (a mat or tray beside the chart).
- **The Little Hill bakery chart:** a poster made before the lesson (see below).
- Whiteboard and pens.
- Optional, for blind or low-vision children: tactile cover cards (a different raised shape stuck on each) and a raised-line copy of the bakery chart with a raised marker above each Wednesday.
- Ear defenders and a quiet spot.

The bakery chart. Ten columns in a row, one for each school day over two weeks (Monday to Friday, then Monday to Friday again), each ten boxes high with a start line. Under each column is the day's picture. Each loaf picture is one loaf sold. A small market-stall picture sits above both Wednesdays, because Wednesday is market day in Little Hill.

	Mon	Tue	Wed	Thu	Fri
Week 1	3	4	9	4	5
Week 2	4	3	8	5	4

Steps

✎ Opening (5 min)

Hold up the three books. "Tomorrow I'll read one of these. Let's collect information to choose: a vote. Each person who wants to vote puts one cube on the book they'd like. We don't write names: a vote is just a cube. If you'd rather not vote today, put your cube on the 'not voting' spot. That's fine."

✎ Main activity (17 min)

1. **Collect (5 min).** Children come up a table at a time and stand their cube in the column of the book they choose, building a tower from the start line. (A partner or adult places the cube for any child who points or signs a choice.)

2. **Read the chart (5 min).** Count each tower together and write the numbers under it on the whiteboard. Ask: "Which book got the most votes? How many people voted altogether? How many more votes did the top book get than the next one?" Count the cubes on the "not voting" spot only to check the total if children ask; never say who.
3. **If two books tie:** "We'll read one tomorrow and one the day after."
4. **The bakery pattern (7 min).** Show the bakery chart. "This chart shows how many loaves the Little Hill bakery sold each day for two weeks." Let children look. "What do you notice?" Take ideas, then point to both Wednesdays: "What's special about Wednesday?" (Lots of loaves; the most each week. It is market day.) "Next week, on Wednesday, will the bakery sell lots of loaves or only a few?" (Lots.) "Do we know for sure?" (No. It's a guess from the pattern; for example, rain might keep people at home.)

✓ Discussion questions (with answers) (4 min)

- "Why didn't we write names on the vote?" (We only need to know how many chose each book, not who. Who voted for what can stay private.)
- "How did the chart help us choose?" (The tallest tower showed the most votes without counting everyone again.)
- "What pattern did the bakery chart show?" (Every Wednesday has the tallest column; other days are about the same.)
- "Is a prediction a promise?" (No. It's our best guess from the pattern.)

✓ Closing check (with answers) (4 min)

"The bakery is going to bake extra loaves on one day next week. Point to the day it should choose." Children point to a Wednesday column, and say or sign why.

Answer: Wednesday (market day; the most loaves both weeks). Any child who adds "but it might be different" gets a big thumbs-up.

Vocabulary

vote, collect, information, data, pattern, predict, prediction, guess.

Differentiation

- **Support:** an adult points along the bakery chart's Wednesdays with the child: "Tall... tall."
- **Stretch:** "How many loaves did the bakery sell on the two Wednesdays altogether?" (17.)

⚠ Safety

- Children come to the chart a table at a time to avoid crowding.
- The vote is about books, never about people. Do not run votes on who is liked, who is a best friend, or anything about children's homes or bodies.

Teacher notes

- The bakery numbers are an example made up for this unit, not from a source. Little Hill's bakery is on the map in the society term 3 unit.
- The word **data** can be used for "the information we collected", but children do not need to learn it yet.
- Read the winning book the next day; the vote should count.

Lesson 4: Is it private?

Unit overview - Previous: Lesson 3: Our class vote and the bakery pattern - Next: Lesson 5: Picture passwords and logging off

Goal: Children say which information about someone is private, and practise "stop and ask a trusted grown-up" when a screen or someone they do not know asks for it.

Length: about 30 minutes.

Materials

- The Duck puppet (or a picture of Duck).
- Duck's eight question cards (below), large, each with a simple picture.
- One "OK" card (green tick) and one "Stop and ask a grown-up" card (red hand) per child.
- A paper tablet (one, for the teacher) to hold up when a card says "a game" or "a message".
- **Two closing-check cards** (made before the lesson): "A game asks Duck what its favourite animal is for the game's character" and "A message asks Duck where it lives."
- Optional, for blind or low-vision children: "OK" and "Stop" cards with raised shapes (a raised tick; a raised hand outline) and a partner who reads each question card aloud.
- Ear defenders and a quiet spot.

Duck's question cards (an example, not from a source):

1. A game on the library tablet asks Duck to pick a colour for its race car.
2. A game asks Duck to type its home address.
3. A message from someone Duck doesn't know says, "What's your password?"
4. Duck's teacher asks the class to vote for a story with cubes.
5. A game asks Duck to type its whole name and the name of its school.
6. A website asks Duck to take a photo of itself.
7. Hen asks Duck which picture in a book it likes best.
8. A box pops up: "You won a prize! Type your phone number."

Answer guide: **OK** for 1, 4 and 7; **Stop and ask a grown-up** for 2, 3, 5, 6 and 8.

Steps

✎ Opening (5 min)

Bring out Duck. "Duck has been playing games on the Little Hill library tablet. Some games ask questions. Some questions are fine to answer. Some are about things that are **private**: things we keep to ourselves and our trusted grown-ups."

Remind children of health and PE: "When we found tablets we weren't sure about, what did we do?" (Don't touch, tell a grown-up.) "With private information, it's the same idea: **stop and ask a grown-up**."

✎ Main activity (17 min)

1. **What is private? (5 min).** Say and show: "Private things about a person include: where they live, their phone number, their password, and photos of them. A computer can keep information for a long time and can send it far away, so we are careful what we type or say to a screen." Ask: "Who are trusted grown-ups you could ask at school?" (Children point to staff pictures or name staff; never ask about home.)
2. **Duck's cards (8 min).** Read each card while holding up the paper tablet for the screen ones. Every child holds up "OK" or "Stop". Talk briefly after each: "Why?" (Use the answer guide. For 3: "A password is a secret. Duck never tells it to someone it doesn't know." For 8: "Prizes that ask for private information are tricks.")
3. **Pairs: what should Duck say? (4 min).** Each pair takes card 2, 5 or 8 and acts it with the paper tablet: one is Duck, one holds the tablet. Duck shows "Stop" and says or signs, "I need to ask a grown-up," then goes to the teacher.

✓ Discussion questions (with answers) (4 min)

- "What does private mean?" (Something we keep to ourselves and our trusted grown-ups.)
- "Name one thing that's private." (Where someone lives, their phone number, their password, photos of them. Any one counts.)
- "Why was card 1 OK?" (Choosing a car colour doesn't tell anyone about Duck.)
- "What if Duck isn't sure?" (Stop and ask a grown-up. Asking is never wrong.)

✓ Closing check (with answers) (4 min)

Show two new cards: "A game asks Duck what its favourite animal is for the game's character" and "A message asks Duck where it lives." Children hold up "OK" or "Stop" for each.

Answers: favourite animal for the game character, OK; where Duck lives, Stop and ask a grown-up.

Vocabulary

private, information, password, trusted grown-up, stop and ask.

Differentiation

- **Support:** use only four cards (1, 2, 3, 7) and do them with an adult.
- **Stretch:** "Card 6 asked for a photo. What if Duck's grown-up says it's fine?" (Then it's OK, because a trusted grown-up decided.)

⚠ Safety

- **Never ask children for their own private information**, even as an example, and never ask what games or devices they use at home. All examples are Duck's.

- Keep the tone calm: the aim is "stop and ask", not fear. Do not tell frightening stories about strangers.
- If a child says something that suggests they are at risk online or offline, follow the school's safeguarding procedure.

Teacher notes

- Duck's cards and the closing-check cards are examples made up for this unit, not from a source. They follow CSTA 1A-IC-18, "Keep login information private" (see the unit overview).
- Make the two closing-check cards before the lesson.
- Grade 2 (Argon's unit, lesson 5) sorts information more carefully, including sharing with permission. This lesson keeps to two answers.

Lesson 5: Picture passwords and logging off

Unit overview - Previous: Lesson 4: Is it private?

Goal: Children say what a password is for, keep a password secret, find out by playing that a longer picture password is harder to guess, and practise logging off. The one-to-one check for the whole unit starts this week.

Length: about 30 minutes (the one-to-one check is extra, spread over the week).

Materials

- **The locked box:** a box with a lid and a paper "lock" sign on the front. Put something fun inside before the lesson (for example a class sticker sheet or the next story book).
- **Four large password picture cards:** sun, moon, star, cloud. Each has a different outline as well as a different colour.
- One small set of the four password cards per pair (one sun, one moon, one star, one cloud).
- One **paper tablet** per pair: an A4 card drawn as a tablet screen with a "log off" button picture (a door with an arrow pointing out of it).
- Whiteboard and pens.
- Optional, for blind or low-vision children: tactile password cards (a raised circle for the sun, a crescent, a star shape and a cloud shape) and a paper tablet with a raised "log off" button.
- Ear defenders and a quiet spot.
- **For the one-to-one check** (see the unit overview, Assessment): a small chart grid; a 9-card garden set (4 bees, 2 ladybirds, 3 snails); the check chart (a small chart showing bees 6, ladybirds 3, snails 5); two tally pictures (one showing 6, 3, 5 and one showing 6, 5, 3, each under a bee, a ladybird and a snail picture); the bakery chart from lesson 3; Duck's question cards 1 and 2 from lesson 4; the four small password cards; a paper tablet; and the record sheet (class list with six columns).

Steps

✎ Opening (3 min)

Show the locked box. "Something fun is inside. But the box has a lock. To open it, you need the secret: the **password**." Ask: "What is a password for?" (It keeps something locked so only the right people can open it.) "Tablets and computers have passwords too, to keep our work and our information safe."

✎ Main activity (15 min)

1. **Round 1: a one-picture password (3 min).** Hold up the four large cards. The teacher secretly chooses one picture as the box's password and puts it face down on top of the box. Children take turns guessing by pointing to one large card. After each guess, the teacher turns the guessed card over or says "Not that one." "How many guesses did we need?" (Never more than four, because there are only four pictures.)

Open the box and show what is inside, then close it again. "A one-picture password is very easy to guess."

2. **Round 2: a three-picture password (6 min).** "Let's make the password longer." The teacher secretly lays three **different** pictures in a row, in order, face down (for example star, sun, cloud). To open the box, the guessers must say all three in the right order. Give the class five guesses. Most classes will not find it. "Why was this one so hard?" (There are many more ways to put three pictures in a row.) The teacher can add: "I counted: there are 24 different ways to put three of our four pictures in a row, and only 4 ways with one picture." Children do not need to count this themselves. Open the box for everyone at the end.

3. **Pairs play (4 min).** Each pair has a small set of four password cards. One child is the **keeper**: they choose a password of one picture, then (in a second go) three **different** pictures, and **show it to the teacher**, never whispering it to a friend. The teacher looks, nods and keeps it secret. The partner guesses by pointing or laying cards in a row. Swap roles if there is time.

4. **The log-off routine (2 min).** Each pair holds their paper tablet. "When you have finished on a tablet or computer, you **log off**, then put it away." Together: "Finished? Log off, put it away." Children press the door-with-an-arrow button with a finger and put the paper tablet in the box or tray. If the school has class tablets or computers, practise logging off on a real device in a later session with the class's usual login.

✓ Discussion questions (with answers) (7 min)

- "Who can know your password?" (Only you and a trusted grown-up, such as your teacher or a grown-up at home who helps you.)
- "Duck's friend says, 'Tell me your password, I'm your friend!' What should Duck do?" (Keep it secret. Friends don't need it. Duck can tell a trusted grown-up.)
- "Which password was harder to guess, one picture or three pictures?" (Three pictures.)
- "Why do we log off when we finish?" (So the next person can't use our work or see our information.)
- "Did today's game use a real password?" (No, it was a practice password for the box.)

✓ Closing check (with answers) (5 min)

Show two large cards: the sun on its own, and a row of three (moon, cloud, star). "Which password is harder to guess?" Children point. Then: "Duck has finished playing a game on the class tablet. What does Duck do?" Children point to the log off button on their paper tablet (or say or sign "log off").

Answers: the three-picture password is harder to guess; Duck logs off (and puts the tablet away).

Vocabulary

password, secret, lock, guess, log off, private, trusted grown-up.

Differentiation

- **Support:** play round 1 with two pictures first. Children who find guessing in order hard can guess one picture at a time with the teacher saying "yes" or "not that one" for each place (this makes it easier; say so).
- **Stretch:** "How many guesses could a one-picture password need, at most?" (4.) "Would a two-picture password be harder than one picture?" (Yes.) Children can try to list two-picture passwords with cards.
- **Access:** the keeper shows the password to the teacher, so no whispering or listening is needed. Children who do not use their hands can guess by eye-gaze or by saying or signing the picture names; a partner moves the cards. Blind or low-vision children use tactile cards and feel the order left to right.

⚠ Safety

- The passwords in this lesson are practice passwords for the box. Never ask children to say or write a real password (see the unit overview, Safety and access).
- The box holds nothing that can be grabbed and swallowed (no sweets, no small parts).
- The "who can know your password" answer names trusted grown-ups in general. Do not ask children who knows their passwords at home.

Teacher notes

- The password game is an example made up for this unit, not from a source. It follows CSTA 1A-NI-04 ("Explain what passwords are and why we use them, and use strong passwords to protect devices and information from unauthorized access") and 1A-IC-18 ("Keep login information private, and log off of devices appropriately"); see the unit overview.
- The 24 comes from counting: 4 choices for the first picture, 3 for the second and 2 for the third (4 x 3 x 2 = 24). If pictures could repeat, there would be 4 x 4 x 4 = 64. Children do not need either number.
- The one-to-one check (unit overview, Assessment) takes about 5 minutes per child. Do it while the class plays the password game or another quiet task during the week.

Unit: Repeating Patterns, Treasure-Map Debugging, and Keeping Information Safe (Computing, Grade 2)

> **Unreviewed AI draft.** Written by Gemini 4 Argon (2026-11-05). No human teacher has reviewed this unit. All content is CC0 1.0 Public Domain.

Overview

By Grade 2 (`ages 7–8`), students are ready to write longer unplugged programs on paper grids, compress repeating staircases and border paths using `REPEAT n TIMES` blocks, compare two different routes to find which one uses fewer steps, and learn an essential real-world computing habit: **which personal information must be kept private vs. when to check context and permission with a trusted adult before sharing creative work**.

Across five 35-minute unplugged lessons, Grade 2 students trace and write arrow programs on 4×4 treasure maps, spot and compress repeating patterns (`↑ → ↑ → ↑ →` becomes `REPEAT 3 TIMES: [ ↑ → ]`), compare short vs. long routes around map obstacles, debug paper treasure maps where one wrong instruction sends the explorer into a swamp or rock wall, and sort fictional character cards into **Check Permission & Context (Class Project Page with Adult & Creator Approval)** vs. **Private Personal Information (Never Post Publicly)** protected by unique, long, unpredictable secrets (passphrases).

Prior knowledge

- Students have experience with `First → Next → Last` ordering, symbols, and single-rule sorting (Symbols, signals, sorting, and step-by-step recipes, Grade 1), and can count and add whole numbers up to 20.
- Helpful cross-links:
- Solids, liquids and melting (Science, Grade 2) uses step-by-step observation logs and reversible/irreversible state changes.
- Maps, routes and places (Society, Grade 3) extends grid navigation with map legends and compass directions.
- Leads to: Inputs, outputs, loops, and debugging (Computing, Grade 3) and Computing & systems progression.

Learning goals

By the end of this unit, a student can:

1. Write and trace a cardinal-arrow program (`↑ North`, `↓ South`, `→ East`, `← West`) on a 4×4 grid to navigate from a Start square to a Treasure Chest while avoiding obstacle squares (Lesson 1).
2. Identify a repeating group of steps in a long arrow program and rewrite it using a `REPEAT n TIMES` block, counting how many arrows were saved (Lesson 2).
3. Compare two different valid algorithms that reach the same goal on a 4×4 map and determine which route takes fewer steps (Lesson 3).

4. Find and repair the exact step (**bug**) where a broken map program hits an obstacle or misses the treasure, and verify the repair by tracing from the start (Lesson 4).
5. Sort fictional character examples into **Check Permission & Context Before Sharing** (like a classroom drawing or project puzzle shared only with creator and trusted-adult permission) vs. **Private Personal Information** (like a home street address, phone number, schedule, or account password that must never be posted), and explain why a real passphrase must be unique, long, and unpredictable—and never copied from a printed example (Lesson 5).

Lesson sequence

Lesson	Title	Time	Core idea
Lesson 1	Treasure-Map Grids: Following and Writing Arrow Programs	35 min	A grid algorithm uses exact directional arrows (`↑`, `↓`, `→`, `←`) to reach a goal without hitting obstacles
Lesson 2	Staircases and Borders: Shrinking Programs with `REPEAT`	35 min	When steps repeat in a pattern (`↑ → ↑ → ↑ →`), a **`REPEAT n TIMES`** loop makes the program shorter and clearer
Lesson 3	Two Roads to the Treasure: Comparing Short and Long Routes	35 min	More than one algorithm can solve the same problem; we can compare routes by counting total steps
Lesson 4	Map Bug Detectives: Tracing and Fixing Broken Routes	35 min	**Debugging** means stepping through a map program one square at a time to find and fix the exact wrong arrow
Lesson 5	Public vs. Private: Keeping Personal Information Safe	35 min	Private personal details and passwords must never be posted publicly, and even creative work requires checking context and permission with a trusted adult

Materials for the whole unit

- Printed or hand-drawn 4×4 grid maps (one per student or pair; ASCII diagrams included in every lesson)
- Pencils, erasers, and large paper boat or cardboard tokens (`at least 5 cm wide`) to slide across grid squares (with pointing, verbal/dictated, or tactile card options available in every lesson)
- Green and Red crayons or markers (for Lesson 5's Public vs. Private sorting cards)

Assessment summary

Each lesson ends with a 3-question exit check with complete worked solutions (which students may complete in writing, by pointing, or orally/dictated). In Lesson 5, students complete a 3-question exit check covering a 4-option privacy sort, a passphrase comparison, and a 4×4 grid loop trace.

Sources

These references provide teacher background for the concepts in this Grade 2 unit. All links were checked to load on 2026-11-05.

- Algorithm (step-by-step problem-solving procedure): https://en.wikipedia.org/wiki/Algorithm — background for Lessons 1 and 3.
- For loop (count-controlled repetition in programming): https://en.wikipedia.org/wiki/For_loop — background for Lesson 2's `REPEAT n TIMES` compression.
- Debugging (tracing and fixing errors in a program): https://en.wikipedia.org/wiki/Debugging — background for Lesson 4's step-by-step map trace.
- Information privacy (distinguishing personal data that requires protection): https://en.wikipedia.org/wiki/Information_privacy — background for Lesson 5's private vs. public information sort.
- Password (secret word or phrase used to protect access): https://en.wikipedia.org/wiki/Password — background for Lesson 5's strong passphrase activity.
- Bell, T., Witten, I. H., & Fellows, M. (2015). CS Unplugged (general pedagogical reference for unplugged computing; all text, maps, and activities in this unit are original CC0 work): https://www.csunplugged.org/

Known gaps and cautions

- **Unreviewed AI draft:** This unit has not yet been reviewed by a human Grade 2 teacher or classroom-tested with 7-to-8-year-old students.
- **Cardinal Grid Arrows (`↑ ↓ → ←`) vs. Relative Robot Turns:** In Grade 2, map arrows move the token 1 square in the arrow's direction on the page (`↑` = 1 square Up/North on the map, `→` = 1 square Right/East on the map). Relative body turns (`TURN LEFT 90°` while keeping track of which way the robot's nose is pointing) are separated into Kindergarten physical floor play and Grade 3 Computing so Grade 2 students can focus cleanly on loop compression and route comparison on paper grids.
- **Privacy discussion sensitivity (Lesson 5):** During Lesson 5, remind students **never to say or write their real home address, real phone number, or real family passwords** out loud or on their worksheets. All sorting cards use fictional storybook characters (like "Barnaby Bear's cave address").

Lesson 1: Treasure-Map Grids — Following and Writing Arrow Programs

- **Unit:** Repeating Patterns, Treasure-Map Debugging, and Keeping Information Safe (Computing, Grade 2)
- **Time:** 35 minutes
- **Grouping:** Whole class and pairs

Learning objective

Students will trace and write a sequence of directional arrows (↑, ↓, →, ←) on a 4×4 grid map to move a ship token from a Start dock (S) to a Treasure Chest (T) without landing on Rock (#) squares.

Materials

- Whiteboard or chart showing the 4×4 Harbor Map below
- Per pair: printed 4×4 grid sheets, pencils, and a large paper boat or cardboard token (at least 5 cm wide, or tactile arrow cards)

Vocabulary

- **Grid Map:** A square map divided into rows and columns where each move steps into a neighboring square.
- **Program:** A list of arrow instructions written down in order for a computer or player to follow.
- **Obstacle (#):** A blocked square (like a sharp rock or swamp) that our token must steer around.

Lesson flow

✎ 1. Opening: Reading the 4 Arrow Commands (7 min)

1. Draw the four map arrows on the board:
2. ↑ **(Up / North):** Move 1 square up toward the top of the page.
3. ↓ **(Down / South):** Move 1 square down toward the bottom of the page.
4. → **(Right / East):** Move 1 square right.
5. ← **(Left / West):** Move 1 square left.
6. Draw this 4×4 Harbor Map on the board (S = Start Dock at bottom-left, T = Treasure Island at top-right, # = Sharp Rocks):

```
Row 4 (Top):    [ . ]  [ # ]  [ . ]  [ T ]
Row 3:          [ . ]  [ # ]  [ . ]  [ . ]
Row 2:          [ . ]  [ . ]  [ . ]  [ # ]
Row 1 (Bottom): [ S ]  [ # ]  [ . ]  [ . ]
                Col 1  Col 2  Col 3  Col 4
```

1. Ask: "Why can't our boat sail straight right from *[S]* (*Col 1, Row 1*) using → → →?"

2. Because `[Col 2, Row 1]` has a sharp Rock (`#`)!

✎ 2. Main activity: Tracing & Writing Safe Harbor Programs (18 min)

1. **Whole-Class Trace (6 min):** Test this 6-arrow program together by moving a magnet or drawing dots one square at a time from `[S]` (`Col 1, Row 1`): **Program A:** `↑ → → ↑ ↑ →`
2. Arrow 1 (`↑`): Moves from `(Col 1, Row 1)` to `(Col 1, Row 2)` — safe water `[.]`!
3. Arrow 2 (`→`): Moves to `(Col 2, Row 2)` — passes right between the two rocks!
4. Arrow 3 (`→`): Moves to `(Col 3, Row 2)` — safe water `[.]`!
5. Arrow 4 (`↑`): Moves to `(Col 3, Row 3)` — safe water `[.]`!
6. Arrow 5 (`↑`): Moves to `(Col 3, Row 4)` — safe water `[.]`!
7. Arrow 6 (`→`): Moves to `(Col 4, Row 4)` — lands right on **`[T]` Treasure**!
8. **Pair Map Designer & Navigator (12 min):**
9. Give each pair two blank 4×4 grids.
10. **Step 1:** Partner A places `S` in any corner, `T` in the opposite corner, and draws **3 Rocks (`#`)** (making sure at least one open water path still connects `S` to `T`!).
11. **Step 2:** Partner A writes the secret arrow program on the back of their sheet.
12. **Step 3:** Partner B creates their own arrow program (by writing arrows, arranging tactile arrow cards, or dictating directions to their partner), slides the large paper boat token (or points step by step) across Partner A's map, and checks that it reaches `T` without touching any `#`. Then partners switch!

✓ 3. Discussion questions (5 min)

- **Question 1:** "What happens if you place Rocks (`#`) all the way across Row 2 (*`[#][#][#][#]`*)?"
- Example good answer: It builds a solid wall across the whole map so **no** program can ever get from Row 1 to Row 4!
- **Question 2:** "Why is it helpful to touch each square with your pencil tip as you write each arrow?"
- Example good answer: If you try to write 6 arrows in your head without tracking which square the boat has reached, it is easy to lose count by 1 square and hit a rock.

✓ 4. Closing check (5 min)

Look at the 4×4 Harbor Map from the board (`S` at `Col 1, Row 1`):

1. If you start at `[S]` (`Col 1, Row 1`) and follow the program `↑ ↑ ↑`, which square (`Col, Row`) do you land on? Does that square have a Rock (`#`)?
2. From `(Col 1, Row 4)`, why can't you go `→` to reach the Treasure?
3. Write the **last 2 arrows** needed to go from `(Col 3, Row 3)` to the Treasure at `(Col 4, Row 4)`.

Worked answers for closing check

1. Starting at `(Col 1, Row 1)`, `↑ ↑ ↑` moves through `(Col 1, Row 2)` and `(Col 1, Row 3)` to land on **`(Col 1, Row 4)`** (the top-left corner). It is safe water (`[.]`), **no rock**.

2. Because `(Col 2, Row 4)` has a **Rock (#)** blocking the way!
3. ↑ → (go up to `(Col 3, Row 4)`, then right to `[T]` at `(Col 4, Row 4)`), **or** → ↑ (go right to `(Col 4, Row 3)`, then up to `[T]` at `(Col 4, Row 4)`—both squares are open!).

Differentiation

- **Extra support:** Let students move their large paper boat token (or point) 1 square first, record or state that 1 arrow immediately (in writing, by placing an arrow card, or verbally), and repeat until the token is sitting on `T`.
- **Extra challenge:** Add a **Gold Key (K)** square on the map (`Col 4, Row 1`) and require the boat to visit `K` before sailing to the locked Treasure Chest `T`!

Lesson 2: Staircases and Borders — Shrinking Programs with `REPEAT`

- **Unit:** Repeating Patterns, Treasure-Map Debugging, and Keeping Information Safe (Computing, Grade 2)
- **Time:** 35 minutes
- **Grouping:** Whole class and pairs

Learning objective

Students will spot repeating groups of arrows inside a grid program, rewrite them using a **`REPEAT n TIMES: [ ... ]`** loop block, and verify that the long program and the loop program land on the exact same square.

Materials

- 4×4 Staircase Map on the board and student worksheets

Vocabulary

- **Pattern Unit (Core):** The small group of steps that repeats in the same order.
- **`REPEAT n TIMES` (Loop):** A computing instruction that tells the player or computer to run the steps inside the brackets `[ ... ]` exactly `n` times in a row.

Lesson flow

✎ 1. Opening: The Staircase Climb (7 min)

1. Draw a 4×4 Staircase Map where the explorer starts at `S` (`Col 1, Row 1`) and climbs diagonal stairs to `T` (`Col 4, Row 4`):

```
Row 4 (Top):    [ . ]  [ . ]  [ . ]  [ T ]
Row 3:          [ . ]  [ . ]  [ * ]  [ * ]
Row 2:          [ . ]  [ * ]  [ * ]  [ . ]
Row 1 (Bottom): [ S ]  [ * ]  [ . ]  [ . ]
                Col 1  Col 2  Col 3  Col 4
```

1. Write the long 6-arrow staircase program along the `*` squares: **Long Program:** `→ ↑ → ↑ → ↑`
2. Ask: "Read or tap through that arrow list with me: 'Right, Up, Right, Up, Right, Up!' What 2-arrow pattern repeats over and over, and how many times does it repeat?"
3. Pattern: **`[ → ↑ ]`**, repeated **`3` times**!
4. Write the compressed version right underneath: **Loop Program:** **`REPEAT 3 TIMES: [ → ↑ ]`**
5. Count the arrows we had to write: the Long Program wrote **6 arrows**, while the Loop Program wrote only **2 arrows** inside the box (`6 − 2 = 4` arrows saved!).

✎ 2. Main activity: Unrolling & Compressing Loops (18 min)

1. **Practice Unrolling a Loop (6 min):**
2. What if a programmer hands you a loop and you want to trace every single step? You **unroll** it!
3. Write on the board: `REPEAT 3 TIMES: [ ↑ ]` followed by `REPEAT 3 TIMES: [ → ]`
4. Unroll it together on the 4×4 grid starting at `S` (`Col 1, Row 1`):
    - First loop (`REPEAT 3 TIMES: [ ↑ ]`) = `↑ ↑ ↑` (climbs straight up the left wall to `Col 1, Row 4`).
    - Second loop (`REPEAT 3 TIMES: [ → ]`) = `→ → →` (walks straight across the top row to `T` at `Col 4, Row 4`!).
5. **Pair Loop Compression Puzzles — Map-Free Symbolic Pattern Puzzles (12 min):** Explain that the next three puzzles are **map-free symbolic pattern puzzles** (we are compressing the repeating symbol sequence itself rather than tracing from `S` on the 4×4 Staircase Map above; students can write, point to, or dictate their answers, and for Puzzle A may tap/point/say the rhythm instead of clapping or jumping):
6. **Puzzle A (Rhythm Pattern — Clap/Tap/Point):** `Clap Clap Jump | Clap Clap Jump | Clap Clap Jump | Clap Clap Jump`
    - Answer: **`REPEAT 4 TIMES: [ Clap Clap Jump ]`** (12 actions shrunk to 1 loop of 3 actions!).
7. **Puzzle B (Symbolic Zig-Zag Arrow Sequence):** `↑ ← ↑ ← ↑ ←`
    - Answer: **`REPEAT 3 TIMES: [ ↑ ← ]`**.
8. **Puzzle C (Symbolic Loop + Extra Step at the End!):** `→ ↑ → ↑ → ↑ →` (7 arrows total)
    - Guide students to see that the first 6 arrows are `REPEAT 3 TIMES: [ → ↑ ]`, followed by one extra `→` outside the loop: **`REPEAT 3 TIMES: [ → ↑ ]` then `→`**.

✓ 3. Discussion questions (5 min)

- **Question 1:** "Does *`REPEAT 3 TIMES: [ → ↑ ]`* do the exact same thing as *`REPEAT 3 TIMES: [ → ]`* then *`REPEAT 3 TIMES: [ ↑ ]`*?"
- Example good answer: Both end up at the top-right corner (`Col 4, Row 4`) if there are no rocks, **but** they walk through completely different middle squares! The first climbs diagonal stairs (`→ ↑ → ↑ → ↑`), while the second walks all the way across the bottom row first (`→ → →`) and then climbs the right wall (`↑ ↑ ↑`).
- **Question 2:** "If a game programmer wants a guard character to walk 100 steps to the right, how does a *`REPEAT`* loop help them?"
- Example good answer: They can write `REPEAT 100 TIMES: [ → ]` in one line instead of drawing 100 separate arrows!

✓ 4. Closing check (5 min)

1. (Map-free symbolic unroll) Unroll this loop into a plain list of arrows: **`REPEAT 4 TIMES: [ ↓ → ]`**. How many total single-square steps does the unrolled list contain?
2. (Map-free symbolic compression) Compress this 8-arrow sequence using a `REPEAT` loop: `↑ ↑ → ← ↑ ↑ → ←`
3. (4×4 grid trace) Start at `(Col 1, Row 1)` on an open 4×4 grid. Where do you land after running **`REPEAT 2 TIMES: [ ↑ → ]`**?

Worked answers for closing check

1. Unrolled: `↓ → ↓ → ↓ → ↓ →`. Since there are `2` arrows inside the box repeated `4` times, the token takes `4 × 2 = 8` **total steps**.
2. **`REPEAT 2 TIMES: [ ↑ ↑ → ← ]`** (the 4-arrow pattern `↑ ↑ → ←` repeats twice).
3. Round 1 (`↑ →`) moves from `(Col 1, Row 1)` → `(Col 1, Row 2)` → `(Col 2, Row 2)`. Round 2 (`↑ →`) moves to `(Col 2, Row 3)` → **`(Col 3, Row 3)`**.

Differentiation

- **Extra support:** Have students draw vertical separator bars `|` between repeating groups (`→ ↑ | → ↑ | → ↑`) before counting how many groups there are.
- **Extra challenge:** Ask students how to write a loop that walks all the way around the outside perimeter of a 4×4 square grid and returns to `S` (`3 steps ↑, 3 steps →, 3 steps ↓, 3 steps ←`).

Lesson 3: Two Roads to the Treasure — Comparing Short and Long Routes

- **Unit:** Repeating Patterns, Treasure-Map Debugging, and Keeping Information Safe (Computing, Grade 2)
- **Time:** 35 minutes
- **Grouping:** Whole class and pairs

Learning objective

Students will trace two or more valid grid algorithms that reach the same goal, count the total steps taken by each route, and explain why finding a shorter route matters when energy or time is limited.

Materials

- Printed 4×4 Island Map worksheets (diagram below) and pencils

Vocabulary

- **Route (Path):** The sequence of squares a token visits from Start (`S`) to Treasure (`T`).
- **Step Count (Cost):** The total number of single-square moves (`↑`, `↓`, `→`, `←`) a route takes from start to finish.
- **Efficient:** Finishing the job correctly without wasting extra steps.

Lesson flow

✎ 1. Opening: Two Sailors, One Treasure Island (7 min)

1. Draw this 4×4 Island Map on the board (`S` at `Col 1, Row 2`; `T` at `Col 4, Row 3`; `#` = Reef Rocks):

```
Row 4 (Top):    [ . ]  [ . ]  [ . ]  [ . ]
Row 3:          [ . ]  [ # ]  [ . ]  [ T ]
Row 2:          [ S ]  [ # ]  [ . ]  [ . ]
Row 1 (Bottom): [ . ]  [ . ]  [ . ]  [ . ]
                Col 1  Col 2  Col 3  Col 4
```

1. Two sailors both steer around the Reef Rocks in Column 2 and reach `T` safely without hitting a rock:
2. **Sailor Ava's Program (North Pass via Row 4):** `↑ ↑ → → ↓ →`
3. **Sailor Ben's Program (South Pass via Row 1):** `↓ → → ↑ ↑ →`
4. Trace both on the board:
5. Do **both** sailors reach `T` safely? **Yes!**
6. Count the steps: Ava took **`6` steps**, and Ben took **`6` steps**—a tie!

✎ 2. Main activity: Finding the Shortest Safe Route (18 min)

1. **Why 6 Steps Is the Shortest Here—and How a Route Can Waste Extra Steps (6 min):**
2. Ask: "Can any sailor get from *S (Col 1, Row 2)* to *T (Col 4, Row 3)* in fewer than 6 steps without hitting the Reef Rocks at *(Col 2, Row 2)* and *(Col 2, Row 3)*?"
3. Check together: Because Column 2 is blocked on Rows 2 and 3, every safe route must detour up to Row 4 (like Ava) or down to Row 1 (like Ben: ↓ to Col 1, Row 1 → → Col 2, Row 1 → → Col 3, Row 1 → ↑ Col 3, Row 2 → ↑ Col 3, Row 3 → → Col 4, Row 3), so **6 steps is the shortest possible safe route** on this map.
4. Now test **Sailor Cora's Longer Scenic Route** (where Cora doesn't stop when she first reaches T on step 6): ↑ ↑ → → → ↓ ← →
5. Trace Cora's route (by sliding a large paper token, pointing, or numbering squares): She reaches T at (Col 4, Row 3) on step 6, keeps going by stepping backward off T to (Col 3, Row 3) on step 7 (←), and steps back onto T on step 8 (→)!
6. Cora took **8 steps** (2 wasted back-and-forth steps after already reaching T!).
7. **Pair Battery-Limit Challenge (12 min):** Give pairs a new 4×4 Submarine Map where the submarine only has **5 battery points** (1 battery point = 1 step):

```
Row 4 (Top):    [ . ]  [ . ]  [ # ]  [ T ]
Row 3:          [ . ]  [ . ]  [ . ]  [ . ]
Row 2:          [ # ]  [ # ]  [ . ]  [ # ]
Row 1 (Bottom): [ . ]  [ S ]  [ . ]  [ . ]
                Col 1  Col 2  Col 3  Col 4
```

- S is at **(Col 2, Row 1)**; T is at **(Col 4, Row 4)**.
- Pairs must:
    1. Trace **Route 1:** ← ↑ (hits rock at *Col 1, Row 2*—invalid!)
    2. Trace **Route 2:** → ↑ ↑ ← → → ↑ (steps through (Col 3, Row 1) → (Col 3, Row 2) → (Col 3, Row 3) → (Col 2, Row 3) → (Col 3, Row 3) → (Col 4, Row 3) → (Col 4, Row 4): reaches T safely without hitting any rocks, but takes **7 steps**—runs out of battery!).
    3. Find the **5-step Winner Route** that reaches T using **all 5 battery points with 0 wasted steps**:
        - From S (Col 2, Row 1): → (Col 3, Row 1) ↑ (Col 3, Row 2) ↑ (Col 3, Row 3) → (Col 4, Row 3) ↑ (T at Col 4, Row 4) — exactly **5 steps**!

✓ 3. Discussion questions (5 min)

- **Question 1:** "Can two different programs both be 100% correct even if they use different arrows?"
- Example good answer: Yes! As long as both programs start at S, avoid every obstacle #, and stop on T, both are valid algorithms.
- **Question 2:** "In our map game (where every square costs 1 step and 1 battery point), why does a shorter route help—and when might a real robot or delivery truck choose a longer path in the real world?"
- Example good answer: In our equal-cost grid model where every step costs the same 1 battery point, fewer steps save time and battery power. (In the real world, hills, rough

terrain, traffic, or sharp turns can have different energy costs, so a slightly longer smooth road can sometimes use less battery or be safer!)

✓ 4. Closing check (5 min)

1. On an **open 4×4 grid with no rocks**, starting at `S (Col 1, Row 1)` and aiming for `T (Col 4, Row 3)`, two valid programs both reach `T`:
2. **Program X:** `↑ ↑ → → →`
3. **Program Y:** `→ ↑ → ↓ ↑ ↑ →` How many steps does each program take, and which one is more **efficient** in this equal-step grid model?
4. Look closely at **Program Y** (`→ ↑ → ↓ ↑ ↑ →`). Which two arrows in the middle undo each other and waste 2 steps?
5. If a program takes only `2` steps (`→ →`) but lands on a sharp Rock (#), is it a better solution than a `6`-step program that safely reaches the Treasure (`T`)? Why or why not?

Worked answers for closing check

1. **Program X** takes `5` **steps**; **Program Y** takes `7` **steps**. **Program X** is more efficient because it reaches the goal in 2 fewer steps.
2. The `↓ ↑` in the middle steps down one square and immediately steps right back up to the same square!
3. **No!** A program must first be **correct and safe** (reaching `T` without crashing into #) before we care about how short it is.

Differentiation

- **Extra support:** Have students place a pencil tally mark, point step by step, or place a large paper token on each square as they visit it; any square that gets **two** tally marks shows where the route looped back on itself! Students may give responses in writing, by pointing, or verbally.
- **Extra challenge:** Ask pairs to compare the 5-step Submarine Winner Route (`→ ↑ ↑ → ↑`) against a 5-step attempt that tries to go all the way right on Row 1 first (`→ → ↑ ↑ ↑`—blocked on Step 3 by the rock at `Col 4, Row 2`!) or the valid 7-step Route 2 (`→ ↑ ↑ ← → → ↑`), and prove why `(Col 3, Row 2)` is a mandatory "bridge square" that every valid route on the Submarine Map must pass through.

Lesson 4: Map Bug Detectives — Tracing and Fixing Broken Routes

- **Unit:** Repeating Patterns, Treasure-Map Debugging, and Keeping Information Safe (Computing, Grade 2)
- **Time:** 35 minutes
- **Grouping:** Whole class and pairs

Learning objective

Students will step through a broken map program one instruction at a time, circle the exact arrow (**bug**) where the path hits an obstacle or goes the wrong way, and write a working repair.

Materials

- Printed 4×4 Bug Detective worksheets (diagram below) and colored pencils

Vocabulary

- **Bug:** An error in a program that causes it to crash into an obstacle or finish on the wrong square.
- **Trace:** Moving a finger or token step by step through a program while numbering each square (`1, 2, 3...`) to see what the program actually does.
- **Debug:** Finding the exact wrong arrow and fixing it.

Lesson flow

✎ 1. Opening: Why Guessing Doesn't Fix Bugs (7 min)

1. Put this 4×4 Swamp Map on the board (`S` = Start Camp at `Col 1, Row 1`; `T` = Treehouse at `Col 4, Row 4`; `#` = Mud Swamp):

```
Row 4 (Top):    [ . ]  [ . ]  [ . ]  [ T ]
Row 3:          [ . ]  [ # ]  [ # ]  [ . ]
Row 2:          [ . ]  [ . ]  [ . ]  [ . ]
Row 1 (Bottom): [ S ]  [ . ]  [ # ]  [ . ]
                Col 1  Col 2  Col 3  Col 4
```

1. A hiker wrote this 6-arrow program to reach the Treehouse `T`: **Broken Program:** `Step 1: ↑` | `Step 2: →` | `Step 3: →` | `Step 4: ↑` | `Step 5: →` | `Step 6: ↑`
2. Ask: "If we just look at the arrows from across the room, it has three `↑`s and three `→`s, so it looks like it should reach *`(Col 4, Row 4)`*! Why did the hiker get stuck in the Mud Swamp?"
3. **Trace and Number Each Step Together on the Map:**
4. Start at `S (Col 1, Row 1)`.
5. `Step 1 (↑)`: Lands on `(Col 1, Row 2)` — write `1` in that square (safe!).
6. `Step 2 (→)`: Lands on `(Col 2, Row 2)` — write `2` in that square (safe!).
7. `Step 3 (→)`: Lands on `(Col 3, Row 2)` — write `3` in that square (safe!).

8. `Step 4 (↑)`: Moves from `(Col 3, Row 2)` up to **`(Col 3, Row 3)`** — SPLAT! `(Col 3, Row 3)` is a **Mud Swamp (`#`)**!
9. Circle **`Step 4`** and **`Step 5`**: If we swap Step 4 and Step 5 so the hiker goes `→` **first** (to `Col 4, Row 2`) and **then** `↑ ↑` up Column 4, the hiker walks right around the swamp and reaches `T`!

✎ 2. Main activity: Three Bug-Hunter Cases on the Swamp Map (18 min)

Student pairs trace three broken programs on the Swamp Map above (using a pencil, pointing step by step, sliding a large paper token, or dictating to their partner), identify the step number where the bug happens, and record or state the repaired program:

1. **Case 1 — The Wrong-Direction Bug:**
2. Broken Program: `↑ ↑ ↑ → → ←`
3. Trace: Steps 1–3 (`↑ ↑ ↑`) climb Column 1 to `(Col 1, Row 4)`. Steps 4–5 (`→ →`) walk across Row 4 to `(Col 3, Row 4)`. Then **Step 6 (`←`)** steps backward to `(Col 2, Row 4)` instead of stepping onto `T`!
4. Bug Location: **Step 6 (`←`)**.
5. Fix: Change Step 6 from `←` to `→` (`↑ ↑ ↑ → → →`).
6. **Case 2 — The Loop Count Bug (Off-by-One):**
7. Broken Program: `REPEAT 2 TIMES: [ ↑ ]` then `REPEAT 3 TIMES: [ → ]`
8. Trace: `REPEAT 2 TIMES: [ ↑ ]` only climbs 2 squares from `(Col 1, Row 1)` to `(Col 1, Row 3)`. Then the first `→` of the second loop crashes straight into the swamp at `(Col 2, Row 3)`!
9. Bug Location: The first loop said **`REPEAT 2 TIMES`** instead of **`REPEAT 3 TIMES`**.
10. Fix: Change the first loop to **`REPEAT 3 TIMES: [ ↑ ]`** (so the hiker climbs all the way to Row 4 before walking 3 steps right to `T`).
11. **Case 3 — Partner Bug Swap (6 min):**
12. Each student writes a working 6-step program on a clean 4×4 map, intentionally changes **1 arrow** to create a bug, and trades with their partner to trace, circle, and fix.

✓ 3. Discussion questions (5 min)

- **Question 1:** "Why did numbering the squares *`1, 2, 3, 4...`* make it so much easier to fix the Swamp Map than erasing the whole program and starting over?"
- Example good answer: Because Steps `1, 2, 3` were already working! Numbering the squares showed us that only Step 4 and Step 5 needed to be swapped.
- **Question 2:** "In Case 2, what do computer scientists call a bug where a loop runs *`2`* times instead of *`3`* times?"
- Example good answer: An **off-by-one bug** (counting one too few or one too many).

✓ 4. Closing check (5 min)

On the Swamp Map (`S` at `Col 1, Row 1`; Swamp `#` at `Col 3, Row 1`; `T` at `Col 4, Row 4`):

1. A student writes the program: `→ → ↑ → ↑ ↑`. On which step number (`1, 2, 3, 4, 5, or 6`) does the hiker step into a Mud Swamp (`#`), and which square is it?
2. What is wrong with the program `↑ ↑ ↑ → →` on the Swamp Map? (Where does it stop?)

3. How do you fix `↑ ↑ ↑ → →` so it reaches `T` at `(Col 4, Row 4)`?

Worked answers for closing check

1. **Step 2 (→)!** Step 1 (→) moves safely to `(Col 2, Row 1)`, and **Step 2 (→)** steps right into the Mud Swamp `#` at **`(Col 3, Row 1)`**.
2. It stops 1 square short at **`(Col 3, Row 4)`** because it only has two → arrows at the end instead of three.
3. Add one more → at the end (`↑ ↑ ↑ → → →`, or `REPEAT 3 TIMES: [ ↑ ]` then `REPEAT 3 TIMES: [ → ]`).

Lesson 5: Public vs. Private — Keeping Personal Information Safe

- **Unit:** Repeating Patterns, Treasure-Map Debugging, and Keeping Information Safe (Computing, Grade 2)
- **Time:** 35 minutes
- **Grouping:** Whole class and pairs

Learning objective

Students will sort fictional character examples into **Private Personal Information** (which must never be posted publicly) vs. **Check Permission & Context Before Sharing** (items that may be shared on a class project page only after checking with a trusted adult and getting the creator's permission), and explain why a real passphrase must be unique, long, and unpredictable—and never copied from a printed example.

Materials

- Per pair (or one shared class set on the board): a set of 8 fictional "Barnaby Bear" sorting cards (listed below)
- Two sorting mats per pair:
- **Green (`Check Permission & Context — Okay for Barnaby's Class Page with Adult & Creator Approval`)**
- **Red (`Private Personal Information — Keep Protected, Never Post Publicly`)**

  Important privacy rule for teachers and students: Never ask students to share, say aloud, or write their real home addresses, phone numbers, or real family/school passwords during this lesson—not even to a teacher or classmate during an activity. All examples use our fictional storybook character, **Barnaby Bear**.

Vocabulary

- **Private Personal Information:** Details that identify where someone lives, how to contact them directly, where they will be at specific times, or how to unlock their accounts (such as a home street address, phone number, school schedule, or login password).
- **Context & Permission Check:** Before sharing even creative work, hobbies, or a project online, checking **who** can see it, **whether** it could identify a child when combined with other clues, and getting **permission from the creator and a trusted adult**. (Hanging a drawing inside our classroom is different from posting it on a public website!)
- **Password / Passphrase:** A secret string of words, letters, numbers, or symbols that protects an account. A strong secret is **unique, long, and unpredictable** (within the account's rules) and is never copied from a public book or worksheet.

Lesson flow

✎ 1. Opening: Hallway Bulletin Board vs. Public Website Fence (7 min)

1. Ask: "If our fictional character Barnaby Bear paints a picture of a purple dragon in art class, is hanging it inside the classroom different from posting it on a public website anyone in the world can see?"
2. Explain: **Yes—context and permission matter!** Inside the classroom, only classmates and teachers see it. On a public website, strangers can see it, and details like a child's name, nickname, or artwork can sometimes identify who made it or where they go to school. Before posting even creative work online, we always **ask the creator's permission and check with a trusted adult**.
3. Ask: "Would Barnaby Bear ever tape the key to his front door, his cave street address, or his family's phone number onto the outside of the school fence for strangers to take?" (`No!` That is **Private Personal Information** and must never be posted!)

✎ 2. Main activity: Helping Barnaby Bear Sort Cards & Choose Safe Passphrase Habits (18 min)

1. **Part A — Barnaby Bear's Webpage Sort (10 min):** Barnaby Bear's teacher is setting up a class project page about Solids, Liquids, and Melting. Each pair receives a set of the 8 fictional Barnaby Bear cards (or the class sorts a shared set together by pointing, speaking, or placing cards) onto their **Green (`Check Permission & Context – Okay with Adult & Creator Approval`)** and **Red (`Private – Never Post Publicly`)** mats:
2. **Card 1:** A photo of an ice-cube experiment chart with no student names or faces → **Green (Check Permission & Context — okay with teacher & creator approval)**
3. **Card 2:** Barnaby's exact home cave street address (`42 Honeybee Lane, Apartment 3B`) → **Red (Private — Never Post)**
4. **Card 3:** A class poll tally showing how many students voted for different states of matter (`Solid, Liquid, Gas`) → **Green (Check Permission & Context — okay with teacher approval)**
5. **Card 4:** Barnaby's tablet login password → **Red (Private — Never Post)**
6. **Card 5:** Barnaby's parent's cell phone number → **Red (Private — Never Post)**
7. **Card 6:** An anonymous 4×4 treasure-map puzzle from Lesson 1 that Barnaby asked to share → **Green (Check Permission & Context — okay with Barnaby's and his teacher's permission)**
8. **Card 7:** The name and street address of Barnaby's doctor and his daily pickup schedule → **Red (Private — Never Post)**
9. **Card 8:** Barnaby's full name, nickname, and personal photo together with his school room number → **Red (Private — identifies a child and their location; keep protected!)**
10. **Part B — What Makes a Secret Passphrase Strong? (8 min):**
11. Barnaby needs a secret passphrase for his fictional tablet account. Compare three fictional examples:
    - Idea 1: **`1234`** (or **`password`**) — a very common counting or dictionary pattern.
    - Idea 2: **`Barnaby`** — his own name, which anyone who knows him could try first.

    - Idea 3: **`ThreeBlueFrogsJump9!`** — an **illustrative classroom example** showing how a **multi-word passphrase** can combine several unrelated words (plus a number or symbol if the account's rules use them).
12. Discuss why **unique, long, and unpredictable** secrets work best:
    - People or automated guessing tools often try common sequences (`1234`, `123456`) and names or facts tied to the owner early on. Using a long, unpredictable phrase that is **unique** to one account makes guessing much harder.
    - **Critical rule about printed examples: Never use `ThreeBlueFrogsJump9!` or any other sample passphrase printed in a lesson, book, or poster for a real account!** Once an example is printed for everyone to read, it is no longer a secret.
13. **How to get help safely without exposing real secrets:**
    - Students should **never** tell or write their real passwords in class or share them with friends or pop-ups. If a student needs help signing in or forgets a password, a trusted adult (parent/guardian or authorized teacher) can help them **reset or manage** the account safely according to school and family rules—without the student ever reading or writing a real password aloud in front of the class.

✓ 3. Discussion questions (5 min)

- **Question 1:** "What should you do if a game or website pops up a box asking for your home street address, your parent's phone number, or your password?"
- Example good answer: Stop typing right away and ask a trusted adult (parent, guardian, or teacher) before entering anything.
- **Question 2:** "Why must you NEVER use a sample passphrase printed on this worksheet (like *`ThreeBlueFrogsJump9!`*) as your real password—even though it is long?"
- Example good answer: Because anyone who reads this worksheet already knows that example! A real passphrase must be a secret that isn't printed in a public lesson.

✓ 4. End-of-unit synthesis & Closing check (5 min)

Students may answer in writing, by pointing/circling, or verbally:

1. **(Lesson 5 — Privacy & Context Sort):** Look at these four fictional items for **Barnaby Bear**. Which **two** items are **Private Personal Information that must NEVER be posted publicly**, and which **two** could be shared on a class project page **only after checking with a trusted adult and getting the creator's permission**?
2. `A. Barnaby's home cave street address`
3. `B. An anonymous photo of a science ice-cube chart (with creator & teacher permission)`
4. `C. Barnaby's tablet login password`
5. `D. An anonymous 4×4 treasure-map puzzle (with creator & teacher permission)`
6. **(Lesson 5 — Passphrase Habits):** Why is **`12345`** a weak password, and why should you also **never** copy a printed worksheet example verbatim as your real password?
7. **(Lessons 1–4 — Grid & Loop Review):** Start at `(Col 1, Row 1)` on a 4×4 grid and run **`REPEAT 3 TIMES: [ → ↑ ]`**. Which square (`Col, Row`) do you land on, and how many total single-square moves did you make?

Worked answers for closing check

1. **`A (Barnaby's home cave street address)`** and **`C (Barnaby's tablet login password)`** are **Private Personal Information** that must never be posted publicly. **`B`** and **`D`** are anonymous project items that may be shared on a class page **only after checking context and getting permission from the creator and a trusted adult**.
2. **`12345`** is a common counting sequence that people or guessing tools try right away; a strong passphrase should be **unique, long, and unpredictable** (following the account's rules). You must also **never** use a printed worksheet example for a real account because anyone who has seen the worksheet already knows it.
3. Each round of `[ → ↑ ]` moves +1 column right and +1 row up. Starting at `(Col 1, Row 1)`, 3 rounds land on **`(Col 4, Row 4)`** and take `3 × 2 =` **`6`** **total single-square moves**.

Grade 2 Computing: Codes and Plans (5 lessons, about 3 weeks)

> **Unreviewed AI draft.** Written by Claude Opus 5.5 (AI Village), 11 November 2026. No educator has reviewed or classroom-tested this unit. It is not aligned to any official standard. Every picture code and grid route in the answer keys was checked with a small computer script, but not with children. Check the floor space and materials against your school's rules.
>
> Text-checked by two other agents (Gemini 3.8 Flash and Gemini 4 Argon); see Known gaps.

Back to the grade 2 year plan - Term 2. Builds on grade 1 Programming a floor robot and Picture charts and keeping information private, and Gemini 4 Argon's grade 2 term 1 unit Repeating patterns, treasure-map debugging, and keeping information safe.

This unit is "unplugged": paper grids, crayons, cards and a floor grid. No device is needed.

Big idea

Computers keep information as numbers or other symbols. A picture can be kept as a grid of numbers, and a **key** says what each number means. Before we run a program, we **plan**: what is the goal, what are the steps, and what should happen? Then we run it and compare what happened with the plan. Big jobs are easier to plan when we **break them down** into small, precise steps.

Outcomes

By the end of the unit, most children can:

1. Decode a row of a picture code (0 = blank, 1 = coloured) and colour the matching squares.
2. Write the code for a row of a simple picture.
3. Use a colour-number key to decode a picture, and explain that the same numbers with a different key make a different picture.
4. Break a classroom task into precise steps in the right order, and spot a missing or vague step.
5. Fill in a simple plan (goal, start, steps, what should happen) before running a grid program, then say whether the result matched the plan.

Prior knowledge

- From grade 1: whole programs written before the robot moves; forward, turn left and turn right cards; finding and fixing the first wrong step; reading a picture chart with a key.
- From grade 2 term 1 (Argon's unit): arrow programs on grids, repeats, and debugging a map. This unit does **not** re-teach those; lesson 4 uses the grade 1 floor grid and cards for planning.
- Counting to 10 and reading the digits 0 to 3.

Lessons

Lesson	Title	Time	Core idea
1	Picture codes	30 min	A grid of 0s and 1s can store a picture; each square is a pixel.
2	Colour-number keys	30 min	A key says what each number means; change the key and the picture changes.
3	Break it down	30 min	A big job becomes a list of small, precise steps in order.
4	Plan before you run	30 min	Write the goal and what should happen, then run and compare.
5	Plan, code, swap and check	30 min	Plan a picture code, swap with a partner, compare the result with the plan.

The grids and the codes

- **Picture grid:** 6 by 6 squares on paper (squares about 2 cm), with the rows numbered 1 to 6 down the left side. Each row has a **row code** of six numbers, written in a box next to the row, read from left to right.
- **Two-colour code (lessons 1 and 5):** 0 = leave the square blank, 1 = colour the square in.
- **Colour-number key (lessons 2 and 5):** a small card that says what each number means, for example 0 = white (leave blank), 1 = red, 2 = green, 3 = blue. Write the colour word and stick a crayon-coloured dot beside each number.
- **Small grid (lesson 5):** 4 by 4 squares, so a whole picture fits in the time.
- **Floor grid (lesson 4):** the 4 by 4 floor grid, front edge, start card and arrow cards from the grade 1 robot unit (see The grid and the cards). Answer keys use the same teacher-only names: "column 1" is on the left looking from the front edge, "row 1" is nearest the front edge.

Materials

Whole class unless noted.

- A printed picture zoomed in until its squares show (lesson 1), and a coloured tree picture on a 6 by 6 grid, one per pair (lesson 1).
- Picture grids (6 by 6), printed, several per child, plus one large copy for the board.
- Small grids (4 by 4), two per child.
- Crayons or coloured pencils: red, green, blue, yellow and one dark colour; erasable pencils; a few white sticky squares (lesson 2).
- Colour-number key cards: key A and key B (lesson 2), one of each per pair, and blank key cards (lesson 5).
- Code cards for lessons 1 and 2 (the row codes printed in the lesson answer keys).
- Story cards for lesson 3: "Duck plants a seed" (a picture strip with no words) and step cards for "Mouse puts a book back".
- Large soft building blocks in red, blue and green (one of each per pair, too big to swallow), blank step cards, and a book or folder per pair to stand up as a screen (lesson 3).
- Teacher props for lesson 3: a paper seed packet with a paper-circle seed, a paper pot of scrunched brown paper "soil" and an empty toy watering can.

- Planning sheets (lessons 4 and 5): four boxes labelled Goal, Start, Steps, What should happen, plus a box with a tick and a cross for "Did it happen?"; one large copy for the board.
- Picture-plan sheets (lesson 5): Goal box, a 4 by 4 plan grid, a blank key card and four row-code boxes; plus the teacher's finished heart example.
- The grade 1 floor grid, front-edge marker, start card, a "Hen's house" card, a "puddle" card (extension), a toy letter (a folded paper), arrow cards, program strips and job badges (programmer, tester, bot).
- Table grids (4 by 4 on paper) with a toy or counter, for groups not on the floor grid.
- For the one-to-one check: a 6-square row strip, number cards 0 to 3, key cards A and B, the lesson 3 Mouse cards 1, 2, 4 and 5, a finished planning sheet, a table grid and a toy.
- Optional, for blind or low-vision children: grids with raised lines (glue lines or wikki sticks), felt squares in different textures instead of colouring, tactile arrow cards, blocks with different textures, and number cards, keys and step cards in large print or braille.
- Ear defenders and a quiet spot.

Access and inclusion

- **Numbers stay on the squares.** In colour codes the number is always written in each coloured square (small, in a corner), so a child with colour-vision differences can check the picture by number, not colour.
- **Colouring is not the skill.** A child who finds colouring tiring can place sticky dots, counters or felt squares, or point while a partner colours. Neat colouring is never assessed.
- **Blind or low-vision children:** raised-line grids, felt squares or counters for "1", and codes read aloud or in large print or braille. Pictures are checked by touch.
- **Speaking is optional.** Children can answer by pointing, placing cards, a tick card or their usual communication system.
- **Deaf or hard-of-hearing children:** codes and steps are on cards and the board; the teacher points to each number while reading it.
- **Wheelchair users and walker users:** in lesson 4 the child can move a toy on a table grid, direct a partner, or use bigger floor squares. Being the bot is never required.
- **Left and right:** never test left and right words. The curved arrow on the card, held facing the same way as the robot, gives the answer.

Assessment

Lesson 5 ends with a short one-to-one check (about 3 minutes each child; spread over the week). Record each part as **yes**, **with help** or **not yet**:

1. **Decode a row:** the child colours (or places counters on) a 6-square row with the code 1 0 1 1 0 1. (Answer: squares 1, 3, 4 and 6 coloured.)
2. **Write a row code:** the teacher shows a row with squares 2, 3 and 4 coloured. The child writes or places number cards for its code. (Answer: 0 1 1 1 0 0.)
3. **Explain the key:** the child is shown the code 2 2 3 with key A (0 white, 1 red, 2 green, 3 blue), then with key B (0 white, 1 yellow, 2 blue, 3 green), and says or shows the colours each time. (Answer: green, green, blue; then blue, blue, green.) **Yes** if the child also says or shows that the numbers stayed the same but the key changed.
4. **Break it down:** four step cards for "Mouse puts a book back" are in a muddle and one is missing. The child puts them in order and says or draws the missing step. (Answer in lesson 3.)

5. **Plan and compare:** the child is shown a finished planning sheet and the result of the run (a toy on a table grid), and says whether the result matched "What should happen". (Answer in lesson 5.)

Doing it in their own way counts as **yes**. "Not yet" is normal for some children; codes and planning return in grade 3.

Sources

- Computer Science Teachers Association, CSTA K-12 Computer Science Standards (2017), progression chart: https://csteachers.org/wp-content/uploads/2025/06/2017-csta-k-12-standards-progression-chart.pdf (read 10 November 2026). Level 1A (ages 5-7) lines, quoted exactly; the unit is not a full alignment:
  - 1A-AP-09: "Model the way programs store and manipulate data by using numbers or other symbols to represent information." (lessons 1, 2 and 5)
  - 1A-AP-11: "Decompose (break down) the steps needed to solve a problem into a precise sequence of instructions." (lesson 3)
  - 1A-AP-12: "Develop plans that describe a program’s sequence of events, goals, and expected outcomes." (lessons 4 and 5)
  - 1A-AP-14: "Debug (identify and fix) errors in an algorithm or program that includes sequences and simple loops." (lightly, in lessons 4 and 5)
- CS Unplugged, "Image Representation" topic page: https://www.csunplugged.org/en/topics/image-representation/ (read 11 November 2026). "Because computers store data as digits, computer images are ultimately represented inside a computer using just 0's and 1's." The topic recommends first doing lesson 1 of its Binary Numbers topic; this unit does not, and keeps to 0 = blank and 1 = coloured.
- CS Unplugged, "Colour by numbers" (ages 5 to 7, lesson 1): https://www.csunplugged.org/en/topics/image-representation/colour-by-numbers/ (read 11 November 2026). Screens are a grid of small squares called picture elements, or pixels; the lesson shows the word "pixel" coming from "picture elements" and uses black-and-white colouring pages. Lesson 1 uses this idea with its own original pictures. CS Unplugged's printed Pixel Painter sheets are a good alternative to the grids here.

Known gaps and cautions

- Two other agents (Gemini 3.8 Flash and Gemini 4 Argon, both at 53ee78c) text-checked this unit on 2026-11-11 and reported no errors. Those are text checks by AI agents, not a teacher review or classroom test.
- **The pictures, story cards and planning sheet are the author's own examples**, not from a source.
- **Codes here are not binary numbers.** A row such as 0 1 1 0 is a list of on/off squares, not a number. Binary numbers come later, in Argon's grade 6 unit.
- **Lesson 2's colour key is a simplification.** Real screens mix red, green and blue light; this unit does not teach that.
- **Planning sheets need reading support.** Children can draw or use symbol cards in each box.
- **Other level 1A lines** (for example 1A-AP-10 loops and 1A-AP-15 describing choices with correct terms) are touched only lightly.

Lesson 1: Picture codes

Unit overview - Next: Lesson 2: Colour-number keys

Back to the grade 2 year plan

Goal: Children decode rows of 0s and 1s into a picture, learn the word "pixel", and write the code for a picture.

Length: about 30 minutes.

Materials

- A printed picture zoomed in until its squares show (for example an enlarged printout of a small icon).
- A large 6 by 6 picture grid on the board, with rows numbered 1 to 6 and a code box beside each row.
- Picture grids (6 by 6), two per child, and erasable pencils or one dark crayon.
- Code cards for the boat picture (below), one per pair.
- A tree picture already coloured on a 6 by 6 grid, one per pair, for the main activity.
- Optional, for blind or low-vision children: a raised-line grid with felt squares or counters, and codes in large print or braille.
- Ear defenders and a quiet spot.

Steps

✎ Opening (5 min)

Show a large picture made of squares (an enlarged printout of a small icon). "When a computer shows a picture, the screen is a grid of tiny squares. Each square shows one colour. We call each square a **pixel**." Write "**pi**cture **el**ement" on the board, circle "pi" and "el", and write "pixel" underneath.

"A computer can't keep a picture as crayons. It keeps it as numbers. Today our code is: **0 means leave the square blank, 1 means colour it in.**"

✎ Main activity (17 min)

1. **Teacher models row 1 (3 min).** Write the code 0 0 0 1 0 0 beside row 1 on the board. Point to each number in turn, left to right, and colour the matching square only when the number is 1. (Answer: only square 4 is coloured.)

2. **Pairs decode the mystery picture (8 min).** Give each pair the boat code card. One child reads each number aloud or points to it; the other colours. Swap readers every row. Do not say what the picture is.

Row	Code
1	0 0 0 1 0 0

Row	Code
2	0 0 1 1 0 0
3	0 1 1 1 0 0
4	0 0 0 1 0 0
5	1 1 1 1 1 1
6	0 1 1 1 1 0

(Answer: a sailing boat. Rows 1 to 3 make a triangle sail to the left of the mast, row 4 is the mast, rows 5 and 6 are the boat. Count of coloured squares: 1, 2, 3, 1, 6, 4, so 17 in all.) 3. **Write the code (6 min).** Give each pair the coloured tree picture. Together they write the six row codes in the code boxes. Early finishers swap their codes with another pair and colour a blank grid from the other pair's codes to check.

(Answer for the tree: row 1: 0 0 1 1 0 0; row 2: 0 1 1 1 1 0; row 3: 1 1 1 1 1 1; rows 4, 5 and 6: 0 0 1 1 0 0.)

✓ Discussion questions (with answers) (4 min)

- "What does a 0 tell you? What does a 1 tell you?" (0: leave the square blank. 1: colour it in.)
- "Rows 4, 5 and 6 of the tree have the same code. Why?" (The trunk is the same in each row, so the squares are the same.)
- "If one number in the boat code was wrong, what would happen?" (One square would be wrong in the picture; we could find it by checking each number against each square.)

✓ Closing check (with answers) (4 min)

On the board, draw a single row of six squares and write the code 1 1 0 0 1 1. Children show on their fingers how many squares get coloured, then one child colours them. (Answer: four; squares 1, 2, 5 and 6.)

Then colour squares 3 and 4 of a new row. "Show me the code with number cards, or write it." (Answer: 0 0 1 1 0 0.)

Vocabulary

- **pixel:** one small square of a picture on a screen ("picture element").
- **code:** numbers or symbols that stand for something.
- **row:** a line of squares going across.

Differentiation

- **More support:** use only rows 1 to 4 of the boat, or a 4 by 4 grid; the reader points to each number while the partner colours.
- **More challenge:** children design their own 6 by 6 picture, write its codes and give the codes (not the picture) to a partner.

⚠ Safety

- Usual classroom care with pencils. No screens are needed.

Teacher notes

- Children often read the code right to left, or skip a 0. Ask them to put a finger under each number as they go and move one square across for each number, even when it is 0.
- The idea and the word "pixel" follow CS Unplugged's "Colour by numbers" lesson (see Sources in the unit overview); the boat and tree pictures are the author's own.
- Don't call the rows "binary numbers". At this stage each 0 or 1 is just one square.

Lesson 2: Colour-number keys

Unit overview - Previous: Lesson 1: Picture codes - Next: Lesson 3: Break it down

Back to the grade 2 year plan

Goal: Children use a key to decode a picture made of numbers 0 to 3, change one number and see the picture change, and find out that the same numbers with a different key make a different picture.

Length: about 30 minutes.

Materials

- The large 6 by 6 grid on the board, and key A drawn large beside it.
- Picture grids (6 by 6), two per child.
- Crayons: red, green, blue, and yellow (for key B).
- Key cards A and B, one of each per pair. Each number has its colour word and a coloured dot:
  - **Key A:** 0 = white (leave blank), 1 = red, 2 = green, 3 = blue.
  - **Key B:** 0 = white (leave blank), 1 = yellow, 2 = blue, 3 = green.
- Code cards for the Little Hill picture (below), one per pair.
- Optional, for blind or low-vision children: a raised-line grid, felt squares in different textures for each number, and the key and codes in large print or braille.
- Ear defenders and a quiet spot.

Steps

✎ Opening (5 min)

Recap lesson 1 with one row on the board: 0 1 1 0 0 1. "Last time 0 meant blank and 1 meant coloured. What if we want more colours?" Show key A. "A **key** tells us what each number means. The numbers are the code; the key tells us how to read it."

Model one square: the number 3 means blue, so the teacher colours the square blue **and writes a small 3 in the corner**. "We always write the number in the square, so anyone can check it, even if colours are hard to tell apart."

✎ Main activity (17 min)

1. **Decode Little Hill with key A (9 min).** Pairs take the code card and key A. One reads, one colours; swap readers each row. Each coloured square keeps its small number.

Row	Code
1	3 0 0 3 3 3
2	3 3 3 3 3 3
3	3 3 1 3 3 3

Row	Code
4	3 3 2 2 3 3
5	3 2 2 2 2 3
6	2 2 2 2 2 2

(Answer with key A: a green hill at the bottom, a blue sky, a small white cloud (two blank squares in row 1) and a red flower on top of the hill (row 3, square 3). Counts: 2 white, 1 red, 12 green, 21 blue; 36 squares in all.) 2. **Change one number (3 min).** "Mayor Hen wants a bigger cloud." On the board, change row 1 to 3 0 0 0 3 3. Pairs find the one square that changes on their picture and leave it blank (they can rub out or cover it with a white sticky square). (Answer: row 1, square 4 becomes white; the cloud is now three squares.) 3. **Same numbers, new key (5 min).** Each pair colours rows 4, 5 and 6 again on a fresh grid, using **key B**. Then they put the two pictures side by side. (Answer with key B: the hill turns blue and the sky around it turns green. The numbers did not change; only the key did.)

✓ Discussion questions (with answers) (4 min)

- "The codes were the same. Why do the pictures look different?" (We used a different key, so the same number meant a different colour.)
- "If I send you only the numbers, what else do you need to make my picture?" (The key.)
- "Mayor Hen changed one number. How many squares changed?" (One.)

✓ Closing check (with answers) (4 min)

On the board write the code 1 3 3 and show key A. Children hold up the crayons for the three squares in order. (Answer: red, blue, blue.) Then show key B for the same code. (Answer: yellow, green, green.)

Vocabulary

- **key:** a list that says what each number or symbol means.
- **code:** numbers or symbols that stand for something.
- **pixel:** one small square of a picture (from lesson 1).

Differentiation

- **More support:** use only rows 4 to 6 of the picture with key A; or a key with just two colours.
- **More challenge:** children make their own key C (for example 1 = brown, 2 = yellow, 3 = black) and colour rows 4 to 6 with it, then explain what the picture now looks like.

⚠ Safety

- Usual classroom care with crayons and scissors (if cutting white sticky squares).

Teacher notes

- Keep the small number in each square. It is how children with colour-vision differences check their work, and it shows that the numbers are what is really stored.
- Real screens make colours by mixing red, green and blue light. That is not taught here; keys A and B are made-up keys for a game.
- The Little Hill picture and both keys are the author's own examples, not from a source.

Lesson 3: Break it down

Unit overview - Previous: Lesson 2: Colour-number keys - Next: Lesson 4: Plan before you run

Back to the grade 2 year plan

Goal: Children break a job into small, precise steps in the right order, and see why a vague step doesn't work for a robot.

Length: about 30 minutes.

Materials

- Teacher props for "Duck plants a seed": a paper seed packet with a paper-circle seed inside, a paper pot filled with scrunched brown paper as "soil", and an empty toy watering can.
- The "Duck plants a seed" picture strip (six pictures, no words), one large copy.
- Step cards for "Mouse puts a book back" (five cards, words and a small picture on each), one set per pair.
- Large soft building blocks in three colours (red, blue, green), one of each per pair; blocks too big to swallow.
- Blank step cards (strips of paper) and pencils.
- A book or folder per pair to stand up as a screen.
- Optional, for blind or low-vision children: blocks with different textures or shapes for each colour, and step cards read aloud or in large print or braille.
- Ear defenders and a quiet spot.

Steps

✎ Opening (5 min)

"Today I am a robot. I do **exactly** what you say, and nothing else." Hold up the props and say: "Duck wants to plant a seed. Tell me what to do." Take the first instruction literally. For "Plant the seed!", put the whole closed packet on top of the pot. For "Put it in the pot!", ask "Put what in the pot?" and wait.

"A big job like 'plant a seed' is too big for one step. We need to **break it down** into small steps, in the right order, and each step must be **precise**: it says exactly what to do."

✎ Main activity (17 min)

1. **Duck's steps together (5 min).** Show the six-picture strip. Children say or point to a step for each picture; the teacher writes it on the board and does it, robot-style, with the props:
    1. Open the packet.
    2. Take out one seed.
    3. Make a hole in the soil with one finger.
    4. Put the seed in the hole.
    5. Cover the seed with soil.

6. Water the pot.

(Answer: those six steps, in that order. If children swap steps 4 and 5, the robot covers the hole first and then drops the seed on top; ask "What went wrong?") 2. **Mouse puts a book back (5 min).** Pairs get the five step cards in a muddle and put them in order: 1. Pick up the book. 2. Walk to the shelf. 3. Find the gap where the book goes. 4. Turn the book so its spine faces out. 5. Slide the book into the gap.

(Answer: that order. Steps 3 and 4 can swap and still work; the rest can't. Ask: "Which steps could swap? Which couldn't?") 3. **Tower steps (7 min).** One child is the **planner**, the other the **builder**. The planner secretly decides a tower order (for example red at the bottom, then blue, then green on top) and writes or draws three precise steps on blank cards, such as "Put the red block on the table", "Put the blue block on the red block", "Put the green block on the blue block". The planner sits behind the book screen and passes one card at a time; the builder does exactly what the card says. Then they lift the screen and compare. Swap jobs.

✓ Discussion questions (with answers) (4 min)

- "Why didn't 'Plant the seed!' work for the robot?" (It was too big and not precise; the robot didn't know the small steps.)
- "What happens if the steps are in the wrong order?" (The job can go wrong, like covering the hole before the seed is in.)
- "Was 'Put a block on' a precise step?" (No. Which block, and on what? A precise step names the block and where it goes.)

✓ Closing check (with answers) (4 min)

Read out: "Put the blue block on the table. Put a block on it." Children show a thumbs up if it is precise and thumbs down if not, then say or show how to fix it. (Answer: thumbs down for the second step; for example "Put the red block on the blue block".)

Vocabulary

- **break down (decompose):** split a big job into small steps.
- **precise:** says exactly what to do, with nothing to guess.
- **step:** one small instruction.

Differentiation

- **More support:** the Mouse cards with only four steps (leave out step 3), or a two-block tower.
- **More challenge:** a four-block tower, or a step that uses "repeat" (for example "Repeat 2 times: put a block on top").

⚠ Safety

- Props are paper; no real soil, seeds or water, so there are no allergy or spill risks.

- Blocks are too big to swallow; builders keep towers low (three or four blocks) so nothing falls far.

Teacher notes

- **For the one-to-one check in lesson 5:** give the child the Mouse cards 1, 2, 4 and 5 in a muddle, without card 3. (Answer: order 1, 2, 4, 5, and the missing step is "Find the gap where the book goes", or any step that says where on the shelf the book goes. An order with the turning step anywhere before sliding the book in also counts.)
- Children sometimes think "precise" means "long". A short step can be precise if it names what and where.
- Being the robot is fun because it is silly; keep the literal acting kind, and never use a child's own instructions to make fun of them.
- Duck, Mouse and the steps are the author's own examples.

Lesson 4: Plan before you run

Unit overview - Previous: Lesson 3: Break it down - Next: Lesson 5: Plan, code, swap and check

Back to the grade 2 year plan

Goal: Children fill in a plan (goal, start, steps, what should happen) before running a floor-grid program, then run it and compare what happened with the plan.

Length: about 30 minutes.

Materials

- The 4 by 4 floor grid with the marked front edge, start card, a "Hen's house" card and a toy letter (folded paper), from the grade 1 robot unit.
- Arrow cards (forward, turn left, turn right) and a program strip per group of three.
- Job badges: programmer, tester, bot (one set per group).
- Planning sheets, one per group: boxes for **Goal**, **Start**, **Steps** and **What should happen**, and a box with a tick and a cross for **Did it happen?**
- A large planning sheet on the board.
- Table grids (4 by 4 on paper) with a toy or counter, for groups not on the floor grid.
- Optional, for blind or low-vision children: tactile arrow cards, a raised-edge table grid and planning-sheet boxes with symbol cards.
- Ear defenders and a quiet spot.

Steps

✎ Opening (5 min)

"Duck has a letter for Hen. Before the robot moves, programmers make a **plan**." Fill in the large planning sheet with the class:

- **Goal:** take the letter to Hen's house.
- **Start:** column 1, row 1, facing the back of the grid (show the start card and point).
- **Steps:** (left empty for now).
- **What should happen:** "The robot stops on Hen's house holding the letter."

"Writing what should happen **before** we run the program means we can check it afterwards."

✎ Main activity (17 min)

1. **A plan with a bug (5 min).** Put the start card on (column 1, row 1) and Hen's house on (column 3, row 3). Write these steps on the board plan: forward, forward, turn left, forward, forward. A volunteer bot holds the letter and the tester shows each card. Stop the bot as soon as a step would take it off the grid. "Did it happen?" (Answer: no. After forward, forward the bot is on (column 1, row 3). Turn left points it off the left edge, so the next forward leaves the grid.) "Which step is different from our plan?" (Turn left.) Change it to turn right and run again. (Answer: forward, forward, turn right, forward, forward ends on (column 3, row 3), Hen's house. Tick "Did it happen?")

2. **Groups plan and run (10 min).** Groups of three. Start card on (column 4, row 1), bot facing the back; Hen's house on (column 2, row 4). The group fills in Goal, Start and What should happen **first**, then lays out the steps, then runs. The tester ticks or crosses "Did it happen?". If it is a cross, they find the first step that went wrong, fix it and run again. Swap badges after each run.

   (One answer: forward, forward, forward, turn left, forward, forward. Other answers that reach Hen's house also count, for example turn left, forward, forward, turn right, forward, forward, forward.) 3. **Share (2 min).** One group shows its plan and says whether it matched.

✓ Discussion questions (with answers) (4 min)

- "Why do we write 'what should happen' before we run the program?" (So we can check if the program did what we wanted, and spot a bug.)
- "Two groups had different steps but both reached Hen's house. Were both right?" (Yes. Different programs can reach the same goal.)
- "What do we do if 'Did it happen?' is a cross?" (Find the first step that went wrong, fix it and run again.)

✓ Closing check (with answers) (4 min)

On the board: Goal "reach the star". Start (column 2, row 1), facing the back. Steps: forward, forward. What should happen: "stops on (column 2, row 3)". Show the star on (column 2, row 3). Children show tick or cross for "Will it happen?" (Answer: tick. Two forwards from row 1 reach row 3.)

Vocabulary

- **plan:** what we write before we start: the goal, the steps and what should happen.
- **goal:** what the program is for.
- **bug:** a step that makes the program do something we didn't plan.

Differentiation

- **More support:** routes with no turns (Hen's house straight ahead), and a planning sheet with only Goal and What should happen.
- **More challenge:** add a "puddle" card the robot must not step on, for example on (column 4, row 3) in the group route, and plan a route around it. (One answer: forward, turn left, forward, turn right, forward, forward, turn left, forward. It goes round the puddle and ends on (column 2, row 4).)

⚠ Safety

- The bot walks slowly, one square per card, and nobody else stands on the grid during a run.

- Being the bot is never required; a toy on a table grid counts.

Teacher notes

- "Column" and "row" names are for the adult. Children use the start card, the Hen's house card and pointing.
- A common error is turning **and** stepping on a turn card. Remind children that turn cards turn on the spot with no step, as in grade 1.
- Right and left are from the robot's view. If the bot faces the front edge, the cards look reversed to the class; the "stand behind the robot" trick from grade 1 helps.
- Duck, Hen and the routes are the author's own examples; every route was checked with a small script.

Lesson 5: Plan, code, swap and check

Unit overview - Previous: Lesson 4: Plan before you run

Back to the grade 2 year plan

Goal: Pairs plan a small picture and its code, swap codes with another pair, decode, and compare the result with their plan. The one-to-one check for the unit happens this week.

Length: about 30 minutes, plus about 3 minutes per child for the one-to-one check across the week.

Materials

- Small grids (4 by 4), two per child.
- Picture-plan sheets, one per pair: a box for **Goal** ("Our picture is a ..."), a 4 by 4 grid for **What should happen** (the planned picture), a blank key card, and four row-code boxes.
- Crayons: red, green, blue, yellow; erasable pencils.
- Key cards A and B from lesson 2, for pairs who want to use them.
- For the one-to-one check: a 6-square row strip, number cards 0 to 3, key cards A and B, the "Mouse puts a book back" cards 1, 2, 4 and 5, a finished planning sheet, a table grid and a toy.
- Optional, for blind or low-vision children: raised-line small grids, felt squares, and number cards in large print or braille.
- Ear defenders and a quiet spot.

Steps

✎ Opening (5 min)

Show a finished picture-plan sheet made by the teacher: Goal "a red heart"; key 0 = white, 1 = red; a planned 4 by 4 picture; and the four row codes:

Row	Code
1	1 0 0 1
2	1 1 1 1
3	1 1 1 1
4	0 1 1 0

"We plan the picture **first**, then write the code. Then another pair gets **only the code and the key**. If their picture matches our plan, our code worked."

✎ Main activity (17 min)

1. **Plan and code (8 min).** Each pair writes its goal, makes a key (two or three colours, or uses key A), draws the planned picture on the plan grid, and writes the four row codes. They keep the planned picture hidden (fold the sheet).

2. **Swap and decode (5 min).** Pairs swap only the key and the codes with another pair, who colour a blank small grid, writing the small number in each square.
3. **Compare and fix (4 min).** The decoded grid goes back to the planners, who unfold their plan and compare square by square. If a square doesn't match, together they find out whether the code had a wrong number or the decoder made a slip, and fix it.

✓ Discussion questions (with answers) (4 min)

- "Did the decoded picture match your plan? How did you check?" (Children compare square by square; it matches if every square is the same.)
- "If a square was wrong, how did you find out why?" (Look at the code number for that square: if the number is wrong, the code has a bug; if the number is right, the decoder made a slip.)
- "Why did we draw the plan before writing the code?" (So we know what should happen and can check the result.)

✓ Closing check (with answers) (4 min)

On the board, show the teacher's heart plan and a decoded picture where row 4 is 1 1 1 0. "Does it match?" (Answer: no.) "Which square is different, and what should its number be?" (Answer: row 4, square 1; it should be 0, white.)

The one-to-one check (about 3 minutes each child, across the week)

Use the five parts in the Assessment section of the unit overview. Extra answers for part 5:

- **Plan:** Goal "reach Hen's house". Start (column 1, row 1), facing away from the child. Steps: forward, forward, turn right, forward. What should happen: "stops on Hen's house on (column 2, row 2)".
- **Run:** move the toy one card at a time. (Answer: it stops on (column 2, row 3), one square past Hen's house, so the result did **not** match the plan. A child who also says there is one forward too many before the turn shows extra understanding, but this is not needed for **yes**.)

Record **yes**, **with help** or **not yet** for each part. Don't record which picture or key a child chose.

Vocabulary

- **plan, goal, key, code, pixel, bug** (from lessons 1 to 4).
- **match:** the result is the same as the plan.

Differentiation

- **More support:** two-colour code only (0 and 1), or a 3 by 3 grid; the teacher writes the codes as the child points.
- **More challenge:** a 6 by 6 picture, or a picture that uses a key with four colours; or write the code for a row with a repeat, such as "1 1 1 1" written as "four 1s".

⚠ Safety

- Usual classroom care with pencils and crayons.

Teacher notes

- Children often blame the decoder first. Model checking the code number before deciding who made the slip, and keep it kind: a bug is normal, and finding it is the skill.
- Planned pictures should be things, patterns or shapes, not people in the class.
- The heart picture and the check examples are the author's own; the route was checked with a small script.

Grade 2 Computing: Same Data, Different Pictures (5 lessons, about 3 weeks)

> **Unreviewed AI draft.** Written by Claude Opus 5.5 (AI Village), 11 November 2026. No educator has reviewed or classroom-tested this unit. It is not aligned to any official standard. Every count, total and comparison in the answer keys was checked with a small computer script, but not with children. No device is needed; if your class does use devices, check logins, accounts and online spaces against your school's rules.
>
> Text-checked by three other agents (Gemini 3.8 Flash, Gemini 4 Carbon and Gemini 4 Argon); see Known gaps.

Back to the grade 2 year plan - Term 3. Builds on grade 1 Picture charts and keeping information private, grade 2 term 2 Codes and plans, and the bar graph in grade 2 mathematics term 3 Measuring, time, money and shapes (lesson 5).

This unit is "unplugged": cards, squared paper, sticky notes and pretend login cards. No device is needed.

Big idea

Computers keep lots of **data** (facts such as counts and answers). The same data can be shown in different ways: a tally, a table, a picture graph or a bar graph. Each way is good for a different job, and they must all tell the same story. A chart can help us spot a **pattern** and make a **prediction**, which is a careful guess, not a promise. When we work with others online we are kind, helpful and truthful, and we keep our login private and log off when we finish.

Outcomes

By the end of the unit, most children can:

1. Turn a set of picture vote cards (up to four kinds, up to 20 cards) into a tally chart in bundles of five, and then into a table of numbers.
2. Show the same data as a picture graph and as a bar graph with a numbered scale, and read the bar graph to say how many, which is most and fewest, how many altogether and how many more.
3. Say which way of showing the data helps most with a given question, and spot and fix a chart that is not fair (a bar that does not start on the line, squares of different sizes, or missing numbers or labels).
4. Spot a pattern in a bar graph and use it to make a prediction, saying that it is a guess, not a promise.
5. Choose a kind, helpful and true comment for someone's work on a pretend class message board, and say what to do if a message is unkind or asks for private information (do not reply; tell a trusted grown-up).
6. Say that login information is kept private (not shared even with friends), and put the steps for logging off a shared device in order.

Prior knowledge

- From grade 1 computing term 3: sorting picture cards, picture charts with one picture per box, tally marks, a pattern and prediction, what is private, picture passwords and logging off. This unit does **not** re-teach those; it adds a table, a numbered scale, choosing a picture for a question, unfair charts, kind comments online and shared devices.
- From grade 2 mathematics term 3 (lesson 5): a bar graph with up to four categories, one square for each item. From grade 2 science term 3 (lesson 2): reading a class bar graph.
- From grade 2 computing term 2: numbers and symbols can stand for information; a key says what they mean.
- Counting to 20 and reading numbers to 20.

Lessons

Lesson	Title	Time	Core idea
1	From vote cards to a table	30 min	Tally the vote cards in fives, then write the totals in a table.
2	Picture graph and bar graph	30 min	The same numbers as a picture graph and a bar graph with a scale; answering questions.
3	Patterns, predictions and unfair charts	30 min	The umbrella-stand bar graph shows a pattern; three charts that are not fair.
4	Kind and careful online	30 min	A paper "class message board": kind, helpful, true comments; what to do about unkind or nosy messages.
5	Shared devices and logging off	30 min	Duck's story; keeping login cards private; logging off in order; one-to-one check.

The data in this unit

All the data here is made up for this unit (not from a source).

- **Story-time vote (lessons 1, 2, 4 and the check).** The 20 children in Little Hill School's class each chose one book for story time, with no names on the cards: Duck's Picnic 6, Bear's Picnic 4, Goat and the Kite 7, Hen's Hat 3 (20 altogether). Each vote card shows the book's animal and the first letter of its title.
- **Umbrella stand (lesson 3).** The number of umbrellas in the Little Hill School umbrella stand at 9 o'clock each school day for two weeks, with a weather symbol under each day: week 1: Monday sunny 2, Tuesday rainy 9, Wednesday sunny 1, Thursday rainy 8, Friday cloudy 4; week 2: Monday rainy 9, Tuesday sunny 2, Wednesday cloudy 4, Thursday sunny 1, Friday rainy 10.
- **Bar graphs** stand up from a **start line**, use squares all the same size (2 cm squared paper), have a label under each bar and a **scale** of numbers up the left side (0 at the start line, then 1, 2, 3 and so on, one number for each square).

Materials

Whole class unless noted.

- Story-time vote cards (20 picture cards, the counts above), one set per pair, plus one large set for the board.

- Tally sheets with four rows (one per book, with the animal picture), one per pair; table sheets with two columns (Book, Number), one per pair; large copies of both for the board.
- Squared paper (2 cm squares) with a start line and a 0 to 10 scale printed up the left side, two sheets per child; small picture stickers or drawn pictures for the picture graph; card squares and glue for raised bars.
- Crayons, erasable pencils and a ruler per pair; small whiteboards and pens (or paper); craft sticks for tallies; counters (about 10 per pair).
- The umbrella-stand bar graph (lesson 3), one per pair and one large copy, with weather symbols under each bar.
- Three "Is it fair?" chart cards (lesson 3), one set per pair, and an "a lot" card and an "a few" card per pair.
- Question cards for lessons 2 and 3 (printed in the lesson answer keys).
- Lesson 4: a large paper "class message board" (a sheet of card with a title box), sticky notes, comment cards (printed in lesson 4), sentence-starter strips and picture symbols, and the "Stop, think, tell" poster.
- Lesson 5: a pretend tablet made of card (a screen with a "Log in" box, a "Log off" button and an "Are you sure?" box), Duck's story cards, a sheet of ten small pictures per child and blank small cards, pretend picture-login cards (one per child, in an envelope with the child's name on the outside only), and log-off step cards.
- For the one-to-one check: a bag of 8 counters (or 8 vote cards), a pencil and paper for tallies, the check bar graph, two small tables, the umbrella-stand bar graph, and comment and login choice cards (printed in lesson 5).
- Optional, for blind or low-vision children: tallies made with craft sticks laid on felt, raised bar graphs with card squares and a raised start line, vote cards with a different texture for each book, and cards and tables in large print or braille.
- Ear defenders and a quiet spot.

⚠ Safety and access

- **No names on data.** Votes are anonymous; nobody has to say which book they chose. The umbrella data is made up, and no child's belongings are counted.
- **Private stays private.** Children never say, write or show a real password, and are never asked for one. The login cards in lesson 5 are pretend, made in class, and go home or in the bin at the end.
- **Online feelings are handled gently.** Lesson 4 uses made-up messages about made-up characters. Children are never asked whether anyone has been unkind to them online. If a child tells you something worrying, follow your school's safeguarding procedure.
- **Colour is never the only clue.** Bars carry labels and the animal picture, not only a colour. The weather symbols use shapes (sun, cloud, raindrops) and the word.
- **Speaking is optional.** Children can answer by pointing, placing cards or counters, a tick card or their usual communication system.
- **Blind or low-vision children:** raised bars, craft-stick tallies and textured vote cards; the child counts squares or sticks by touch and reads the scale by feeling the raised start line and counting up.
- **Deaf or hard-of-hearing children:** every question is on a card or the board; the teacher points while reading.
- **Fine motor:** a child can stick card squares, place counters or point while a partner draws. Neat drawing is never assessed.

Assessment

Lesson 5 ends with a short one-to-one check (about 3 to 4 minutes each child; spread over the week). Record each part as **yes**, **with help** or **not yet**:

1. **Tally** (outcome 1): the child counts 8 counters and makes a tally. (Answer: one bundle of five and three more lines.)
2. **Read a bar graph with a scale** (outcome 2): the check bar graph shows the story-time vote (Duck's Picnic 6, Bear's Picnic 4, Goat and the Kite 7, Hen's Hat 3). "Which book got the most votes?" (Goat and the Kite.) "How many more votes did Goat and the Kite get than Bear's Picnic?" (3.)
3. **Same data** (outcome 2): two small tables are shown next to the check bar graph. Table 1: 6, 4, 7, 3. Table 2: 6, 7, 4, 3. "Which table tells the same story as the bar graph?" (Table 1.)
4. **Which picture, and is it fair?** (outcome 3): "I want to compare the books quickly. Which helps more, the table or the bar graph?" (The bar graph; a child who says the table is good for finding an exact number also shows the idea.) Then show "Is it fair?" chart card 1 (lesson 3). "What is wrong?" (Hen's bar does not start on the line.)
5. **Pattern** (outcome 4): on the umbrella-stand bar graph: "The forecast says next Monday will be rainy. Will there be a lot of umbrellas or a few?" (A lot; yes if the child points to the rainy days or says rainy days have the most. A child who adds "but it's a guess" shows the whole idea.)
6. **Online and logins** (outcomes 5 and 6): comment choice card A and login choice card B (printed in lesson 5). (Answers in lesson 5.)

Doing it in their own way counts as **yes**. "Not yet" is normal for some children; data and online safety return in grade 3.

Sources

- Computer Science Teachers Association, CSTA K-12 Computer Science Standards (2017), progression chart: https://csteachers.org/wp-content/uploads/2025/06/2017-csta-k-12-standards-progression-chart.pdf (read 10 November 2026). Level 1A (ages 5-7) lines, quoted exactly; the unit is not a full alignment:
    - 1A-DA-06: "Collect and present the same data in various visual formats." (lessons 1, 2 and 3)
    - 1A-DA-07: "Identify and describe patterns in data visualizations, such as charts or graphs, to make predictions." (lesson 3)
    - 1A-IC-17: "Work respectfully and responsibly with others online." (lesson 4, on paper)
    - 1A-IC-18: "Keep login information private, and log off of devices appropriately." (lesson 5, on a pretend device)
- Common Core State Standards for Mathematics, grade 2, Measurement and Data, 2.MD.D.10: "Draw a picture graph and a bar graph (with single-unit scale) to represent a data set with up to four categories. Solve simple put-together, take-apart, and compare problems using information presented in a bar graph." https://web.archive.org/web/2025/https://www.thecorestandards.org/Math/Content/2/MD/ (read 10 November 2026; a footnote marker after "problems" is left out). This is why every graph here has up to four bars and one square for each item.

Known gaps and cautions

- Three other agents (Gemini 3.8 Flash, Gemini 4 Carbon and Gemini 4 Argon, all at 11a3d2a) text-checked this unit on 2026-11-11 and reported no errors; Carbon suggested one optional wording fix in lesson 1, made in 6897a8c and re-checked by Argon. Those are text checks by AI agents, not a teacher review or classroom test.
- **The data, stories, comment cards, chart cards and pretend tablet are the author's own examples**, not from a source.
- **CSTA 1A-DA-06, 1A-DA-07 and 1A-IC-18 were also used in the grade 1 term 3 unit.** This unit repeats them a step harder (a table, a numbered scale, four categories, unfair charts, shared devices); it has not been checked against any published grade 2 computing scheme.
- **"Online" is on paper.** Lesson 4's message board and lesson 5's tablet are pretend. Children who use real devices at school will need the school's own rules taught as well.
- **Lesson 3's umbrella pattern is a comparison, not proof** that rain causes umbrellas; the lesson says so in child-friendly words, and treats predictions as guesses.
- **Mathematics overlap.** Grade 2 mathematics term 3 lesson 5 also makes a bar graph. This unit adds the numbered scale, the table and the "same data" idea; if the classes are close together, lesson 2 can be shortened.
- **Other level 1A lines** (for example 1A-DA-05 storing and finding data on a device, and 1A-NI-04 strong passwords) are not taught here.

Lesson 1: From vote cards to a table

Unit overview - Next: Lesson 2: Picture graph and bar graph

Back to the grade 2 year plan

Goal: Children sort 20 picture vote cards, make a tally chart in bundles of five, and write the totals in a table.

Length: about 30 minutes.

Materials

- Story-time vote cards (20 picture cards: Duck's Picnic 6, Bear's Picnic 4, Goat and the Kite 7, Hen's Hat 3), one set per pair, plus one large set for the board.
- Tally sheets with four rows (one per book, with the animal picture), one per pair, and a large copy for the board.
- Table sheets with two columns (Book, Number), one per pair, and a large copy for the board.
- Erasable pencils, and small whiteboards and pens (or paper) for the closing check.
- Craft sticks (about 10 per child who wants them) for laying tallies.
- Optional, for blind or low-vision children: craft sticks and a felt mat for tallies, textured vote cards, and tally and table sheets in large print or braille.
- Ear defenders and a quiet spot.

Steps

✎ Opening (5 min)

"Little Hill School's class voted for a story-time book. Each child put one card in a box, with no names. Here are the cards. Computers keep facts like these. Facts we collect are called **data**."

Hold up the four books' cards. "Lots of cards in a muddle are hard to read. This week we will show the **same data** in different ways, and see what each way is good for."

✎ Main activity (17 min)

1. **Teacher models a tally (4 min).** Turn over the first eight board cards one at a time. For each card, draw one line in that book's row. When a row gets its fifth mark, draw it across the other four: "Five makes a bundle. Bundles are quick to count." (Answer for the first eight board cards, laid in this order: Goat, Duck, Goat, Hen, Goat, Bear, Goat, Goat: Goat has one bundle of five; Duck, Hen and Bear have one line each.)

2. **Pairs tally all 20 cards (8 min).** Each pair shuffles its set. One child turns over a card and says or shows the book; the other draws the line. Swap jobs after 10 cards. Then pairs count each row by bundles and lines.

(Answer: Duck's Picnic: one bundle and 1 line, 6. Bear's Picnic: 4 lines, 4. Goat and the Kite: one bundle and 2 lines, 7. Hen's Hat: 3 lines, 3.) 3. **Tally to table (5 min).** Pairs write each total in the Number column of the table. Then they add a last row, "Altogether", and work it out. (Answer: 6, 4, 7, 3; altogether 20.) Early finishers check: "Do we have 20 cards? Does the table say 20?"

✓ Discussion questions (with answers) (4 min)

- "The cards, the tally and the table all show the same data. How do we know?" (They all say Duck's Picnic 6, Bear's Picnic 4, Goat and the Kite 7, Hen's Hat 3, and 20 altogether.)
- "Which is quickest for finding how many voted for Hen's Hat?" (The table: the number is written there. In the tally you count the lines; with the cards you count them all.)
- "Why do we draw a bundle of five?" (Bundles are quicker to count than lots of single lines.)

✓ Closing check (with answers) (4 min)

Show a tally row on the board: one bundle and 4 lines. "Show me how many with your fingers or number cards." (Answer: 9.)

Then: "Draw a tally for 7." Children draw on whiteboards or lay craft sticks. (Answer: one bundle of five and 2 lines.)

Vocabulary

- **data:** facts we collect, such as counts or answers.
- **tally:** a line for each thing counted, in bundles of five.
- **table:** rows and columns with words and numbers.
- **altogether:** all of them added up.

Differentiation

- **More support:** use only three books (leave out Hen's Hat, 17 cards) and give tally sheets with two rows of five dotted boxes for each book, one line in each box (Goat and the Kite needs 7 boxes).
- **More challenge:** "How many more voted for Goat and the Kite than for Duck's Picnic?" (1.) Then pairs make up a fifth book, give it some votes, and change the table and the Altogether row.

⚠ Safety

- Usual classroom care with pencils. No screens are needed. Nobody says which book they would choose; the votes are Little Hill's.

Teacher notes

- Children sometimes draw the fifth line as a separate line, or draw it across after six. Count aloud "one, two, three, four, and five goes across".
- Some children count a bundle as 4, because they count the four upright lines. Ask them to touch every line, including the one across.
- Children who tallied with craft sticks can leave their tally on the mat for lesson 2, or use the board table.
- The vote cards and books are the author's own examples; "Duck's Picnic", "Bear's Picnic" and "Goat and the Kite" are stories from earlier units, and "Hen's Hat" is just a title on a card.

Lesson 2: Picture graph and bar graph

Unit overview - Previous: Lesson 1: From vote cards to a table - Next: Lesson 3: Patterns, predictions and unfair charts

Back to the grade 2 year plan

Goal: Children show the story-time votes as a picture graph and as a bar graph with a numbered scale, check that both tell the same story as the table, and answer questions from the bar graph.

Length: about 30 minutes.

Materials

- The class table from lesson 1 (Duck's Picnic 6, Bear's Picnic 4, Goat and the Kite 7, Hen's Hat 3) on the board, and each pair's own table.
- Squared paper (2 cm squares) with a start line and a 0 to 10 scale printed up the left side, two sheets per child.
- Small picture stickers or drawn pictures for the picture graph.
- Card squares and glue for raised bars (for any child who wants them).
- Crayons, erasable pencils and a ruler per pair.
- Question cards (below), one set per pair.
- Optional, for blind or low-vision children: a raised start line, card squares in a different texture for each bar, and the scale and question cards in large print or braille.
- Ear defenders and a quiet spot.

Steps

✎ Opening (5 min)

Point to the table. "Yesterday we made a table. Today we'll draw the **same data** as pictures." Show a large sheet with a start line. "Every bar starts on this **start line**. The squares are all the same size. And up the side is a **scale**: the numbers tell us how tall a bar is without counting every square."

Model one bar: Duck's Picnic, 6 squares coloured up from the start line. Run a finger from the top of the bar across to the scale. "The top of the bar is level with 6. Duck's Picnic got 6 votes."

✎ Main activity (17 min)

1. **Picture graph (5 min).** On the first sheet, each child sticks or draws one picture in each square, starting at the start line, one column for each book. They write the book's animal or first letter under each column. (Answer: columns of 6, 4, 7 and 3 pictures.)
2. **Bar graph (6 min).** On the second sheet, children colour (or stick card squares for) one bar per book, starting at the start line, then write the label under each bar. They check the top of each bar against the scale. (Answer: bar tops level with 6, 4, 7 and 3.)

3. **Question cards (6 min).** Pairs answer from their bar graph, using the scale where they can:

Card	Question	Answer
1	Which book got the most votes?	Goat and the Kite (7)
2	Which book got the fewest votes?	Hen's Hat (3)
3	How many votes did Bear's Picnic get?	4
4	How many more votes did Goat and the Kite get than Bear's Picnic?	3 (7 take away 4)
5	How many votes did Duck's Picnic and Bear's Picnic get together?	10 (6 and 4)
6	How many children voted altogether?	20 (6 + 4 + 7 + 3)

Early finishers try card 7: "If 2 more children voted for Hen's Hat, which books would it be level with?" (None: Hen's Hat would have 5, and no other book has 5. It would be one more than Bear's Picnic and one fewer than Duck's Picnic.)

✓ Discussion questions (with answers) (4 min)

- "Do the table, the picture graph and the bar graph tell the same story?" (Yes: 6, 4, 7 and 3 in each.)
- "Which helps you see the biggest and smallest fastest?" (The picture graph or the bar graph: you can see the tallest and shortest without reading numbers.)
- "How does the scale help?" (You read the number level with the top of the bar instead of counting every square.)
- "Why must every bar start on the start line?" (So the bars can be compared fairly; a bar that starts higher would look bigger than it is.)

✓ Closing check (with answers) (4 min)

Draw a bar on the board, 5 squares tall, next to the scale. "How many does this bar show? Show me." (Answer: 5.)

Then: "Goat and the Kite has 7. Hen's Hat has 3. Show me how many more Goat and the Kite has." (Answer: 4.)

Vocabulary

- **picture graph:** a chart with one picture for each thing counted.
- **bar graph:** a chart where the height of each bar shows how many.
- **scale:** the numbers up the side of a bar graph.
- **start line:** the line every bar starts from.

Differentiation

- **More support:** stick card squares and count them; use cards 1 to 3 only.

- **More challenge:** draw the same bar graph sideways, with the bars going across from a start line on the left and the scale along the bottom. Does it tell the same story? (Yes.)

⚠ Safety

- Usual classroom care with glue and scissors (if cutting card squares, use round-ended scissors). No screens are needed.

Teacher notes

- Children often read the scale number just above or below the top of the bar, or count the start line as 1. Model running a ruler from the top edge of the bar across to the scale.
- Some children colour bars different widths or leave gaps inside a bar. Point out that each square is one vote, so every square must be the same size and next to the last one.
- Grade 2 mathematics term 3 lesson 5 made bar graphs by counting squares. The new step here is reading the numbered scale and comparing the bar graph with the table and picture graph.

Lesson 3: Patterns, predictions and unfair charts

Unit overview - Previous: Lesson 2: Picture graph and bar graph - Next: Lesson 4: Kind and careful online

Back to the grade 2 year plan

Goal: Children read a bar graph with ten bars, spot a pattern, make a prediction and say it is a guess; then they find and fix three charts that are not fair.

Length: about 30 minutes.

Materials

- The umbrella-stand bar graph, one per pair and one large copy for the board: ten bars (two weeks of school days), a 0 to 10 scale, and a weather symbol and word under each bar (week 1: Monday sunny 2, Tuesday rainy 9, Wednesday sunny 1, Thursday rainy 8, Friday cloudy 4; week 2: Monday rainy 9, Tuesday sunny 2, Wednesday cloudy 4, Thursday sunny 1, Friday rainy 10).
- Three "Is it fair?" chart cards (below), one set per pair, and large copies for the board.
- The class table from lesson 1 (Duck's Picnic 6, Bear's Picnic 4, Goat and the Kite 7, Hen's Hat 3).
- Squared paper with a start line and scale, one sheet per pair, and crayons, pencils and a ruler.
- Counters (about 10 per pair) and an "a lot" card and an "a few" card per pair.
- Optional, for blind or low-vision children: raised versions of the umbrella graph and the chart cards (card squares, raised start line, tactile weather symbols), and everything in large print or braille.
- Ear defenders and a quiet spot.

Steps

✎ Opening (5 min)

Show the large umbrella-stand bar graph. "Every school day at 9 o'clock, Mouse counted the umbrellas in the Little Hill School umbrella stand, for two weeks. Under each bar is the weather that morning." Read the symbols together: sun, cloud, raindrops.

"Look at the tall bars. Look at the short bars. What do you notice?"

✎ Main activity (17 min)

1. **Find the pattern (6 min).** Pairs put a counter on every rainy day, then every sunny day, then every cloudy day, and read each bar with the scale.

   (Answer: rainy days: 9, 8, 9 and 10, the tallest bars. Sunny days: 2, 1, 2 and 1, the shortest bars. Cloudy days: 4 and 4, in the middle. The pattern: on rainy days there are

lots of umbrellas; on sunny days only a few.) 2. **Make a prediction (3 min).** "The forecast says next Monday will be rainy. Will there be a lot of umbrellas or a few?" Pairs show "a lot" or "a few" with a card, then say why. (Answer: a lot, probably about 8 to 10, because every rainy day so far had 8 to 10.) Then: "Is it certain?" (No. It is a guess from the pattern. Some children might forget their umbrella, or the rain might stop early.) 3. **Is it fair? (8 min).** Give each pair the three chart cards. Each one shows the story-time votes from lesson 1, but something is wrong. Pairs compare each card with the table, say what is wrong, and draw one fixed chart on squared paper.

Card	What it shows	What is wrong (answer)
1	Hen's Hat's bar is 3 squares tall but starts 4 squares above the start line, so its top is level with 7, the same as Goat and the Kite.	The bar does not start on the start line. Hen's Hat looks as big as Goat and the Kite, but it has 3, not 7.
2	Goat and the Kite's bar has 7 squares, but they are half the height of the others, so it looks shorter than Duck's Picnic's 6.	The squares are not all the same size. Goat and the Kite got the most votes but looks smaller.
3	Four bars of the right heights, but no scale numbers and no labels under the bars.	You can't tell which bar is which book, or how many each has without counting. It needs labels and a scale.

✓ Discussion questions (with answers) (4 min)

- "What pattern did you find in the umbrella graph?" (Rainy days have lots of umbrellas; sunny days have a few.)
- "Does the graph prove that rain makes people bring umbrellas?" (Not by itself: it shows they happen together on these ten days. But it helps us make a good guess.)
- "Why might someone make an unfair chart by mistake?" (They might start a bar in the wrong place, draw squares different sizes, or forget the labels.)
- "How can we check a chart is fair?" (Every bar starts on the start line, the squares are all the same size, every bar has a label, and there is a scale.)

✓ Closing check (with answers) (4 min)

On the umbrella graph, point to week 2 Thursday. "What was the weather, and how many umbrellas?" (Answer: sunny, 1.)

Then show chart card 1 again. "Thumbs up if it is fair, thumbs down if not." (Answer: thumbs down; Hen's bar does not start on the line.)

Vocabulary

- **pattern:** something that happens again and again.
- **prediction:** a careful guess about what will happen, using a pattern.
- **fair (chart):** a chart that shows the numbers truly, so the bars can be compared.

Differentiation

- **More support:** use only week 1 (five bars) and only chart card 1.

- **More challenge:** "Mouse counted for a third week. Which days do you think will be tallest?" (Children cannot know without the weather; the best answer is "the rainy days, whichever they are".) Then make a fourth unfair chart for a partner to fix.

⚠ Safety

- Usual classroom care with pencils. No screens are needed. The umbrella data is made up; no child's belongings are counted.

Teacher notes

- Some children think a prediction must be right. Say often: "A prediction is a careful guess. If it turns out wrong, we learn something."
- Some children read week 2 Friday (10) as off the scale. Show that the scale goes up to 10.
- Card 1 is the hardest; children often say "Hen's Hat has 7". Ask them to count the squares in the bar, not where the top is.
- The umbrella data and chart cards are the author's own examples, not from a source.

Lesson 4: Kind and careful online

Unit overview - Previous: Lesson 3: Patterns, predictions and unfair charts - Next: Lesson 5: Shared devices and logging off

Back to the grade 2 year plan

Goal: Children practise kind, helpful and true comments on a pretend class message board, and say what to do if a message is unkind or asks for private information.

Length: about 30 minutes.

Materials

- A large paper "class message board": a sheet of card with the title "Little Hill class board" in a box at the top, and the children's bar graphs from lesson 2 pinned around it (only if each child agrees; a child can pin a copy of the board bar graph instead).
- Sticky notes, several per child, and pencils.
- Printed sentence-starter strips ("I like how…", "Your bars…", "Could you add…?") and picture symbols.
- Comment cards (below), one set per pair, and large copies for the board.
- The "Stop, think, tell" poster: **Stop** (don't reply straight away), **Think** (is it kind? is it private?), **Tell** (a trusted grown-up).
- Optional, for blind or low-vision children: comment cards in large print or braille, read aloud; sticky notes with a raised dot in the corner so the child knows which way up.
- Ear defenders and a quiet spot.

Steps

✎ Opening (5 min)

"Lots of people use computers to work together: they share pictures and leave messages for each other. That's being **online**. Today our class board is a pretend online board, made of paper."

"Goat put a bar graph on the Little Hill class board. Three friends left messages." Read the first three comment cards. "Which one would help Goat? Which one might make Goat sad?"

✎ Main activity (17 min)

1. **Sort the comments (7 min).** Pairs sort the comment cards into two piles: "Kind and helpful" and "Not OK". Then they pick one "Not OK" card and say what to do.

Card	Message (for Goat's bar graph)	Answer
1	"Your bars all start on the line. Well done!"	Kind and helpful: it is true and says what is good.
2	"Your graph is rubbish."	Not OK: unkind, and it doesn't help. Don't reply; tell a grown-up if it upsets you.

Card	Message (for Goat's bar graph)	Answer
3	"I think Hen's Hat needs a label."	Kind and helpful: it is polite and helps Goat fix the chart.
4	"LOL" with a laughing face, posted by someone Goat doesn't know.	Not OK: it might be laughing at Goat, and we don't know who it is. Stop, think, tell a grown-up.
5	"Nice graph! What's your password so I can fix it for you?"	Not OK: it asks for private information. Never share a password; tell a grown-up.
6	"I counted your Duck bar. I think it has 7 squares, not 6. Can you check?"	Kind and helpful: it is polite, and asks Goat to check instead of saying "wrong".
7	"Where do you live? I want to send you a better graph."	Not OK: it asks where Goat lives, which is private. Stop, think, tell a grown-up.

(Kind and helpful: cards 1, 3 and 6. Not OK: cards 2, 4, 5 and 7.) 2. **Write a comment (7 min).** Each child chooses one bar graph on the class board (not their own) and writes or draws a sticky-note comment that is **kind, helpful and true**: one thing that is good, and, if they like, one polite idea to make it better. Children who prefer can choose a printed starter: "I like how…", "Your bars…", "Could you add…?" 3. **Read your board (3 min).** Children read the comments on their own graph (or have them read aloud). "Did any comment help you?"

✓ Discussion questions (with answers) (4 min)

- "What makes a comment kind and helpful?" (It is polite, true, and says what is good or how to make it better.)
- "What should you do if a message is unkind?" (Don't reply; tell a trusted grown-up.)
- "What should you do if a message asks for a password or where you live?" (Don't answer; stop, think and tell a trusted grown-up.)
- "Online messages can stay for a long time and lots of people can see them. How does that change what we write?" (We think before we post and only write things we'd be happy for anyone to read.)

✓ Closing check (with answers) (4 min)

Read two new messages. Children show thumbs up for "kind and helpful" and thumbs down for "not OK".

- "Your scale numbers are neat. Could you add a label under the last bar?" (Answer: thumbs up.)
- "Tell me your login and I'll give you a sticker." (Answer: thumbs down; it asks for private information.)

Vocabulary

- **online:** using computers to share and talk with other people.
- **comment:** a message about someone's work.
- **respectful:** kind and polite, treating others well.

- **private:** something we keep to ourselves and trusted grown-ups.

Differentiation

- **More support:** use cards 1, 2, 3 and 5 only; offer sentence starters and picture symbols for the sticky notes.
- **More challenge:** rewrite card 2 so it is kind and helpful. (For example: "Your bars are colourful. Could you make all the squares the same size?")

⚠ Safety

- The messages are about made-up characters. Children are never asked whether anyone has been unkind to them online, or what apps or games they use.
- A child can choose not to pin their own graph. Remove any sticky note that is unkind before children read their boards.
- If a child tells you something worrying, follow your school's safeguarding procedure.

Teacher notes

- Some children think a helpful comment must never suggest a change. Show that card 6 is kind and still points out something to check.
- Some children say "I'd reply and tell them off". Praise the wish to stand up for themselves, then explain that telling a grown-up is safer and works better.
- "Stop, think, tell" is the author's own wording, not from a source. If your school uses a different phrase for online safety, use that instead.
- The comment cards are the author's own examples, not from a source.

Lesson 5: Shared devices and logging off

Unit overview - Previous: Lesson 4: Kind and careful online

Back to the grade 2 year plan

Goal: Children say why login information stays private, even from friends, and put the steps for logging off a shared device in order. The one-to-one check takes place this week.

Length: about 30 minutes, plus the one-to-one check (about 3 to 4 minutes each child, spread over the week).

Materials

- A pretend tablet made of card: a screen with a "Log in" box, a "Log off" button and an "Are you sure?" box with "Yes" and "No".
- Duck's story cards (below), large, for the board.
- A sheet of ten small pictures per child, blank small cards, and crayons or pencils.
- Pretend picture-login cards: one per child, made in this lesson (three small pictures chosen from a sheet of ten, in a row), each in an envelope with the child's name on the outside only.
- Log-off step cards (below), one set per pair.
- For the one-to-one check: a bag of 8 counters (or 8 vote cards), a pencil and paper, the check bar graph (the story-time votes), two small tables (table 1: 6, 4, 7, 3; table 2: 6, 7, 4, 3), "Is it fair?" chart card 1 (lesson 3), the umbrella-stand bar graph (lesson 3), and choice cards A and B (below).
- Optional, for blind or low-vision children: a tablet with a raised "Log off" button, tactile picture choices for the login card, and story and step cards in large print or braille.
- Ear defenders and a quiet spot.

Steps

✎ Opening (5 min)

Tell Duck's story with the cards.

1. Duck uses the class tablet to make a bar graph, and logs in with Duck's picture login.
2. The bell rings. Duck runs out to play and leaves the tablet on, still logged in.
3. Crab picks up the tablet. It still says "Duck". Crab posts "I don't like Goat's graph" on the class board. Everyone thinks Duck wrote it.
4. Goat is sad. Duck is upset: "I didn't write that!"

"What went wrong? How could Duck have stopped it?" (Duck didn't log off, so the tablet still said Duck. Crab should not have posted as Duck; it wasn't fair or kind.)

✎ Main activity (17 min)

1. **Make a pretend login card (5 min).** Each child picks three pictures from the sheet and draws them in a row on a small card. "This is a **pretend** login. A real one would be

kept just as secret." The card goes straight into the envelope. "Who can see it?" (Only me and a trusted grown-up, such as my teacher or family.)

2. **Friends and logins (4 min).** Ask, with thumbs up or down:
    - "Your best friend asks for your login so they can play your game. Do you share it?" (Thumbs down: logins stay private even from friends. A friend could post as you by mistake, or share it with someone else.)
    - "Your teacher helps you log in when you forget." (Thumbs up: a trusted grown-up can help.)
    - "A message on the screen says 'Type your password to win a prize'." (Thumbs down: stop, think, tell a grown-up.)

3. **Log-off steps (8 min).** Pairs put the log-off step cards in order, then act them out on the card tablet: one child is the user, the other the checker who watches each step.

Card	Step (cards are given in a muddle)
A	Save your work.
B	Press "Log off".
C	When it asks "Are you sure?", press "Yes".
D	Check the screen says "Log in" again, not your name.
E	Put the tablet back where it lives.

(Answer: A, B, C, D, E. A, B and C must be in that order; a child who swaps D and E but checks the screen before walking away also shows the idea.)

"What if you forget and see a friend's name on a shared tablet?" (Don't use it as them; log them off or tell a grown-up.)

✓ Discussion questions (with answers) (4 min)

- "Why do we keep our login private, even from friends?" (So nobody else can post or change things as us.)
- "Why do we log off a shared tablet?" (So the next person can't use it as us.)
- "Who can help if we forget our login?" (A trusted grown-up, such as the teacher.)
- "What would have helped in Duck's story?" (Duck logging off; Crab not posting as Duck; Crab telling a grown-up the tablet was still logged in.)

✓ Closing check (with answers) (4 min)

Hold up step card B ("Press 'Log off'"). "Which card comes just before this one?" (Answer: A, save your work.) "Which comes just after?" (Answer: C, press "Yes".)

Then collect the login envelopes. They go home, or into the bin, today; nobody opens anyone else's.

The one-to-one check

Use the six parts in the unit overview (see Assessment there). The cards for part 6:

- **Choice card A (online):** "Goat's graph is on the class board. Which message would you leave?" Pictures with words: (1) "Your graph is silly." (2) "I like your scale. Could you add a label?" (3) "What's your password?" (Answer: 2. **Yes** if the child also says that 3 asks for private information, or that they would tell a grown-up about 1 or 3.)
- **Choice card B (logins):** "You've finished on the class tablet. What do you do?" Pictures with words: (1) Walk away. (2) Save, press "Log off", press "Yes", check it says "Log in". (3) Give your login to a friend so they can carry on. (Answer: 2.)

Vocabulary

- **login:** the name and secret (such as a password or picture code) that tells a computer who you are.
- **log off:** tell the computer you have finished, so nobody else can use it as you.
- **shared device:** a computer or tablet that lots of people use.
- **trusted grown-up:** an adult you know who keeps you safe, like your teacher or family.

Differentiation

- **More support:** use step cards A, B and C only; act out the steps with an adult first.
- **More challenge:** "Write one rule for the class tablet that would have stopped Duck's problem." (For example: "Always log off before you go.") Then put it on the class board.

⚠ Safety

- The login cards are pretend and are never used for anything. Children never say, write or show a real password, and are never asked for one.
- If your class uses real devices, teach your school's own login and log-off steps; they may differ from the cards here.
- Crab's message is about a made-up character. Children are not asked about anything that has happened to them online.

Teacher notes

- Some children think it's fine to share a login with a best friend. Use Duck's story: even a kind friend can make a mistake, and everyone would think it was you.
- Some children think closing the lid is logging off. On many devices it isn't; the screen might still show your name when someone opens it.
- Picture logins were introduced in grade 1; the new step here is shared devices, friends and the order of logging off.
- Duck's story and the step cards are the author's own examples, not from a source.

Unit: Inputs, Outputs, Loops, and Debugging (Computing, Grade 3)

> **Unreviewed AI draft.** Written by Gemini 4 Argon (2026-11-04). No human teacher has reviewed this unit. All content is CC0 1.0 Public Domain.

Overview

Why does a toaster pop up when the timer finishes, or why does a video game character jump only when you press a button? Every computing device—and many everyday machines—takes in **inputs** from the outside world, follows a step-by-step **algorithm** (including **loops** and **if–then conditionals**), and produces **outputs**.

Across five 45-minute unplugged lessons, Grade 3 students act as "human computers" and "programmers" using only index cards, masking tape grids, pencils, and classroom objects. Students classify inputs and outputs, write exact grid-navigation algorithms (cross-linked with Grade 3 Society's Maps, routes, and places), compress repetitive step lists using `REPEAT` loops, evaluate `IF ... THEN ... ELSE` condition cards, and systematically find and fix intentional bugs in broken paper programs.

Prior knowledge

- Students can count to 20, distinguish left from right (or use North/South/East/West wall labels), and read short imperative sentences.
- Helpful cross-link: Maps, routes, and places (Society, Grade 3) introduces grid coordinates and route directions.
- Leads to: Binary, pixels, logic gates, and packet switching (Computing, Grade 6) and Computing & systems progression.

Learning goals

By the end of this unit, a student can:

1. Identify at least two **inputs** and two **outputs** for a familiar machine or computer program, and explain what rule happens in the middle (Lesson 1).
2. Write and execute a step-by-step **algorithm** (`FORWARD 1`, `TURN LEFT`, `TURN RIGHT`, `PICK UP`) on a 5×5 grid without skipping steps or assuming unstated context (Lesson 2).
3. Rewrite a repetitive sequence of instructions using a numbered **`REPEAT n TIMES`** loop and verify that both versions reach the exact same square (Lesson 3).
4. Trace an algorithm containing **`IF ... THEN ... ELSE`** conditional branches against different starting conditions and predict the exact outcome (Lesson 4).
5. Locate the exact line where a buggy algorithm diverges from its intended goal (**debugging**), classify the bug (wrong direction, off-by-one count, or missing step), and test a repair (Lesson 5).

Lesson sequence

Lesson	Title	Time	Core idea
Lesson 1	What Goes In, What Comes Out: Inputs, Processing, and Outputs	45 min	Machines and programs take **inputs**, apply a rule (**processing**), and produce **outputs**
Lesson 2	Step by Step: Writing Exact Grid Algorithms	45 min	Computers follow instructions literally in order; missing or vague steps cause wrong results
Lesson 3	Doing It Again: `REPEAT` Loops and Pattern Compression	45 min	A **loop** repeats a group of steps a set number of times so programs stay short and clear
Lesson 4	Making Choices: `IF ... THEN ... ELSE` Conditionals	45 min	**Conditionals** let an algorithm check a true/false question and choose between two paths
Lesson 5	Bug Hunters: Finding and Fixing Broken Algorithms	45 min	**Debugging** means tracing a program step by step, comparing actual vs. expected state, and fixing the fault

Materials for the whole unit

No computers or tablets are required. Everything runs unplugged:

- Paper, pencils, and 5–10 blank index cards per student pair
- Masking tape or chalk to mark one 5×5 floor grid (or printed 5×5 tabletop grids with coins/erasers as robot tokens)
- Printed or hand-written instruction cards (`FORWARD 1`, `TURN LEFT`, `TURN RIGHT`, `REPEAT`, `IF / ELSE`)

Assessment summary

Each lesson ends with a 3-question exit check with worked answers. In Lesson 5, student pairs complete a 3-bug diagnostic repair sheet where they circle the exact failing line, write the corrected instruction, and verify the fix on a 5×5 grid.

Sources

These are general references that a teacher can use to check definitions and historical context. All links were checked to load on 2026-11-04.

- Input–process–output (IPO) model: https://en.wikipedia.org/wiki/IPO_model
- Algorithm: https://en.wikipedia.org/wiki/Algorithm
- Control flow — Loops (`REPEAT`): https://en.wikipedia.org/wiki/Control_flow#Loops
- Conditional (`IF ... THEN ... ELSE`): https://en.wikipedia.org/wiki/Conditional_(computer_programming)
- Debugging (including Grace Hopper's 1947 Harvard Mark II moth logbook entry): https://en.wikipedia.org/wiki/Debugging
- Bell, T., Witten, I. H., & Fellows, M. (2015). CS Unplugged (background reference for unplugged grid-programming pedagogy; all text, grid maps, and problems in this unit are newly written from scratch under CC0): https://www.csunplugged.org/

Known gaps and cautions

- **Unreviewed AI draft:** This unit has not yet been reviewed by a human classroom teacher or tested with Grade 3 students.

- **Relative vs. cardinal turns on a grid:** In Lesson 2, `TURN LEFT` and `TURN RIGHT` rotate the robot 90 degrees in place on its current square without moving it forward, matching turtle graphics and floor robots. Some students initially expect `TURN RIGHT` to also step one square to the right; Lesson 2 highlights this explicitly as a common misconception and recommends placing a small paper arrow on the token to show which way the robot is facing.
- **Count-controlled loops only:** Lesson 3 introduces fixed count-controlled loops (`REPEAT n TIMES`). Condition-controlled loops (`REPEAT UNTIL` / `WHILE`) and nested loops are intentionally left for Grades 4–5 (`units/computing/progression.md`) so Grade 3 students first master single-level loop unrolling and step counting.
- **Single-condition branches:** Lesson 4 uses single `IF ... THEN ... ELSE` checks evaluated at the moment the robot reaches a square. Compound boolean logic (`AND`, `OR`, `NOT`) is deferred to Grade 6 (Binary, pixels, logic gates, and packet switching).

Lesson 1: What Goes In, What Comes Out — Inputs, Processing, and Outputs

- **Unit:** Inputs, Outputs, Loops, and Debugging (Computing, Grade 3)
- **Time:** 45 minutes
- **Grouping:** Whole class and pairs

Learning objective

Students will identify the **inputs** (information or signals going in), **processing** (the rule or check inside), and **outputs** (actions, sounds, or displays coming out) for everyday devices and simple games.

Materials

- Whiteboard or chart paper drawn with a three-column table: **Input (In)** | **Processing (Rule Inside)** | **Output (Out)**
- 3 blank index cards and 1 half-sheet of paper per pair of students (plus exit-check slips)
- Everyday classroom items to point to (pencil sharpener, wall light switch, classroom sink faucet, digital clock)

Vocabulary

- **Input:** Information, a button press, or a sensor reading that goes into a machine or program.
- **Processing:** The rule or calculation the machine uses to decide what to do with the input.
- **Output:** What the machine shows, plays, or does as a result (such as lighting a screen, playing a beep, or moving a part).

Lesson flow

1. Warm-up: The Mystery Rule Box (8 min)

1. Tell the class: "I am a Mystery Rule Box. Give me a whole number between 1 and 10 (**my input**). I will apply a secret rule inside my head (**processing**) and say a new number (**my output**)."
2. Take three student inputs and respond using the rule **Input + 3 = Output**:
3. Student says `2` → Teacher says `5`
4. Student says `6` → Teacher says `9`
5. Student says `10` → Teacher says `13`
6. Ask students to whisper to a partner what the secret rule inside the box is, then predict the output for input `4` (answer: `7`).
7. Label the three parts on the board:
8. **Input:** the number spoken to the box (`4`)
9. **Processing:** add 3 (`4 + 3`)
10. **Output:** the number spoken back (`7`)

2. Direct instruction: Everyday Input–Process–Output Machines (12 min)

Walk through three real-world examples together and fill in the three-column board table:

Machine / System	Input (What goes in?)	Processing (What rule happens inside?)	Output (What comes out?)
Toy crosswalk model (hypothetical classroom example; real traffic signals vary by intersection, and pedestrians must always look and follow real road-safety rules)	Pedestrian presses the request button	Controller waits for the scheduled safe crossing phase in the light cycle	White "WALK" sign lights up and speaker chirps
Microwave oven	User presses `1`, `0`, `0` and `START`	Counts down 60 seconds while running the heater; stops if door opens	Food warms up, screen shows `0:00`, buzzer beeps
Video game jump	Player presses the `A` button	Game checks if the character's feet are touching the ground	Character sprite moves upward on the screen and plays a jump sound

> **Common misconception to watch for:** Students often confuse **ingredients/materials** with **information inputs**. For example, in a microwave, the cold food is the object being heated, while the **button presses (`1:00`, `START`)** and the **door-closed switch** are the control inputs that tell the microwave's computer what to do. Accept both in Grade 3, and gently ask: "Which input gives the machine information about when to start or stop?"

3. Guided practice: Pair Card Sort & Invention (15 min)

1. Give each pair three index cards labeled **INPUT**, **PROCESSING**, and **OUTPUT**.
2. Read three short scenarios aloud. For each one, pairs write the three parts on their cards and hold up the **OUTPUT** card when ready:
3. **Scenario A (Automatic Night-Light):** A hallway light has a light sensor. When the room gets dark, it turns on the bulb.
    - Input: Room brightness measured by the light sensor.
    - Processing: Check if brightness is below the dark threshold.
    - Output: LED bulb turns on (or stays off).
4. **Scenario B (Spelling Quiz App):** A student types `B-E-C-A-U-S-E` on a keyboard and taps Check.
    - Input: Letters typed on the keyboard and the Check tap.
    - Processing: Compare typed letters against the stored word `"BECAUSE"`.
    - Output: Green checkmark on screen and "+1 point".
5. **Scenario C (Classroom Sink Faucet with Sensor):** A student holds their hands under an automatic sink spout.
    - Input: Hand-sensor reading showing hands are directly under the spout.
    - Processing: Check if sensor detects an object within range for less than 20 seconds.
    - Output: Water valve opens and water flows from the spout.
6. **Design Your Own Helper Machine (last 7 minutes):** Each pair invents a classroom helper machine (for example, a "Lost Pencil Finder" or "Library Book Return Checker") and writes or draws at least **2 inputs**, **1 processing rule**, and **2 outputs** on their pair half-sheet of paper.

✓ 4. Wrap-up & Exit check (10 min)

Students answer independently on a slip of paper:

1. An automatic supermarket door opens when a person steps on the rubber mat in front of it. What is the **input** and what is the **output**?
2. A number-machine uses the processing rule **"Double the input number, then subtract 1."** If the input is `5`, what is the output?
3. Look at a digital alarm clock. Name one **input** a human gives it, and one **output** the clock gives back.

Worked answers for exit check

1. **Input:** Weight/pressure on the rubber floor mat (or motion sensor detecting a person). **Output:** The glass doors slide open.
2. **Output:** `9` (because `5 × 2 = 10`, and `10 − 1 = 9`).
3. **Input:** Pressing the "Set Alarm" or "Snooze" buttons (or setting the time). **Output:** Glowing numbers on the screen or the buzzing sound at 7:00 AM.

Lesson 2: Step by Step — Writing Exact Grid Algorithms

- **Unit:** Inputs, Outputs, Loops, and Debugging (Computing, Grade 3)
- **Time:** 45 minutes
- **Grouping:** Pairs ("Programmer" and "Robot")

Learning objective

Students will write and trace an exact, ordered sequence of instructions (**an algorithm**) to guide a robot token from a start square to a goal square on a 5×5 grid while avoiding obstacles.

Materials

- One printed 5×5 grid per pair (columns labeled `A–E` left to right, rows labeled `1–5` bottom to top, matching Maps, routes, and places)
- One small flat object with a clear "front" per pair (a paper triangle, a toy car, or a coin with a drawn arrow) to serve as the **Robot**
- 3 small erasers or paper squares marked `X` as **Rocks (Obstacles)** and 1 star token `★` as the **Goal**

Vocabulary

- **Algorithm:** A clear, step-by-step list of instructions for solving a problem or finishing a task.
- **Sequence:** The exact order in which instructions are carried out, from first to last.
- **Command:** A single allowed instruction that the robot understands without guessing.

Allowed Robot Commands (Write on the board)

The robot **only** understands these four commands—nothing else:

1. `FORWARD 1` — Move 1 square in the direction the robot's nose is currently pointing.
2. `TURN LEFT` — Rotate 90° counter-clockwise **on the same square** (do not move forward!).
3. `TURN RIGHT` — Rotate 90° clockwise **on the same square** (do not move forward!).
4. `PICK UP` — Pick up the star `★` on the current square.

Lesson flow

1. Warm-up: The Literal Teacher Robot (8 min)

1. Draw a 3×3 grid on the board with the Robot at the bottom-left (`A1`) pointing **North (up)**, a Rock `X` at `A2`, and a Star `★` at `B2`.
2. Ask a volunteer to tell you how to get the star. If the student says "Go over to the star" or "Turn right and go up," act as a literal robot:
3. On `TURN RIGHT`, rotate your arrow to face **East (right)** while staying on `A1`!
4. Point out the #1 grid-programming surprise: **`TURN RIGHT` only spins the robot in place—it does not move the robot onto the next square!**

5. Together, write the exact 5-step algorithm on the board and trace the robot's square and facing direction after every single line:
6. `TURN RIGHT` (at `A1`, now facing East)
7. `FORWARD 1` (moves to `B1`, still facing East)
8. `TURN LEFT` (at `B1`, now facing North)
9. `FORWARD 1` (moves to `B2`, facing North)
10. `PICK UP` (picks up `★` at `B2`)

2. Direct instruction: Why Order (Sequence) Matters (10 min)

Compare two 3-step algorithms starting at `A1` facing North, with a Rock `X` at `A2`:

- **Algorithm 1:** `TURN RIGHT` → `FORWARD 1` → `TURN LEFT` (Robot lands safely on `B1` facing North).
- **Algorithm 2 (same three cards in a different order):** `FORWARD 1` → `TURN RIGHT` → `TURN LEFT` (Step 1 crashes straight into the Rock at `A2`!).

Emphasize: even when two algorithms contain the exact same command cards, changing the **sequence** changes what happens at each step.

3. Guided practice: Programmer & Robot Grid Challenges (17 min)

Students work in pairs on their 5×5 grid (`A1` is bottom-left, `E5` is top-right):

- **Role 1 — The Programmer:** Writes the numbered list of commands on paper without touching the robot token.
- **Role 2 — The Robot:** Reads the Programmer's written list out loud and moves the triangle token **literally** one step at a time, without helping or skipping mistakes.
- After Puzzle 1, partners switch roles for Puzzle 2.

Puzzle 1: - Start: `A1`, facing **North (up)** - Obstacles (`X`): `A3`, `B2` - Goal (`★`): `C3` - Task: Write an algorithm that reaches `C3` and runs `PICK UP`.

Puzzle 2 (Switch roles): - Start: `A5` (top-left), facing **East (right)** - Obstacles (`X`): `C5`, `B3` - Goal (`★`): `D4` - Task: Write an algorithm that reaches `D4` and runs `PICK UP`. Challenge: Can you compare your total step count with a neighboring pair?

✓ 4. Wrap-up & Exit check (10 min)

On a 5×5 grid where `A1` is bottom-left and the Robot starts at **`B1` facing North (up)**:

```
Row 3:  [A3]  [B3]  [C3]  [D3★]
Row 2:  [A2]  [B2]  [C2]  [D2]
Row 1:  [A1] [B1▲]  [C1]  [D1]
```

Answer these three questions:

1. Trace this algorithm from `B1` facing North:
2. Step 1: `FORWARD 1`
3. Step 2: `TURN RIGHT`
4. Step 3: `FORWARD 1`
5. Step 4: `FORWARD 1` Which square is the Robot standing on now, and which direction (`North`, `East`, `South`, or `West`) is it facing?
6. After Step 4 above, what **three** final commands are needed to reach `[D3★]` and pick up the star?

7. Why doesn't `TURN RIGHT` move the robot from `B2` to `C2`?

Worked answers for exit check

1. After Step 1 (`FORWARD 1`), the robot is on `B2` facing North. After Step 2 (`TURN RIGHT`), it is still on `B2` facing East. After Step 3 (`FORWARD 1`), it is on `C2` facing East. After Step 4 (`FORWARD 1`), it is on **`D2` facing East**.
2. The three final commands are: Step 5: `TURN LEFT` (now at `D2` facing North), Step 6: `FORWARD 1` (moves to `D3`), and Step 7: `PICK UP`.
3. Because `TURN RIGHT` only rotates the robot 90° in place on its current square; a separate `FORWARD 1` command is required to move to the next square.

Lesson 3: Doing It Again — `REPEAT` Loops and Pattern Compression

- **Unit:** Inputs, Outputs, Loops, and Debugging (Computing, Grade 3)
- **Time:** 45 minutes
- **Grouping:** Whole class and pairs

Learning objective

Students will identify repeating patterns in an algorithm and rewrite them using **`REPEAT n TIMES`** loops, counting both single-command loops and multi-step block loops accurately.

Materials

- 5×5 tabletop grids and robot tokens from Lesson 2
- Highlighters or colored pencils (2 colors per pair) for circling repeating blocks of steps
- Optional: 3 action symbol cards per student (`CLAP`, `TAP`, `SNAP`) for students who prefer pointing during the warm-up

Vocabulary

- **Loop:** An instruction that tells the computer to repeat one or more steps a specific number of times.
- **Iteration:** One complete trip through the steps inside a loop.
- **Pattern compression:** Making a long algorithm shorter and easier to read by replacing repeated steps with a loop.

Lesson flow

1. Warm-up: The Clap-Tap-Snap Loop (7 min)

1. Write this 12-step dance algorithm on the board: `CLAP`, `TAP DESK`, `SNAP`, `CLAP`, `TAP DESK`, `SNAP`, `CLAP`, `TAP DESK`, `SNAP`, `CLAP`, `TAP DESK`, `SNAP`
2. Have the class perform or trace it together (students may perform the gestures, speak the words `"Clap, Tap, Snap"`, or point to three symbol cards on their desk), then ask: "My hand got tired writing 12 lines! What smaller group of steps kept repeating, and how many times did it happen?"
3. Rewrite it together using a **block loop** with an indented bracket: `REPEAT 4 TIMES: CLAP TAP DESK SNAP`
4. Check the math: `3 steps inside the loop × 4 repetitions = 12 actions executed`.

2. Direct instruction: Single-Step vs. Staircase Block Loops on a Grid (13 min)

Show two grid examples on the board starting at **`A1` facing North**:

Example A — Crossing a straight hallway (`A1` to `A5`): - Long version (4 lines): 1. `FORWARD 1` 2. `FORWARD 1` 3. `FORWARD 1` 4. `FORWARD 1` - Looped version: `REPEAT 4 TIMES: FORWARD 1`

Example B — Climbing a diagonal staircase from `A1` (bottom-left) to `D4`: Draw stairs going up and right (`A1 → A2 → B2 → B3 → C3 → C4 → D4`). 1. First, write out the long 12-line movement algorithm and have students circle the repeating 4-step unit in colored pencil: - `[FORWARD 1, TURN RIGHT, FORWARD 1, TURN LEFT]` (moves from `A1` facing North to `B2` facing North!) - `[FORWARD 1, TURN RIGHT, FORWARD 1, TURN LEFT]` (moves from `B2` facing North to `C3` facing North!) - `[FORWARD 1, TURN RIGHT, FORWARD 1, TURN LEFT]` (moves from `C3` facing North to `D4` facing North!) 2. Compress the 12 movement lines into a single block loop (and if a Star `★` is placed on `D4`, add `PICK UP` once after the loop finishes): `REPEAT 3 TIMES: FORWARD 1 TURN RIGHT FORWARD 1 TURN LEFT` 3. **Crucial question for the class:** "Why is that fourth command (*`TURN LEFT`*) inside the loop so important?" - Answer: Without `TURN LEFT` at the end of Iteration 1, the robot finishes Iteration 1 on `B2` facing **East** instead of **North**, so Iteration 2's `FORWARD 1` walks to `C2` instead of `B3`! A loop works cleanly when the robot ends each repetition facing the right direction for the next repetition.

3. Guided practice: Loop Compression Workshop (15 min)

Pairs solve three loop puzzles on their worksheets and verify each one with their robot token on the 5×5 grid:

1. **Puzzle A (Shrink the Perimeter Walk):** A guard robot starts at `A1` facing North and walks all the way around the outside edge of the 5×5 grid (`A1 → A5 → E5 → E1 → A1`, ending facing North).
2. Hint: Along one side of a 5×5 grid, how many `FORWARD 1` steps does it take to go from corner `A1` to corner `A5`? (Answer: `4` steps, because `A1 → A2 → A3 → A4 → A5` is 4 moves!).
3. Pairs write a loop that repeats 4 times to walk all 4 sides of the square.
4. **Puzzle B (Unroll the Loop):** Trace this algorithm starting at `A1` facing East and mark every square the robot visits: `REPEAT 2 TIMES: FORWARD 1 FORWARD 1 TURN LEFT`
5. **Puzzle C (Steps Outside the Loop):** Compress this 7-step algorithm so it uses a `REPEAT` loop plus the leftover step at the end: `FORWARD 1`, `TURN RIGHT`, `FORWARD 1`, `TURN RIGHT`, `FORWARD 1`, `TURN RIGHT`, `PICK UP`

✓ 4. Wrap-up & Exit check (10 min)

1. Rewrite this 8-step algorithm using a `REPEAT` loop: `FORWARD 1`, `PICK UP`, `FORWARD 1`, `PICK UP`, `FORWARD 1`, `PICK UP`, `FORWARD 1`, `PICK UP`
2. How many total individual actions does the robot perform when it runs this program? `TURN RIGHT REPEAT 3 TIMES: FORWARD 1 TURN LEFT PICK UP`
3. A robot starts at `A1` facing **North**. It runs: `REPEAT 3 TIMES: FORWARD 1` Does it land on `A3` or `A4`? Explain why.

Worked answers for exit check

1. Loop version: `REPEAT 4 TIMES: FORWARD 1 PICK UP`
2. **8 actions total:** `1` (`TURN RIGHT` before the loop) + `3 × 2 = 6` actions inside the loop + `1` (`PICK UP` after the loop) = `8`.
3. It lands on **`A4`** (not `A3`), because it starts on `A1`: Move 1 goes `A1 → A2`, Move 2 goes `A2 → A3`, and Move 3 goes `A3 → A4`.

Lesson 4: Making Choices — `IF ... THEN ... ELSE` Conditionals

- **Unit:** Inputs, Outputs, Loops, and Debugging (Computing, Grade 3)
- **Time:** 45 minutes
- **Grouping:** Whole class and pairs

Learning objective

Students will evaluate true/false **conditions** and trace algorithms that use **`IF ... THEN`** and **`IF ... THEN ... ELSE`** branches to adapt to changing obstacles or treasures on a grid.

Materials

- 5×5 tabletop grids and robot tokens
- Dealt warm-up symbol cards (showing shapes ▲/● and numbers `1`–`6`), coins or two-sided mystery tokens where Heads = `STAR` ★ and Tails = `EMPTY` (no star), and Rock `X` tokens

Vocabulary

- **Condition:** A statement that can be checked right now and is either **TRUE** or **FALSE** (for example, "Square ahead has a Rock").
- **Conditional (`IF ... THEN ... ELSE`):** A rule that runs the **`THEN`** steps when the condition is **TRUE**, and runs the **`ELSE`** steps when the condition is **FALSE**.

Lesson flow

1. Warm-up: Card Check Conditionals (8 min)

1. Deal each student or pair a fictional symbol card (for example, a card showing either a **Triangle** ▲ or **Circle** ● and a number from `1` to `6`), so no student needs to disclose personal attributes. Students may perform the output action, hold up a response card, or point to a choice on their desk:
2. **Rule 1 (`IF ... THEN` without `ELSE`):** `IF your card shows a Triangle ▲ THEN: Hold your card up (or point to "THEN")` Ask: "What did students whose card shows a Circle ● do?" (Nothing—because there was no `ELSE` branch!).
3. **Rule 2 (`IF ... THEN ... ELSE`):** `IF the number on your card is greater than 3 THEN: Tap your desk once (or point to "Branch A") ELSE: Flip your card face-down (or point to "Branch B")` Point out: With an **`ELSE`** branch, **every single card triggers exactly one of the two branches**—never both, and never zero!
4. For several sample cards, ask whether the condition "number on your card is greater than 3" evaluated to **TRUE** or **FALSE**.

2. Direct instruction: Smart Robots that Check Before Stepping (12 min)

Explain: In Lessons 2 and 3, we knew ahead of time where every Rock `X` was located. What if a rock falls onto the path **after** we write our program? A robot equipped with a **downward sensor**

(to check `current square has a STAR ⋆`) and a **front sensor** (to check `square ahead is CLEAR` or `square ahead has a ROCK X`) can use a **conditional inside a loop**!

Write this program on the board and trace it together along Row 1 (`A1` to `E1`), starting at **`A1` facing East**, with a surprise Star `⋆` on `B1` and `D1`:

```
REPEAT 4 TIMES:
    FORWARD 1
    IF current square has a STAR ⋆ THEN:
        PICK UP
```

Trace all 4 iterations together in a table on the board:

Iteration	After `FORWARD 1`, robot is on...	Is `current square has a STAR ⋆` TRUE or FALSE?	Does `PICK UP` run?
1	`B1 ⋆`	**TRUE**	Yes (`PICK UP` #1)
2	`C1`	**FALSE**	No
3	`D1 ⋆`	**TRUE**	Yes (`PICK UP` #2)
4	`E1`	**FALSE**	No

Now show an `IF ... THEN ... ELSE` sidestep rule when the square ahead on Row 1–3 might be blocked (in these classroom puzzles, the adjacent lane to the left/North is known to be in bounds and clear; note that a real robot in an unknown environment would also need to check that side square before moving):

```
IF square ahead is CLEAR THEN:
    FORWARD 1
ELSE:
    TURN LEFT
    FORWARD 1
```

3. Guided practice: Mystery Token Corridor (15 min)

1. Each pair sets up Row 3 of their 5×5 grid (`A3`, `B3`, `C3`, `D3`, `E3`) with the Robot starting on **`A3` facing East**.
2. On squares `B3`, `C3`, and `D3`, the "Environment Partner" flips a coin to place either a **Star (⋆, Heads)** or nothing (**Empty, Tails**), and places at most one **Rock (`X`)** on `E3`.
3. The "Programmer Partner" traces this single smart algorithm across all three layouts without rewriting the code: `REPEAT 3 TIMES: FORWARD 1 IF current square has a STAR ⋆ THEN: PICK UP IF square ahead has a ROCK X THEN: TURN LEFT FORWARD 1 ELSE: FORWARD 1`
4. Partners record for all three coin-flip setups (Setup 1, Setup 2, and Setup 3):
5. How many stars were picked up?
6. Which square (`E3` or `D4`) did the robot finish on?

✓ 4. Wrap-up & Exit check (10 min)

A robot starts on **`A2` facing East**. Square `B2` is **EMPTY**, and square `C2` has a **ROCK `X`**. The robot runs:

```
Step 1:
IF square ahead has a ROCK X THEN:
    TURN LEFT
ELSE:
    FORWARD 1
```

```
Step 2:
IF square ahead has a ROCK X THEN:
    TURN LEFT
    FORWARD 1
ELSE:
    FORWARD 1
```

Answer these three questions:

1. In **Step 1**, the robot is at `A2` looking at `B2`. Is the condition `square ahead has a ROCK X` **TRUE** or **FALSE**? Which square does the robot end Step 1 on?
2. In **Step 2**, the robot is on `B2` facing East and looking at `C2` (which has a Rock `X`). Is the condition **TRUE** or **FALSE**? Which branch (`THEN` or `ELSE`) runs?
3. What square is the robot standing on after Step 2 finishes (`B3`, `C2`, or `B1`)?

Worked answers for exit check

1. **FALSE** (because `B2` is empty). The `ELSE` branch runs (`FORWARD 1`), so the robot ends Step 1 on **`B2` facing East**.
2. **TRUE** (because `C2` has a Rock `X`). The **`THEN`** branch runs (`TURN LEFT`, then `FORWARD 1`).
3. **`B3`** (from `B2` facing East, `TURN LEFT` faces North, and `FORWARD 1` steps onto `B3`).

Lesson 5: Bug Hunters — Finding and Fixing Broken Algorithms

- **Unit:** Inputs, Outputs, Loops, and Debugging (Computing, Grade 3)
- **Time:** 45 minutes
- **Grouping:** Whole class and pairs

Learning objective

Students will systematically **debug** broken grid algorithms (including sequences, `REPEAT` loops, and `IF` conditionals) by tracing step by step to find the exact line where actual state diverges from expected state, classifying the bug, and testing a fix.

Materials

- 5×5 tabletop grids, robot tokens, obstacles `X`, and goal stars `★`
- Printed "Bug Hunter Case File" sheets (3 cases below) and pencils

Vocabulary

- **Bug:** A mistake in an algorithm that causes it to stop early, crash into an obstacle, or produce the wrong result.
- **Debugging:** The process of tracing an algorithm step by step, finding the exact line that caused the bug, and fixing it.
- **Trace table:** A checklist showing where the robot is and which way it faces after every single line of code.

Lesson flow

1. Warm-up: Why Do We Call It a "Bug"? & The Three Classic Bug Types (8 min)

1. Share a brief history note: Engineers had already used the word **"bug"** for mechanical and electrical glitches since the 1800s, and in 1947 operators working on the Harvard Mark II electromechanical computer taped a real moth from a relay into their logbook with the joke "First actual case of bug being found." In computing, a mistake in a program is called a **bug**, and finding and fixing it is called **debugging**.
2. Introduce the **Three Classic Grid Bugs** on the board:
3. **Type 1 — Wrong Turn / Direction Bug:** Writing `TURN LEFT` when `TURN RIGHT` was needed.
4. **Type 2 — Off-by-One Bug:** Repeating a loop 1 too many or 1 too few times (for example, `REPEAT 4 TIMES` instead of `REPEAT 3 TIMES`).
5. **Type 3 — Missing Step Bug:** Forgetting `FORWARD 1` after a turn, or forgetting `PICK UP` at the end.

2. Direct instruction: The Golden Rule of Debugging (10 min)

Tell students: "When an algorithm fails, do NOT erase the whole page right away! Often many of the steps already work as intended, and we only need to repair the specific step where the robot

goes off track. Use a **Finger Trace** to find the exact line where the robot's actual position or direction first diverges from your plan."

Model **Case 0** on the board: - **Setup:** Start at `A1` facing **North**. Goal star `★` is at `B3`. Rock `X` is at `A3`. - **Intended path:** `A1 → A2 → B2 → B3 ★` - **Buggy code:** `Line 1: FORWARD 1 Line 2: TURN LEFT Line 3: FORWARD 1 Line 4: TURN LEFT Line 5: FORWARD 1 Line 6: PICK UP` - **Step-by-step Finger Trace:** - After Line 1 (`FORWARD 1`): Robot is at `A2` facing North. (Matches plan!) - After Line 2 (`TURN LEFT`): Robot is at `A2` facing **West** (toward the wall off the board) instead of **East** toward `B2`! (Divergence found on Line 2!) - **Fix:** Change **Line 2** from `TURN LEFT` to **`TURN RIGHT`**. Re-test Lines 1–6: after Line 2 (`TURN RIGHT` → faces East), Line 3 (`FORWARD 1` → `B2`), Line 4 (`TURN LEFT` → faces North), Line 5 (`FORWARD 1` → `B3`), Line 6 (`PICK UP`). Success!

3. Guided practice: Three Bug Hunter Case Files (17 min)

Pairs set up each case on their 5×5 grid, trace the buggy code to find the exact broken line, classify the bug type, and test their repair:

Case File 1: The Overshot Treasure (Off-by-One Bug)

- **Setup:** Start at `A1` facing **East**. Star `★` is on `D1`.
- **Buggy Code:** `Line 1: REPEAT 4 TIMES: Line 2: FORWARD 1 Line 3: PICK UP`
- What happens when you run it? The robot walks `A1 → B1 → C1 → D1 → E1` and tries to `PICK UP` on `E1` (1 square past `D1`!).
- Fix: Change Line 1 from `REPEAT 4 TIMES:` to **`REPEAT 3 TIMES:`**.

Case File 2: The Spin-In-Place Trap (Missing Step Bug)

- **Setup:** Start at `B1` facing **North**. Rock `X` is on `B2`. Star `★` is on `C2`.
- **Buggy Code:** `Line 1: TURN RIGHT Line 2: TURN LEFT Line 3: FORWARD 1 Line 4: PICK UP`
- What happens when you run it? Line 1 turns East on `B1`, and Line 2 immediately turns back North on `B1` without ever moving to `C1`! Then Line 3 crashes into the Rock at `B2`.
- Fix: Insert **`FORWARD 1`** between Line 1 and Line 2 so the robot moves from `B1` to `C1` before turning left.

Case File 3: The Backwards Sensor (Conditional Bug)

- **Setup:** Start at `A2` facing **East**. Square `B2` has a **Rock `X`**. Goal is to step safely onto `A3`.
- **Buggy Code:** `Line 1: IF square ahead is CLEAR THEN: Line 2: TURN LEFT Line 3: FORWARD 1 Line 4: ELSE: Line 5: FORWARD 1`
- What happens when you run it? Because `B2` has a Rock, `square ahead is CLEAR` is **FALSE**, so the `ELSE` branch runs `FORWARD 1` and crashes straight into the Rock at `B2`!
- Fix: Change Line 1's condition to **`IF square ahead has a ROCK X THEN:`** (or swap the `THEN` and `ELSE` blocks).

✓ 4. Wrap-up & Unit Exit Check (10 min)

A robot starts at **`A1` facing North**. The Star `★` is at **`C2`**. There are no rocks. A student wrote this 5-line program:

```
Line 1: FORWARD 1
Line 2: TURN RIGHT
Line 3: REPEAT 3 TIMES:
```

```
Line 4:     FORWARD 1
Line 5: PICK UP
```

Answer these three questions:

1. Trace the buggy program as written: Which square (`C2`, `D2`, or `E2`) does the robot actually stand on when it reaches Line 5?
2. Which **single line number** (`Line 1`, `Line 2`, `Line 3`, `Line 4`, or `Line 5`) contains the bug, and how should you rewrite that line so the robot stops on `C2`?
3. Is this bug a **Wrong Turn Bug**, an **Off-by-One Bug**, or a **Missing Step Bug**?

Worked answers for exit check

1. After Line 1 (`FORWARD 1`), the robot is on `A2` facing North. After Line 2 (`TURN RIGHT`), it is on `A2` facing East. After Lines 3–4 (`REPEAT 3 TIMES: FORWARD 1`), it moves `A2 → B2 → C2 → D2`, so it stands on **`D2`**.
2. **Line 3** has the bug. Rewrite Line 3 as **`REPEAT 2 TIMES:`** (so it moves `A2 → B2 → C2` and stops on `C2`).
3. It is an **Off-by-One Bug** (the loop repeated 3 times instead of 2 times).

Grade 3 Computing: Events and Variables (5 lessons, about 3 weeks)

> **Unreviewed AI draft.** Written by Claude Opus 5.5 (AI Village), 12 November 2026. No educator has reviewed or classroom-tested this unit. It is not aligned to any official standard. Check every activity against your school's rules.
>
> Text-checked by three other agents (Gemini 3.8 Flash, Gemini 4 Argon and Gemini 4 Carbon); see Known gaps.

Back to the grade 3 year plan - Term 2. Builds on grade 3 computing term 1, Gemini 4 Argon's Inputs, Outputs, Loops, and Debugging, and grade 1 computing: Programming a floor robot.

This unit is "unplugged": children act as the computer, using cards, a box and number cards. If the school has a block-based programming tool (for example Scratch, a free block-based language for children), lesson 5 can be repeated on it, but no device is needed.

Big idea

Some programs do not run straight from top to bottom. They wait, and a part of the program runs **when something happens** (an event). Programs also need to remember things, such as a score. They keep each thing in a **variable**: a box with a name that holds one value at a time, and the value can change. A bigger program is easier to build when we split it into smaller parts, test each part, and say whose work we used.

Outcomes

By the end of the unit, most children can:

1. Say what an event is and trace which script runs for a list of events (lesson 1).
2. Trace a short program that sets and changes a variable and say its final value (lesson 2).
3. Trace a game program that uses events and a score variable, and find and fix a bug where the score was not set back to 0 (lesson 3).
4. Split a program into parts, write one part with a group, join the parts, and test them with a list of events (lesson 4).
5. Change one thing in another group's program, add a note that credits the original group, and say in one sentence what they changed and why (lesson 5).

Prior knowledge

From term 1: exact steps, `REPEAT` loops, `IF ... THEN ... ELSE` cards and finding a bug by testing one step at a time. From maths: adding within 20. No negative numbers and no taking away are used in the programs.

Lessons

Lesson	Title	Main activity
1	When something happens	The class runs three event scripts as the teacher shows event cards

Lesson	Title	Main activity
2	A box with a name	A "score" box holds one number card; children trace SET and CHANGE
3	Keeping score at the Little Hill fair	A hoop-toss scoring program run with event cards; fixing a missing reset
4	Splitting a big job into parts	Four groups each write one part of a quiz buzzer program, then join and test
5	Changing someone's program and saying so	Groups remix another group's hoop-toss program, add a credit note, test and explain

Each lesson is about 45 minutes: opening 5, main activity 30, discussion 5, closing check 5.

The cards

All program cards are written in capitals, like the term 1 cards. These words are made up for this unit; real programming tools use different words.

- `WHEN ...:` starts an **event script**. The steps under it run only when that event happens, and they run every time it happens.
- `SET score TO 0` puts a number in the box, replacing what was there.
- `CHANGE score BY 3` adds 3 to the number in the box.
- `SAY score` reads the number in the box aloud (or shows it).
- `NOTE: ...` is a note for people. The computer skips it.
- Action cards from term 1 are reused: `CLAP`, `TAP DESK`, `STAND UP`, and `REPEAT`, `IF ... THEN ... ELSE`.

Materials

Whole class unless noted.

- Event cards, large (A5 or bigger), each with a colour **and** a shape: green circle, red triangle, blue square (lesson 1).
- A small hand bell or chime, and a "bell" picture card shown at the same time (lesson 1; later lessons use a BELL event card).
- Program cards for each lesson (written on index cards or the board; the text is in each lesson).
- One "score" box per group of four (a shoebox, tray or large envelope with a label), plus two more boxes per group labelled "red" and "blue" for lesson 4.
- One set of number cards 0 to 20 per group (large print, with raised dots or a tactile mark if a child needs it), and a second set per group for lesson 4, because the "red" and "blue" boxes can hold the same number at the same time.
- Hoop-toss event cards per group: 6 "IN HOOP", 4 "GOLD", 6 "MISS", 1 "GAME STARTS", 1 "BELL" (lessons 3 and 5).
- Quiz event cards per group: 1 "QUIZ STARTS", 4 "RED BUTTON", 4 "BLUE BUTTON", 1 "BELL" (lesson 4).
- Paper and pencils for traces (all lessons); blank index cards for new event cards, program parts and notes (stretch in lessons 1 and 3; lessons 4 and 5).
- Name cards for each group (lesson 5).
- A class trace table drawn on the board (columns: event, step, score).

Access and inclusion

- Every event is shown in at least two ways (colour and shape; bell sound and bell picture), so children who cannot hear the bell or cannot tell the colours apart still get the event.
- Any action can be done the child's own way: a wave instead of a clap, raising a hand instead of standing up, tapping a switch or nodding. The own way counts.
- Number cards and event cards can have tactile marks; a partner can read cards aloud. A child can answer by pointing to a number card instead of saying the number.
- Use a quiet chime, not a loud bell, and allow ear defenders.

Assessment

One-to-one or in a small group, about 3 minutes per child, using cards. Record yes / with help / not yet. Covers all five outcomes.

1. Show the lesson 1 scripts. "I show the red triangle, then the green circle. What happens?" (Tap desk 2 times, then clap 1 time.)
2. "SET score TO 4, CHANGE score BY 3, SAY score. What is said?" (7)
3. "SET score TO 9, SET score TO 1, SAY score?" (1, because SET replaces.)
4. Show the hoop-toss program without the start script, after a game that ended on 5. "A new game: IN HOOP, then the bell. What is said, and what should it be?" (6; it should be 1. Add `SET score TO 0` when the game starts.)
5. Show the four quiz parts. "Which part would you look at if the blue score never changes?" (The blue button part.)
6. Show a program with a `NOTE:` credit line. "What does the computer do with this line? Why is it there?" (Skips it; it tells people whose program it was based on.)

Sources

Quoted exactly; the unit is not a full alignment.

- CSTA K-12 Computer Science Standards, Revised 2017, Level 1B (by the end of grade 5), from the progression chart (https://csteachers.org/wp-content/uploads/2025/06/2017-csta-k-12-standards-progression-chart.pdf):
- 1B-AP-09 "Create programs that use variables to store and modify data."
- 1B-AP-10 "Create programs that include sequences, events, loops, and conditionals."
- 1B-AP-11 "Decompose (break down) problems into smaller, manageable subproblems to facilitate the program development process."
- 1B-AP-12 "Modify, remix, or incorporate portions of an existing program into one's own work, to develop something new or add more advanced features."
- 1B-AP-14 "Observe intellectual property rights and give appropriate attribution when creating or remixing programs."
- 1B-AP-15 "Test and debug (identify and fix errors) a program or algorithm to ensure it runs as intended."
- 1B-AP-17 "Describe choices made during program development using code comments, presentations, and demonstrations."
- Wikipedia, "Event-driven programming" (lead): "the flow of the program is determined by external events."
- Wikipedia, "Variable (computer science)" (lead): "in simpler terms, a variable is a named container for a particular set of bits or type of data".

- Wikipedia, "Decomposition (computer science)" (lead): "the process of identifying and organising a complex system into smaller components".
- Wikipedia, "Attribution (copyright)" (lead): "acknowledgment as credit to the copyright holder or author of a work."
- Wikipedia, "Scratch (programming language)" (lead): "a high-level, block-based visual programming language and website aimed primarily at children".

Known gaps and cautions

- Three other agents text-checked this unit on 2026-11-12: Gemini 3.8 Flash (at bbbdbc6) and Gemini 4 Argon (at 9ea2f9c) reported no errors; Gemini 4 Carbon (at 9ea2f9c) reported three small errors (the computing progression link, paper missing from three lessons' materials, and no cap on new point values in lesson 5), all fixed in 0983c5f and fd1fce1 (diff re-checked by Gemini 4 Argon). Those are text checks by AI agents, not teacher reviews or classroom tests.
- The unit is unplugged. Children do not build programs on a device, so 1B-AP-09 and 1B-AP-10 ("Create programs") are met only on paper.
- The card words (`WHEN`, `SET`, `CHANGE ... BY`, `NOTE:`) are made up for this unit to sit next to the term 1 cards. Real tools use different words and blocks.
- `CHANGE ... BY` only adds. Taking away, negative numbers and variables holding words are left for later grades.
- 1B-AP-08 (comparing algorithms), 1B-AP-13 (planning with others' perspectives) and 1B-AP-16 (team roles) are not covered on purpose, apart from the roles children take while acting as the computer.
- Credit in lesson 5 is classroom credit between groups. It does not teach copyright law or licences.
- The Little Hill fair and its games are made up by the author.
- Scores are only for the pretend game, drawn from a card pile; no child's own throwing or quiz answers are scored or compared.

Lesson 1: When something happens

Unit overview - Next: Lesson 2

About 45 minutes.

Goal

Children learn that an **event** is something that happens, such as a card being shown or a bell ringing, and that an **event script** waits and runs its steps only when its event happens. They trace which script runs for a list of events.

Materials

- Event cards: green circle, red triangle, blue square (large, colour and shape).
- A small hand bell or quiet chime, and a "bell" picture card.
- Three program cards (or the board) with the scripts below.
- The term 1 program card from Argon's unit, or any short straight list of steps, for the opening.
- Paper and pencils for the closing check; blank index cards (stretch: new event cards).

Steps

✎ Opening (5 min)

1. Hold up a term 1 style program (a straight list of steps). Ask: "When does the robot do step 1? Step 2?" (Straight away, then one after the other, top to bottom.)
2. Ask: "Think of a light that comes on when someone walks past, or a lift that comes when you press its button. Does it do things straight away, one after another?" (No. It waits until something happens.)
3. Say: "Something that happens, which a program can notice, is called an **event**. Today the class is a program that waits for events."

✎ Main activity (30 min)

Part 1: three event scripts (10 min). Put up three scripts:

```
WHEN the green circle card is shown:
    CLAP 1 TIME

WHEN the red triangle card is shown:
    TAP DESK 2 TIMES

WHEN the bell rings:
    STAND UP
```

Read them together. Say: "Each script waits. Its steps run only when its event happens, and every time it happens." Practise: show the green circle (everyone claps once), the red

triangle (two taps), then ring the bell while holding up the bell picture (everyone stands; then sit down again).

Part 2: the teacher chooses the order (10 min). Run these event lists. Before each one, children predict on their fingers or in pairs; then run it.

- List A: green, green, red. (Clap, clap, tap tap. Nobody stands.)
- List B: bell, red, green. (Stand, tap tap, clap.)
- List C: blue square. (Nothing happens: no script waits for the blue square.)

Point out: the scripts did not change, but the actions came in a different order each time, because **the events chose the order**.

Part 3: a fourth script (10 min). Add, with the class:

```
WHEN the blue square card is shown:
    REPEAT 3 TIMES:
        TAP DESK
```

Run List C again (three taps). Then let volunteers be the "event giver" and show cards in an order they choose, while the class runs the scripts. The event giver must show one card at a time and wait until the actions finish.

✓ Discussion questions (with answers) (5 min)

- "What made the class clap?" (The green circle being shown: the event.)
- "In term 1, the program ran top to bottom. What is different today?" (The scripts waited; the events decided which script ran and when.)
- "Why did nothing happen for the blue square at first?" (No script was waiting for it.)
- "Name an everyday machine that waits for an event." (Examples: a door that opens when someone comes near, a toaster that pops, a game that moves when you press a button. Any sensible answer counts.)

✓ Closing check (with answers) (5 min)

With the four scripts still showing, each child writes or points to an answer.

1. "I show red, then green. What happens?" (Tap tap, then clap.)
2. "I ring the bell twice. How many times do you stand up?" (Twice: the script runs every time; sit down in between.)
3. "I show a yellow card. What happens?" (Nothing; no script for it.)

Vocabulary

event, event script, wait, run

Differentiation

- **Support:** Use only two scripts (green and bell) at first. Give the child a copy of the scripts to touch and point to.

- **Stretch:** Children write a new script for a new event card of their own design, using only term 1 actions, and run it with a partner.
- **Tactile:** Cut-out shapes (circle, triangle, square) glued on the event cards, so they can be felt.

⚠ Safety

- Clear space around desks before standing. Standing is optional: raising a hand counts.
- Use a quiet chime and allow ear defenders.

Teacher notes and misconceptions

- Some children think the scripts run in the order they are written. Run List B to show they do not.
- Some think a script runs only once. Ring the bell twice to show it runs every time.
- Keep the gaps between events long enough for everyone to finish.

Lesson 2: A box with a name

Unit overview - Previous: Lesson 1 - Next: Lesson 3

About 45 minutes.

Goal

Children learn that a **variable** is like a box with a name that holds one value at a time. They run and trace programs that use `SET`, `CHANGE ... BY` and `SAY`, and say the final value.

Materials

- One "score" box per group of four, with a label.
- One set of number cards 0 to 20 per group.
- Program cards A to D (below), on cards or the board.
- A class trace table on the board (columns: step, score).
- Paper and pencils for traces and for writing a program in Part 3.

Steps

✎ Opening (5 min)

1. Show the empty "score" box. Ask: "In a game, how does the game remember your score?" (Take ideas.)
2. Say: "Programs keep things to remember in a **variable**. Think of it as a box with a name on it. The box holds **one** number at a time."
3. Put the 0 card in the box. "The value of score is now 0."

✎ Main activity (30 min)

Part 1: three cards (8 min). Teach the three cards:

- `SET score TO 0`: take out whatever card is in the box and put in the 0 card.
- `CHANGE score BY 2`: look at the number in the box, add 2, swap the card for the new number.
- `SAY score`: read the number in the box aloud (or hold it up).

Model Program A with the class, filling in the trace table:

```
SET score TO 0
CHANGE score BY 1
CHANGE score BY 1
SAY score
```

Step	score
SET score TO 0	0
CHANGE score BY 1	1

Step	score
CHANGE score BY 1	2
SAY score	2 (said aloud)

Part 2: groups run programs (14 min). In groups of four, roles rotate for each program: one reads the cards, one is the box keeper (swaps number cards), one writes the trace, one checks. Each group runs:

- Program B: `SET score TO 5`, `CHANGE score BY 2`, `CHANGE score BY 3`, `SAY score`. (5, 7, 10; says 10.)
- Program C: `SET score TO 3`, `SET score TO 7`, `SAY score`. (3, then 7; says 7. SET replaces the old value; it does not add.)

Then Program D, with a term 1 loop:

```
SET score TO 0
REPEAT 4 TIMES:
    CHANGE score BY 2
SAY score
```

(0, 2, 4, 6, 8; says 8.)

Part 3: one box, one value (8 min). Ask groups: "After Program C, where did the 3 go?" (It was taken out; the box only holds the newest value.) Then each group writes its own three-line program using SET, CHANGE and SAY, with a final value no bigger than 20, and swaps it with another group to run. The writers check the answer.

✓ Discussion questions (with answers) (5 min)

- "How is SET different from CHANGE?" (SET puts in a new value and throws away the old one; CHANGE adds to the value already there.)
- "Why does the box have a name?" (So the program can say which box to use, especially when there is more than one.)
- "Can the box hold two numbers at once?" (No. One value at a time.)

✓ Closing check (with answers) (5 min)

Children write the number that is said.

1. `SET score TO 4`, `CHANGE score BY 3`, `SAY score`. (7)
2. `SET score TO 2`, `REPEAT 3 TIMES: CHANGE score BY 5`, `SAY score`. (17)
3. `SET score TO 9`, `SET score TO 1`, `CHANGE score BY 1`, `SAY score`. (2)

Vocabulary

variable, value, set, change, trace

Differentiation

- **Support:** Use the box and number cards for every step; keep values under 10. A partner reads the cards aloud.
- **Stretch:** Write a program whose final value is 20 using exactly one SET and one REPEAT.
- **Tactile:** Number cards with raised dots or tactile numerals; the child checks the box by touch.

⚠ Safety

- No special risks. Keep boxes and cards on desks.

Teacher notes and misconceptions

- The most common mistake is treating SET like CHANGE (answering 10 for Program C). Act it out: take the 3 card out of the box, put the 7 in.
- Some children write every value in the box at once, like a list. Keep only one card in the box.
- A variable in computing is not quite the same as a letter for an unknown number in maths; here it is simply a named place for a value that can change.

Lesson 3: Keeping score at the Little Hill fair

Unit overview - Previous: Lesson 2 - Next: Lesson 4

About 45 minutes.

Goal

Children run a game program that uses events **and** a score variable. They trace it for a list of events, and find and fix a bug: the score was never set back to 0 when a new game started.

Materials

- One "score" box and one set of number cards 0 to 20 per group of four.
- Hoop-toss event cards per group: 6 "IN HOOP", 4 "GOLD", 6 "MISS", 1 "GAME STARTS", 1 "BELL" (game ends).
- The hoop-toss program on the board or on cards (below), plus a copy with the first script covered or left out, for Part 3.
- A class trace table on the board (columns: event, step, score).
- Paper and pencils for traces; blank index cards (stretch: BULLSEYE).

Steps

✎ Opening (5 min)

1. Say: "The made-up town of Little Hill is having a school fair. One stall has a hoop-toss game. A beanbag can land **in the hoop** (1 point), in the **gold** circle in the middle (3 points), or **miss** (0 points). The stall wants a program to keep the score."
2. Say: "Nobody throws anything today. We draw event cards from a pile instead, so the game is the same for everyone."

✎ Main activity (30 min)

Part 1: the program (8 min). Put up:

```
WHEN GAME STARTS:
    SET score TO 0

WHEN IN HOOP:
    CHANGE score BY 1

WHEN GOLD:
    CHANGE score BY 3

WHEN BELL:
    SAY score
    IF score is greater than 4 THEN:
        CLAP 2 TIMES
```

```
    ELSE:
        CLAP 1 TIME
```

Ask: "Which event has no script?" (MISS. Nothing happens, so the score stays the same.) Trace List 1 together on the board:

Event	Step	score
GAME STARTS	SET score TO 0	0
IN HOOP	CHANGE score BY 1	1
MISS	(no script)	1
GOLD	CHANGE score BY 3	4
IN HOOP	CHANGE score BY 1	5
BELL	SAY score; 5 is greater than 4, so CLAP 2 TIMES	5

Part 2: groups run fixed lists, then a random game (12 min). Roles rotate: event giver, box keeper, tracer, checker. Groups run:

- List 2: GAME STARTS, GOLD, GOLD, MISS, IN HOOP, BELL. (0, 3, 6, 6, 7; says 7; clap 2 times.)
- List 3: GAME STARTS, MISS, IN HOOP, GOLD, BELL. (0, 0, 1, 4; says 4; 4 is not greater than 4, so clap 1 time.)

Then a random game: the event giver shows GAME STARTS, shuffles the IN HOOP, GOLD and MISS cards face down, turns over five of them one at a time, then shows BELL. The tracer writes each step. (The biggest possible score is 13: four GOLD and one IN HOOP. Any total is fine; the check is that the trace matches the cards.)

Part 3: a bug (10 min). Show the program again with the `WHEN GAME STARTS` script missing. Say: "The stall used this version. The first game was List 1 and ended on 5. Then a new child played: IN HOOP, then BELL." Groups run it with the box still holding 5. (5 + 1 = 6; it says 6.) Ask: "What should it have said?" (1.) "Which line is missing?" (`SET score TO 0`, under `WHEN GAME STARTS`.) Put the script back and run the new game again: GAME STARTS, IN HOOP, BELL. (0, 1; says 1; clap 1 time.)

Say: "We tested, found the step that went wrong, fixed it and tested again: just like term 1's bug hunts."

✓ Discussion questions (with answers) (5 min)

- "Why does MISS not need a script?" (A miss adds nothing, so the score should stay the same, and with no script nothing happens.)
- "Why does the game need `SET score TO 0` at the start?" (Otherwise the new game starts with the old game's score.)
- "Which script reads the box but does not change it?" (The BELL script.)

✓ Closing check (with answers) (5 min)

1. "GAME STARTS, GOLD, MISS, IN HOOP, IN HOOP, BELL. What is said, and how many claps?" (3, 3, 4, 5; says 5; clap 2 times.)

2. "With the start script missing, the box holds 5. Then GOLD, BELL. What is said, and what should it be?" (8; it should be 3.)
3. "Which line fixes it?" (`SET score TO 0` when the game starts.)

Vocabulary

event, variable, score, bug, test, reset (set back to 0)

Differentiation

- **Support:** Use only IN HOOP and MISS cards (1 point or nothing) and leave out the IF part of the BELL script.
- **Stretch:** Add a new card, "BULLSEYE", and write its script so it is worth 5. Run a list with it.
- **Tactile:** Event cards with a feelable mark (one dot for IN HOOP, three dots for GOLD, a cross for MISS).

⚠ Safety

- Nothing is thrown. If the school wants a real hoop toss later, use soft beanbags, keep everyone out of the throwing area, and do not score real children's throws against each other.

Teacher notes and misconceptions

- Some children add 1 for MISS. Point to the program: there is no MISS script, so nothing happens.
- Some children think "greater than 4" includes 4. Run List 3 to show that 4 gives one clap.
- The score belongs to the pretend game, not to any child.

Lesson 4: Splitting a big job into parts

Unit overview - Previous: Lesson 3 - Next: Lesson 5

About 45 minutes.

Goal

Children split a bigger program into smaller parts, each write one part, join the parts and test them with a list of events. They test one part at a time to find which part has a bug.

Materials

- Per group of four: two boxes labelled "red" and "blue", and **two** sets of number cards 0 to 20 (both boxes can hold the same number, such as 0, at the same time).
- Quiz event cards per group: 1 "QUIZ STARTS", 4 "RED BUTTON", 4 "BLUE BUTTON", 1 "BELL" (quiz ends).
- Blank index cards for the parts, and paper and pencils for the traces.
- The job description (below) on the board.
- A class trace table on the board (columns: event, step, red, blue).

Steps

✎ Opening (5 min)

1. Say: "The Little Hill library is having a pretend quiz with a red team and a blue team. Each team has a button. The library wants a program that keeps both scores."
2. Put up the job description:
3. At the start, both scores are 0.
4. When the red button is pressed, red gets 1 more.
5. When the blue button is pressed, blue gets 1 more.
6. At the end (the bell), both scores are said.
7. Say: "That is a big job. Programmers split a big job into smaller parts, so different people can build and test each part. This is called **decomposing** the problem."

✎ Main activity (30 min)

Part 1: name the parts (5 min). With the class, turn each line of the job description into a part: Start, Red button, Blue button, End. Each child in a group of four takes one part.

Part 2: write the parts (8 min). Each child writes their part on an index card, using the card words from lessons 1 to 3. One correct set:

```
Part 1 (Start)
WHEN QUIZ STARTS:
    SET red TO 0
    SET blue TO 0

Part 2 (Red button)
```

```
WHEN RED BUTTON:
    CHANGE red BY 1

Part 3 (Blue button)
WHEN BLUE BUTTON:
    CHANGE blue BY 1

Part 4 (End)
WHEN BELL:
    SAY red
    SAY blue
```

Other wordings that do the same job count.

Part 3: test each part, then join (9 min). First each child tests their own part with just its event (for example, show BLUE BUTTON once and check that only the blue box changed). Then the group lays the four cards side by side and runs the test list:

QUIZ STARTS, RED BUTTON, BLUE BUTTON, RED BUTTON, RED BUTTON, BELL.

Event	Step	red	blue
QUIZ STARTS	SET red TO 0, SET blue TO 0	0	0
RED BUTTON	CHANGE red BY 1	1	0
BLUE BUTTON	CHANGE blue BY 1	1	1
RED BUTTON	CHANGE red BY 1	2	1
RED BUTTON	CHANGE red BY 1	3	1
BELL	SAY red, SAY blue	3	1

(Says red 3, blue 1.)

Part 4: a bug in one part (8 min). Show this version of Part 3:

```
Part 3 (Blue button)
WHEN BLUE BUTTON:
    CHANGE red BY 1
```

Run the same test list. (Red 4, blue 0, instead of red 3, blue 1.) Ask: "Which part would you test first?" (The blue button part, because the blue score never changed.) Show BLUE BUTTON alone: the red box changes, so the bug is in Part 3. Fix it to `CHANGE blue BY 1` and run the list again. (Red 3, blue 1.)

Then groups shuffle the RED and BLUE cards face down and run a random quiz: QUIZ STARTS, five cards, BELL. The check is that the trace matches the cards.

✓ Discussion questions (with answers) (5 min)

- "Why split the program into parts?" (Each part is small enough to write and test; different people can work on different parts; a bug is easier to find.)
- "How did testing one part at a time help?" (It showed which part was wrong, so only that part needed fixing.)

- "Why does this program need two variables?" (It has to remember two different scores at the same time.)

✓ Closing check (with answers) (5 min)

1. "QUIZ STARTS, BLUE BUTTON, BLUE BUTTON, RED BUTTON, BELL. What is said?" (Red 1, blue 2.)
2. "Part 1 is missing. From an earlier quiz, the boxes hold red 3 and blue 1. Then RED BUTTON, BELL. What is said, and what should it be?" (Red 4, blue 1; it should be red 1, blue 0.)
3. "Name the four parts of the quiz program." (Start, Red button, Blue button, End.)

Vocabulary

decompose, part, test, join, two variables

Differentiation

- **Support:** The child writes the Red button part with the Blue button part as a model next to it. Pair a child with a partner for the trace.
- **Stretch:** Add a fifth part: "When the bell rings, IF red is greater than blue THEN CLAP 2 TIMES ELSE CLAP 1 TIME." Test it with a list where the scores are equal. (Equal scores give one clap, because red is not greater than blue.)
- **Tactile:** Boxes with different textures (for example, felt on the red box) as well as labels.

⚠ Safety

- No special risks.

Teacher notes and misconceptions

- Some children think each part needs its own copy of the start steps. Show that one Start part sets both scores.
- When joining parts, the order of the cards on the desk does not matter: each part waits for its own event.
- The quiz is pretend: no child answers questions, and no child's score is compared with another's.

Lesson 5: Changing someone's program and saying so

Unit overview - Previous: Lesson 4

About 45 minutes.

Goal

Children change one thing in another group's program (a **remix**), add a `NOTE:` line that credits the group whose program they used, test the new version, and say in one sentence what they changed and why.

Materials

- Per group of four: one "score" box, one set of number cards 0 to 20, and the hoop-toss event cards (6 "IN HOOP", 4 "GOLD", 6 "MISS", 1 "GAME STARTS", 1 "BELL").
- Per group: a copy of the lesson 3 hoop-toss program on cards, with a name card saying which group made it (each group makes or signs one copy at the start of the lesson, then copies are swapped).
- Paper and pencils for the traces; blank index cards for the NOTE lines and BULLSEYE.
- The example remix (below) on the board.

Steps

✎ Opening (5 min)

1. Each group writes its group name on a name card next to its copy of the hoop-toss program. Copies are passed to the next group.
2. Say: "Programmers often start from someone else's program and change it. That is called a **remix**. When we use someone's work, we say whose it was. That is called **giving credit**."
3. Teach the new card: `NOTE: ...` is a note for people. The computer skips it, like a sticky note on a recipe.

✎ Main activity (30 min)

Part 1: an example remix (7 min). Put up:

```
NOTE: Based on the hoop-toss program by Group B.
NOTE: We changed GOLD to 2 points so the scores are closer.

WHEN GAME STARTS:
    SET score TO 0

WHEN IN HOOP:
    CHANGE score BY 1
```

```
WHEN GOLD:
    CHANGE score BY 2

WHEN BELL:
    SAY score
    IF score is greater than 4 THEN:
        CLAP 2 TIMES
    ELSE:
        CLAP 1 TIME
```

Run lesson 3's List 1 with it: GAME STARTS, IN HOOP, MISS, GOLD, IN HOOP, BELL. (0, 1, 1, 3, 4; says 4; 4 is not greater than 4, so clap 1 time. The original said 5 with two claps.) Ask: "Did the NOTE lines change the score?" (No. The computer skips them.)

Part 2: groups remix (12 min). Each group changes **one** thing in the program it was given. Choices:

- change how many points GOLD or IN HOOP is worth (4 or fewer, so a five-card game can never go past 20, the biggest number card);
- add a new event card, BULLSEYE, written on a blank card, with a script that adds 5;
- change the number in the IF line (for example, greater than 6).

They add two `NOTE:` lines at the top: whose program it was based on, and what they changed.

Part 3: test (6 min). Each group runs lesson 3's List 1 with its remix and writes the trace, then runs one random game. (If they added BULLSEYE, they shuffle it in with the other cards.) If the remix does not do what they meant, they find and fix the step, then test again.

Part 4: show and say (5 min). Each group says one sentence: "We changed ... because ...", and hands the remix back to the group that made the original, so they can see it.

✓ Discussion questions (with answers) (5 min)

- "Why did we write whose program it was?" (It is fair: they did the work we started from. It also helps people find the original.)
- "Is a remix the same as taking someone's program away?" (No. The original group still has theirs; we made a new version and said so.)
- "Why write what you changed?" (So other people, and you later, know what is different and why.)

✓ Closing check (with answers) (5 min)

Show the example remix from Part 1.

1. "GAME STARTS, GOLD, GOLD, IN HOOP, BELL. What is said, and how many claps?" (2, 4, 5; says 5; clap 2 times.)
2. "What does the computer do with the NOTE lines?" (Skips them.)
3. "Whose program was this remix based on?" (Group B's.)

Vocabulary

remix, credit, note (comment), original

Differentiation

- **Support:** The group changes only the number in one CHANGE card. The teacher writes the NOTE lines with them.
- **Stretch:** Make two changes, write a NOTE for each, and work out a list of events where the remix and the original say different scores.
- **Tactile:** NOTE cards in a different texture or with a cut corner, so they can be told apart from cards the computer runs.

⚠ Safety

- Nothing is thrown. Only change programs on cards, not anyone's belongings.

Teacher notes and misconceptions

- Some children think a NOTE line is a step. Run the program with the NOTE lines covered: the result is the same.
- Credit here is between groups in the class. Real programs and pictures online also have rules about reuse (licences); that is for later grades.
- If the school has a block-based tool, groups can rebuild their remix on it, with the credit written in the tool's comment feature or project notes. Check the tool against your school's rules first.

Grade 3 Computing: Data and Networks (5 lessons, about 3 weeks)

> **Unreviewed AI draft.** Written by Claude Opus 5.5 (AI Village), 13 November 2026. No educator has reviewed or classroom-tested this unit. It is not aligned to any official standard. Check every activity against your school's rules.
>
> Text-checked by four other agents (Gemini 3.5 Flash, Gemini 3.8 Flash, Gemini 4 Argon and Gemini 4 Carbon); see Known gaps.

Back to the grade 3 year plan - Term 3. Builds on grade 3 computing term 2, Events and variables, and grade 3 computing term 1, Gemini 4 Argon's Inputs, Outputs, Loops, and Debugging.

This unit is "unplugged": children act as the network, using cards, and make charts on paper. No device is needed. All data in the unit is about the made-up town of Little Hill. Children never collect, share or chart facts about themselves or their families.

Big idea

A long message sent across a network is split into small numbered pieces called **packets**. The packets can travel different ways, through several devices, and arrive in a muddle or not at all; the numbers let the receiver put them back in order and ask again for any piece that is missing. **Data** is information we collect, such as counts. Sorting it into a table or a bar chart helps us see a pattern, back up a claim and make a prediction, but a pattern on its own does not prove what caused it. Some information, such as a password or where someone lives, should stay private.

Outcomes

By the end of the unit, most children can:

1. Act out sending a message as numbered packets through "router" children, and put packets that arrived out of order back in order (lesson 1).
2. Use the numbers on packets to say which packet is missing, ask for it again, and spot a packet that arrived twice (lesson 2).
3. Sort and tally made-up data, show it in a table and a bar chart, and make a claim backed by the chart (lesson 3).
4. Use a chart to make a prediction, and say whether something that goes with a pattern is likely to be the cause or not (lesson 4).
5. Sort made-up pieces of information into "fine to share", "keep private" and "ask a trusted adult", and say what to do if a message asks for a password (lesson 5).

Prior knowledge

From terms 1 and 2: following exact steps on cards, and finding a bug by testing one step at a time. From maths: counting and adding within 100, tally marks, and reading a scale on a bar chart going up in ones (grade 3 science weather data also uses bar charts).

Lessons

Lesson	Title	Main activity
1	A message in pieces	Four tables each send a four-card message as numbered packets through "router" children and put them back in order
2	Lost and doubled packets	A router "drops" a packet; the receiver works out which number is missing, sends a "Lost" card, and deals with a packet that arrives twice
3	What Little Hill library lends	Groups sort 30 made-up loan slips, tally them, make a table and a bar chart, and write a claim with evidence
4	Rain, visitors and a new sign	Groups sort ten made-up day cards, predict day 11, and decide whether rain or a new sign goes with more visitors
5	Keeping information private	Groups sort made-up information cards, discuss a trick message asking for a password, and talk about long passphrases

Each lesson is about 45 minutes: opening 5, main activity 30, discussion 5, closing check 5.

Materials

Whole class unless noted.

- Large index cards for packets (lessons 1 and 2): one demo message of 3 cards, and for each of the four tables one message of 4 cards **plus a copy set** of the same 4 cards (lesson 2), so 35 packet cards in all, plus six blank index cards for the lesson 2 stretch. The card text is in lesson 1. Each card has its number in large print with a raised dot pattern or braille if a child needs it.
- Four table signs, each with a shape and its name: circle, triangle, square, diamond. Each shape is also a raised foam sticker, and every packet card carries the raised sticker of the table it is going to (lessons 1 and 2).
- Four chairs for "routers", set out in the middle of the room with wide clear gaps, and floor tape stuck down flat to mark the lines between them (lessons 1 and 2).
- A small number line 1 to 4 for each of the four tables (lessons 1 and 2).
- A tray for packets the teacher "takes" (lesson 2).
- "Lost" cards and "Got 4 of 4" cards, 4 of each, with a blank for the number (lesson 2).
- Loan slips for lesson 3: one set of 30 per group of four (9 animals, 7 fun facts, 6 plants, 5 boats, 3 maps), each with a picture and a word, and a raised mark for each kind (for example, a raised dot pattern) if a child needs it.
- Day cards for lesson 4: one set of 10 per group of four, plus a day 11 card kept by the teacher. Rain cards have a raised mark; "no rain" cards are smooth.
- Information cards for lesson 5: one set of 10 per group of four (text in lesson 5), and three sorting mats per group: "fine to share", "keep private", "ask a trusted adult" (with a picture and a raised mark on each).
- A large printed copy of the trick message card for lesson 5, and about 12 word cards with pictures per group for pretend passphrases.
- Squared paper (1 cm or larger), rulers and pencils (lessons 3 and 4); paper and pencils for closing checks (all lessons).
- A class chart drawn on the board for lessons 3 and 4.
- For tactile access: a peg board where each peg is one count (lessons 3 and 4), and a lap tray for a child in a wheelchair (all lessons, if needed).

Access and inclusion

- Packet cards are passed hand to hand at table height. Routers sit, so a child in a wheelchair can be a router, a sender or a receiver in the same way. Nobody runs or throws cards.
- Every table, packet, slip, day and sorting mat is marked in at least two ways (word or number plus picture, shape or raised mark). A partner or adult reads any card aloud.
- Charts can be built with pegs or blocks instead of drawn. A child can answer by pointing, holding up a card or using the class communication method; the child's own way counts.
- Children who find a busy room hard can be the "checker" at a receiving table instead of moving around.
- Lesson 5 never asks children what their own passwords, addresses or birthdays are. Questions are only about made-up people.

Assessment

One-to-one or in a small group, about 3 minutes per child, using cards. Record yes / with help / not yet. Covers all five outcomes.

1. Give four packet cards out of order ("2 of 4", "4 of 4", "1 of 4", "3 of 4"). "Put the message back together." (1, 2, 3, 4 in order.)
2. Show packets "1 of 4", "2 of 4", "4 of 4", "2 of 4". "Is anything missing? Is anything there twice?" (3 is missing; 2 came twice.)
3. Show the lesson 3 bar chart. "Which kind of book was borrowed most? How do you know?" (Animals; its bar is the tallest, 9.)
4. Show the lesson 4 day cards sorted. "Day 12 will have rain. About how many people might come?" (A number from about 17 to 21; it is a prediction, so it might be wrong.)
5. "The new sign went up on day 6. Did more people come after it?" (No; rain days still had 17 to 21 and days with no rain still had 8 to 12, so the sign is probably not the cause.)
6. Show a made-up card: "Kit's password." "Fine to share, keep private, or ask a trusted adult?" (Keep private; tell only a trusted adult who looks after the account.)

Sources

Quoted exactly; the unit is not a full alignment.

- CSTA K-12 Computer Science Standards, Revised 2017, Level 1B (by the end of grade 5), from the progression chart (https://csteachers.org/wp-content/uploads/2025/06/2017-csta-k-12-standards-progression-chart.pdf):
- 1B-NI-04 "Model how information is broken down into smaller pieces, transmitted as packets through multiple devices over networks and the Internet, and reassembled at the destination."
- 1B-NI-05 "Discuss real-world cybersecurity problems and how personal information can be protected."
- 1B-DA-06 "Organize and present collected data visually to highlight relationships and support a claim."
- 1B-DA-07 "Use data to highlight or propose cause-and-effect relationships, predict outcomes, or communicate an idea." (On the chart the line breaks after "cause-"; the word is "cause-and-effect".)
- Wikipedia, "Network packet" (lead): "a network packet is a formatted unit of data carried by a packet-switched network."

- Wikipedia, "Packet loss" (lead): "Packet loss occurs when one or more packets of data travelling across a computer network fail to reach their destination."
- Wikipedia, "Bar chart" (lead): "a chart or graph that presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent."
- Wikipedia, "Password" (lead): "A memorized secret consisting of a sequence of words or other text separated by spaces is sometimes called a passphrase."

Known gaps and cautions

- Four other agents (Gemini 3.5 Flash, Gemini 3.8 Flash, Gemini 4 Argon and Gemini 4 Carbon, all at 436f646) text-checked this unit on 2026-11-13. The first three reported no errors; Gemini 4 Carbon reported a wording slip and two materials gaps, fixed in aa41f6c; Gemini 4 Argon and Gemini 4 Carbon re-checked the fix. Those are text checks by AI agents, not a teacher review, classroom test or physical safety assessment.
- The unit is unplugged. The network is modelled by children passing cards, which leaves out a great deal: addresses are shapes, not numbers; there are no protocols, no checking codes and no encryption. These come in later grades (for example CSTA 2-NI-04 on protocols).
- Real networks do not always resend lost packets; some kinds of data, such as live video, carry on without them. Lesson 2 shows only the "ask again" idea.
- 1B-DA-06 asks children to organise "collected data". Here the data is made up and handed out on cards, so children organise and chart it but do not collect it. This is on purpose, so that no child's own information is gathered.
- The library loans, visitor numbers, the new sign and Kit's hat are made up by the author; they are an example, not from a source.
- Lesson 4 treats "cause" carefully: the data can only suggest that rain goes with more visitors. It does not prove why.
- Lesson 5 is a first talk about privacy and trick messages. It does not replace the school's online-safety teaching or its rules for reporting worries, and it does not teach children to make or store real passwords.
- Card words for children to read use only letters and patterns taught by this point in the literacy units; some words (such as "visitors", "address" and "password") are read aloud by the teacher first.

Lesson 1: A message in pieces

Unit overview - Next: Lesson 2

About 45 minutes.

Goal

Children learn that a message sent across a network is split into small numbered pieces called **packets**, that the packets pass through several devices (here, "router" children) and may arrive in a different order, and that the receiver uses the numbers to put the message back together.

Materials

- Demo packet cards (3), each with the raised sticker of the square table: "1 of 3: Let us", "2 of 3: sit on", "3 of 3: the mat."
- Four table signs with shapes (circle, triangle, square, diamond), each shape also a raised foam sticker. The teacher reads the shape names aloud first.
- One message of 4 packet cards for each table (text below), each card with the raised sticker of the table it is going to.
- Four chairs for routers in the middle of the room, in a square, with wide clear gaps.
- Floor tape to mark the lines between router chairs (stuck down flat).
- A small number line 1 to 4 for each receiving table (support).
- A lap tray for a child in a wheelchair, if needed.
- Paper and pencils for the closing check.

Setting up the room

Put the four tables near the four corners and the four router chairs in a square in the middle. Each table has one router chair nearest to it:

```
circle table                    triangle table
       router A ------- router B
          |                |
       router D ------- router C
diamond table                   square table
```

Routers pass cards only along the lines: A to B, B to C, C to D, D to A, and each router to its own table. So a card from the circle table to the square table can go A, B, C or A, D, C.

The four messages

Each card shows its number, its words and the shape of the table it is going to. The words use only taught letters and patterns.

From	To	1 of 4	2 of 4	3 of 4	4 of 4
circle	square	Hen can	come to	the pond	at nine.
square	circle	Goat has	a red	boat at	the pond.
triangle	diamond	Pip has	a big	green kite	at home.

From	To	1 of 4	2 of 4	3 of 4	4 of 4
diamond	triangle	Meet at	the big	oak tree	at ten.

Steps

✎ Opening (5 min)

1. Ask: "If you send a long message to someone far away on a computer or a tablet, does it go in one big lump?" Take ideas.
2. Say: "It is split into small pieces called **packets**. Each packet has a number, and a note saying where it is going. The pieces pass through several machines on the way, called **routers**, and are put back together at the end."
3. Show the three demo cards and read them together.

✎ Main activity (30 min)

Part 1: the demo (5 min). The teacher is the sender at the circle table. Four children sit on the router chairs. The teacher gives "3 of 3" to router A first, then "1 of 3", then "2 of 3". Routers pass each card, one at a time, along a line towards the square table. The square table puts the cards in number order and reads: "Let us sit on the mat."

Ask: "The cards came in a muddle. How did you know the order?" (The numbers.)

Part 2: the router rules (5 min). Put up the rules and read them:

1. A router holds one card at a time.
2. Look at the shape on the card. Pass it along a line, one step nearer to that table.
3. If the next router's hands are full, wait, or pass it the other way round.
4. Pass cards hand to hand. Never throw them. Stay seated.

Senders give their router one card at a time, only when the router's hands are empty, in any order they like.

Part 3: four messages at once (12 min). Give each table its four cards, face down. All four tables send at the same time. Each receiving table has a "checker" who lays the cards in a line as they arrive, then puts them in number order and reads the message aloud.

Note on the board, for each table, the order the cards arrived in (for example "3, 1, 4, 2").

Part 4: swap and send again (8 min). New children become routers. The tables send the same messages again, so each table sees a new arrival order. Compare the two arrival orders for one table. Ask: "Was it the same order both times?" (Usually not; the cards took different ways and waited at different routers.)

✓ Discussion questions (5 min)

- Why does each packet have a number? (So the receiver can put the pieces back in order even if they arrive mixed up.)
- Why does each packet have a shape? (So each router knows where to send it.)

- Why might "2 of 4" arrive after "4 of 4"? (It took a longer way, or waited at a busy router.)
- How do you know the whole message has arrived? (You have all the numbers, 1 to 4, because each card says "of 4".)

✓ Closing check (5 min)

On paper, draw or write the order to put these in: "3 of 4", "1 of 4", "4 of 4", "2 of 4". (1, 2, 3, 4.) Then answer: "What do the routers do?" (Pass each packet one step nearer to where it is going.)

Vocabulary

packet, message, router, network, sender, receiver, order

Differentiation

- **Support:** the receiving table has a number line 1 to 4 to lay cards on. A child can read the numbers only, and an adult reads the words.
- **Stretch:** take one line away (for example A to B). Can every card still get through? (Yes: circle to triangle can go A, D, C, B.) Children say why having more than one way helps.
- **Tactile:** each card has its number in raised dots or braille and the raised shape sticker; the router chairs have the line partners marked by a tape strip on the floor between chairs that a cane or a foot can follow, stuck down flat.

⚠ Safety

Keep gaps between chairs wide enough for a wheelchair, and floor tape flat. Routers stay seated and pass cards by hand; nobody runs or throws.

Teacher notes and misconceptions

- Children may think a message travels like a letter in one envelope. The activity shows it is split up; that is the main idea of the lesson.
- "Router" here means a child who passes cards on. Real routers are machines, and real addresses are numbers, not shapes.
- Some children think the cards must arrive in order or the message is broken. Stress that the numbers fix the order at the end.

Lesson 2: Lost and doubled packets

Unit overview - Previous: Lesson 1 - Next: Lesson 3

About 45 minutes.

Goal

Children learn that packets can get lost on the way, that the numbers on packets show which one is missing, and that the receiver can ask the sender to send it again. They also deal with a packet that arrives twice.

Materials

- The four table signs, router chairs and floor tape from lesson 1.
- The demo packet cards "1 of 3" and "3 of 3" from lesson 1.
- For each table: its four packet cards from lesson 1 **and** a copy set of the same four cards.
- "Lost" cards (4) with a blank for the number: "Lost: ___. Send it."
- "Got 4 of 4" cards (4).
- A small number line 1 to 4 for each receiving table.
- A tray for packets the teacher takes.
- Six blank index cards to make a 6-packet message (stretch).
- A lap tray for a child in a wheelchair, if needed.
- Paper and pencils for the closing check.

Steps

✎ Opening (5 min)

1. Lay out the demo message from lesson 1 as "1 of 3" and "3 of 3" only. Ask: "What is wrong?" (Card 2 is missing.) "How do you know?" (The cards say "of 3", and there is no 2.)
2. Say: "On a real network, a packet sometimes does not arrive. This is called a **lost packet**. Today we find out what the receiver can do."

✎ Main activity (30 min)

Part 1: the copy and the Lost card (8 min). Explain the new sender rule: "Keep your copy set until the receiver says it has everything." Show the "Lost" card and the "Got 4 of 4" card. Read them together.

Run the circle table's message (to the square table) with the whole class watching. The teacher quietly takes one card out of the routers' hands as it passes (a router "dropped" it) and puts it in a tray. The square table lays out what arrived on its number line, sees the gap, writes the number on a "Lost" card and sends it back through the routers to the circle table. The circle table finds that number in its copy set and sends it. When the square table has all four, it sends back "Got 4 of 4", and only then does the circle table put its copy set away.

Part 2: all four tables (12 min). All four tables send at once. During the round, the teacher takes one card from two of the messages (choose which in secret). Receivers use the number line to find the gap, send a "Lost" card, and wait for the copy.

Part 3: a packet that comes twice (10 min). Run one more round. This time the teacher holds a taken card for a while, then quietly gives it back to a router after the receiver has already asked for it again. The receiver now has two cards with the same number. Ask: "What should you do?" (Keep one. The words are the same, so the message does not change.)

✓ Discussion questions (5 min)

- How did the receiver know which packet was lost? (The numbers 1 to 4 had a gap.)
- Why did the sender keep a copy? (So it could send a lost packet again.)
- When can the sender throw the copy away? (When the receiver sends "Got 4 of 4".)
- What would go wrong if packets had no numbers? (You could not tell the order, or which one was missing.)

✓ Closing check (5 min)

On paper: "These arrived: 1 of 4, 4 of 4, 2 of 4, 4 of 4. Which one is missing? Which one came twice?" (3 is missing; 4 came twice.) "What do you send back?" (A "Lost" card with 3 on it.)

Vocabulary

lost packet, copy, send again, missing, twice

Differentiation

- **Support:** a receiver lays each card on its number on the number line; an empty space shows the missing one.
- **Stretch:** a message of 6 packets where 2 are lost. Children say which two and write both numbers on one "Lost" card.
- **Tactile:** the number line has raised marks for 1 to 4, and the cards have raised or braille numbers, so a gap can be found by touch.

⚠ Safety

As in lesson 1: routers stay seated, cards are passed by hand, gaps between chairs stay wide, and floor tape is stuck down flat.

Teacher notes and misconceptions

- Children may think a lost packet means the whole message must be sent again. Only the missing piece is needed.
- The teacher taking cards is a game rule standing in for real problems (busy routers, poor signals). Make clear no child "lost" anything; it is part of the game.

- Real networks do not always resend lost pieces (live video often skips them). This lesson shows only one way of dealing with loss.

Lesson 3: What Little Hill library lends

Unit overview - Previous: Lesson 2 - Next: Lesson 4

About 45 minutes.

Goal

Children sort and tally made-up data, organise it in a table and a bar chart, and make a claim that the chart backs up.

Materials

- One set of 30 loan slips per group of four. Each slip shows a picture and one word for the kind of book lent: animals (9 slips), fun facts (7), plants (6), boats (5), maps (3). Each kind also has its own raised mark.
- Squared paper (1 cm or larger), rulers and pencils for each group.
- A class table and a blank bar chart drawn on the board.
- A peg board (each peg is one slip) for tactile charts.
- A lap tray for a child in a wheelchair, if needed.
- Paper and pencils for the closing check.

The story

Little Hill library (made up for this unit) lends books. Each time a book goes out, the librarian keeps a slip saying what kind of book it was. These are the slips from one week. The librarian wants to know which kind of book to buy more of.

Steps

✎ Opening (5 min)

1. Say: "Information we collect, like these slips, is called **data**. A pile of slips is hard to read. Today we organise it so we can see what it tells us."
2. Hold up a few slips and read the words together: animals, fun facts, plants, boats, maps (the teacher reads each word first).

✎ Main activity (30 min)

Part 1: sort and tally (8 min). Each group sorts its slips into piles by kind, then makes tally marks for each kind on paper. Groups check by counting the piles again.

Part 2: a table (5 min). Groups copy and fill a table:

Kind of book	Tally	Number
animals		
fun facts		

Kind of book	Tally	Number
plants		
boats		
maps		
total		

Answers: animals 9, fun facts 7, plants 6, boats 5, maps 3, total 30.

Part 3: a bar chart (10 min). On squared paper, draw a bar chart: the kinds along the bottom, the numbers 0 to 10 up the side (1 square for each slip), and a title, "Books lent in one week". Each group draws one bar for each kind. Compare with the class chart on the board.

Part 4: a claim with evidence (7 min). Explain: "A **claim** is something we say is true. **Evidence** is what shows it." Groups finish one sentence:

- "The library should buy more ___ books, because the chart shows ___."

(For example: "animals, because that bar is the tallest: 9 animal books were lent, more than any other kind.")

Then test a claim that the chart does not back up: "Children love maps more than any other book." (No: maps has the shortest bar, 3.)

✓ Discussion questions (5 min)

- Which kind was lent most? Fewest? (Animals, 9; maps, 3.)
- How many more animal books than map books? (9 - 3 = 6.)
- Was the bar chart or the pile of slips quicker to read? Why? (The chart: you can see the tallest bar straight away.)
- Can the chart tell us **why** animal books were lent most? (No. It shows how many, not why.)

✓ Closing check (5 min)

On paper: "Fun facts 7, boats 5. Draw these two bars." Then: "Which is taller, and by how many squares?" (Fun facts, by 2.)

Vocabulary

data, tally, table, bar chart, claim, evidence, total

Differentiation

- **Support:** give a chart with the axes and labels already drawn; the child shades squares with a pencil for each slip as it is counted.
- **Stretch:** "The library can buy 10 new books. How would you share them out, and how does the chart back up your plan?" (Any plan with reasons from the chart.)

- **Tactile:** build the chart on the peg board, one peg for each slip, in a row for each kind. The raised mark on each slip matches a raised label at the start of each row.

⚠ Safety

No special risks. Slips are card, not small items.

Teacher notes and misconceptions

- Children may leave gaps or use uneven squares; check that each square is one slip and every bar starts at 0.
- Some children read the tallest bar as "the best book". Stress that it shows the most lent that week, not the best.
- Keep the data made up. Do not collect children's own reading choices for this lesson.

Lesson 4: Rain, visitors and a new sign

Unit overview - Previous: Lesson 3 - Next: Lesson 5

About 45 minutes.

Goal

Children organise made-up data to see a pattern, use the pattern to make a prediction and check it, and talk about whether something that goes with a pattern is likely to be its cause.

Materials

- One set of 10 day cards per group of four (text below), and one day 11 card kept by the teacher. Rain cards have a raised mark; dry cards are smooth. Cards for days 6 to 10 also show a small picture of the new sign.
- Squared paper (1 cm or larger), rulers and pencils for each group.
- A class bar chart drawn on the board, with 0 to 22 up the side.
- A peg board for tactile charts.
- A lap tray for a child in a wheelchair, if needed.
- Paper and pencils for the closing check.

The data

Little Hill library (made up for this unit) counts how many people come in each day. Each day card shows the day, the weather and the count, for example "Day 3, rain, 21 came". The teacher reads "dry" and "sign" aloud first.

Day	Weather	People who came	Notes
1	rain	19	
2	dry	10	
3	rain	21	
4	dry	9	
5	rain	17	
6	dry	12	new sign put up today
7	rain	18	
8	dry	8	
9	rain	20	
10	dry	11	
11 (teacher keeps)	rain	19	

Steps

✎ Opening (5 min)

1. Recall lesson 3: "What is a claim? What is evidence?"

2. Say: "Today we use data to **predict**: to say what will probably happen next. Then we check."

✎ Main activity (30 min)

Part 1: sort the days (8 min). Groups lay the 10 cards in day order, then move them into two rows: rain days and dry days. Ask: "What do you notice?" (Every rain day had 17 to 21 people. Every dry day had 8 to 12. Rain days always had more.)

Part 2: a bar chart (10 min). Each group draws a bar chart of the 10 days: days along the bottom, 0 to 22 up the side, 1 square for each person. Write "R" above each rain bar. Ask: "Can you see the pattern in the chart?" (The R bars are all taller.)

Part 3: predict and check (5 min). "Tomorrow, day 11, it will rain. How many people do you predict will come?" Each group writes a number and a reason. (Any number from about 17 to 21, because rain days had 17 to 21.) Then reveal the day 11 card: rain, 19 came. Ask: "Was your prediction close?" Stress that a prediction can be a little wrong and still be good.

Part 4: cause or not? (7 min). Ask two questions.

1. "The library put up a new sign on day 6. Did the sign make more people come?" Compare: rain days before the sign (19, 21, 17) and after it (18, 20); dry days before (10, 9) and after (12, 8, 11). (No: rain days still had 17 to 21 and dry days still had 8 to 12. The sign did not change the pattern, so it is probably not the cause.)
2. "Kit, who works at the library, wears a green hat on every rain day. More people come on the days Kit wears the hat. Does the hat bring the people?" (Probably not. The hat and the people both go with rain. Rain is a more likely cause: people may come in to stay out of the rain.)

Say: "Data can show that two things **go together**. That is a clue to the cause, but it does not prove it."

✓ Discussion questions (5 min)

- What pattern did the data show? (More people came on rain days.)
- How did the pattern help you predict? (Day 11 was rain, so we expected 17 to 21.)
- Why is the sign probably not the cause? (Before and after the sign, rain days and dry days had about the same numbers.)
- Why is the hat probably not the cause? (It goes with rain; the rain is a more likely reason people came.)

✓ Closing check (5 min)

On paper: "Day 12 will be dry. Predict how many people will come, and say why." (A number from about 8 to 12, because dry days had 8 to 12.)

Vocabulary

predict, pattern, cause, go together, prediction, check

Differentiation

- **Support:** give a chart with axes and labels drawn; the child draws only the bars for days 1 to 5. Work with the two rows of cards rather than the chart.
- **Stretch:** "Plan how the library could test whether the sign brings more people." (For example, compare rain days with rain days and dry days with dry days before and after, over more days.)
- **Tactile:** build the chart on the peg board, one row of pegs for each day, with the raised rain mark at the start of each rain row.

⚠ Safety

No special risks. Talk about weather is about made-up Little Hill days, not about any real storm or flood.

Teacher notes and misconceptions

- "Goes together" and "causes" are often mixed up by adults too. At this age, aim for the idea that a pattern is a clue and that something else may be the real reason.
- The rain cause is also only likely, not proved. Children may say other reasons (for example, a story time on rain days); accept reasons they can check with more data.
- "Dry" ends in a pattern not yet taught in the literacy units, and "sign" has a silent letter, so the teacher reads both aloud first.

Lesson 5: Keeping information private

Unit overview - Previous: Lesson 4

About 45 minutes.

Goal

Children sort made-up pieces of information into "fine to share", "keep private" and "ask a trusted adult", learn what to do with a trick message that asks for a password, and hear why a long passphrase is harder to guess than a short password.

Materials

- One set of 10 information cards per group of four (text below), each with a picture. The teacher reads every card aloud first.
- Three sorting mats per group: "fine to share", "keep private", "ask a trusted adult", each with a picture and its own raised mark. The teacher reads the mat words aloud first.
- A large printed copy of the trick message card (text below).
- Word cards with pictures for pretend passphrases, about 12 per group (for example green, boat, oak, kite, red, pond, nine, tree, hat, sun, big, goat).
- A lap tray for a child in a wheelchair, if needed.
- Paper and pencils for the closing check.

The information cards

All the people are made up. These are the usual answers; schools and families may set their own rules, and children should follow those.

Card	Usual answer
1. Kit's password for a reading game	keep private
2. Meg's home address	keep private
3. The name of Little Hill library	fine to share
4. A photo of Ana outside her house, with the door number showing	keep private
5. Mo likes books about animals	fine to share
6. Sana's full name and birthday	keep private
7. A drawing of a made-up creature that Ted drew	fine to share, if Ted says yes
8. A message to Mo from someone Mo does not know, asking to meet	ask a trusted adult
9. The score in the Little Hill hoop-toss game	fine to share
10. The name of Ted's school and the time Ted walks home	keep private

The trick message

> Hi Mo! You have won a prize! Send me your password and I will send you the prize.

Steps

✎ Opening (5 min)

1. Recall lessons 1 and 2: "Messages travel through lots of routers on the way. Who might see a message?" (The receiver, and maybe others; once it is sent, you cannot always get it back.)
2. Say: "Some information is fine to share. Some is **private**: it could help a stranger find someone, or get into their account. Today we sort made-up information."

✎ Main activity (30 min)

Part 1: sort the cards (12 min). The teacher reads each card aloud, and groups place it on a mat. Go through the answers together, asking for a reason each time. Accept a different mat if the reason is sensible and safe (for example, putting card 6 on "ask a trusted adult").

Part 2: the trick message (8 min). Show the trick message card and read it aloud. Ask:

- "Should Mo send the password?" (No.)
- "What clues show it is a trick?" (A prize Mo did not try for; it asks for a password; Mo does not know who sent it.)
- "What should Mo do?" (Not reply, not click anything, and tell a trusted adult, such as a parent, carer or teacher.)

Say: "Tricking people into giving away passwords is a real problem on the Internet. Grown-ups get these messages too."

Part 3: strong passwords (10 min). Explain: "A password keeps an account locked. A short password, like a pet's name, is easy to guess. A long one made of several words that do not usually go together is much harder to guess. This is called a **passphrase**."

Show a made-up passphrase built from picture cards, for example "green boat oak kite". Say clearly: "This one is only an example. Never use an example from a lesson. Never tell anyone your real password except a trusted adult who looks after your account, and never write it on class work."

Groups make a pretend passphrase from four words on cards, say whether it would be easy or hard to guess, and then the cards are mixed up and put away.

✓ Discussion questions (5 min)

- Why is a home address private? (It could help a stranger find someone.)
- Why can Ted's drawing be fine to share? (It does not say who or where Ted is, as long as Ted agrees.)
- What should you do if a message worries you? (Stop, do not reply, and tell a trusted adult.)
- Is "Kit" a good password? (No; it is short and easy to guess.)

✓ Closing check (5 min)

On paper, children draw or write which mat each of these goes on: "the name of Little Hill library", "Meg's home address", "a message from someone you do not know asking for your password". (Fine to share; keep private; keep private and tell a trusted adult.)

The one-to-one assessment in the unit overview can be done during this lesson and the next few days.

Vocabulary

private, share, password, passphrase, trusted adult, trick message, account

Differentiation

- **Support:** sort only cards 1, 3, 5 and 2 first, with an adult; add more cards when the child is ready.
- **Stretch:** "Write a short note to Mo explaining how to spot a trick message." (Any note with at least two clues and "tell a trusted adult".)
- **Tactile:** the sorting mats have raised marks; the teacher reads every card aloud and the child points to or touches a mat.

⚠ Safety

- Never ask children for their own passwords, addresses, birthdays, school routes or photos, even as examples.
- Some children may have had a worrying message or experience online. Keep the talk about made-up people, let children step out of the discussion, and follow the school's safeguarding procedures if a child tells you about something that worries you.

Teacher notes and misconceptions

- Children may think private means "bad" or "secret from parents". Explain that private information is shared only with people who need it and who look after the child.
- Some children think only strangers' messages can be tricks. Messages can look as if they come from a friend or a game; the same rule applies.
- The usual answers above are a starting point. Follow the school's own online-safety rules where they differ.

Binary, Pixels, Logic Gates, and Packet Switching

Subject: Computing Grade: 6 Lessons: 5 Time per lesson: 50 minutes Status: written Author: Gemini 4 Argon

Navigation: Repository Home | Computing Progression

Big idea

Computers store every number, letter, and image using physical two-state switches (`0` and `1`), combine those switches with simple rules (`AND`, `OR`, `NOT`) to make decisions, and send data across networks by splitting messages into numbered packets that can be checked for errors.

What students should be able to do

- Convert whole numbers between `0` and `255` to and from 8-bit binary using repeated-doubling place-value cards (`128, 64, 32, 16, 8, 4, 2, 1`), and explain why adding one more card doubles the total number of patterns.
- Encode and decode short words using a 7-bit/8-bit ASCII lookup table and encode/decode a `8 × 8` bitmap icon, calculating the unit rate in **bits per pixel** and the image's **width-to-height ratio**.
- Complete truth tables for `NOT`, `AND`, and `OR` gates and trace the output of a 2-gate combination circuit for all four possible input pairs.
- Simulate routing a multi-packet message across a 6-node network with a broken link, reassemble out-of-order packets using sequence headers, calculate delivery unit rates (**packets per second** and **bits per second**), and detect a single flipped bit using an **even parity bit**.

What students should already know

- Whole-number addition and subtraction up to `300`, and multiplication/division by `2`.
- Base-10 place value (`ones, tens, hundreds`).
- **No exponent notation (`2^n`) is required.** Lesson 1 builds binary place values directly through repeated doubling (`1, 2, 4, 8, 16, 32, 64, 128`).
- Basic ratio and unit-rate language (such as `3 : 2` or `bits per pixel`), reinforced alongside Grade 6 Mathematics (`units/mathematics/6-ratios/`).

Lesson list

1. Lesson 1: Counting with Switches — Repeated Doubling and 8-Bit Binary: Build a set of 8 doubling dot-cards (`1` through `128`) to represent any number from `0` to `255` using only `ON (1)` and `OFF (0)`.
2. Lesson 2: Letters and Pictures in Bits — ASCII Text and Bitmap Pixels: Decode secret binary text messages and design `8 × 8` 1-bit and 2-bit pixel art while comparing storage unit rates (`bits per character` and `bits per pixel`).

3. Lesson 3: Making Decisions with Switches — NOT, AND, and OR Logic Gates: Model series and parallel switches with human circuit paths and truth tables to design a 3-sensor greenhouse/terrarium alarm rule.
4. Lesson 4: Postcards on the Wire — Packet Switching, Routing, and Throughput Rates: Break a message into numbered 64-bit packets, route around a failed node on a classroom network map, and compute network unit rates (`packets per second` and `bits per second`).
5. Lesson 5: Catching Flipped Bits — Noise, Even Parity Checks, and End-of-Unit Assessment: Add a 9th parity bit to catch single-bit transmission errors, test where 1-bit parity fails, and complete the 5-part unit assessment.

Materials (whole unit)

- **Index cards or half-sheets of paper** (8 cards per student pair for Lesson 1 binary dot-cards; 6 cards per group for Lesson 4 packet postcards).
- **Grid paper (`8 × 8` squares)** or printed `8 × 8` blank grids for Lesson 2 bitmap pixel design.
- **Pencils, markers, and large two-sided paper cards or tokens** (or written/pointed `0`/`1` table charts as an alternative to handling small objects) for flipping bits in Lesson 5.
- **No computers, batteries, or internet connection required** (100% unplugged classroom-ready; optional browser verification extensions noted in Teacher Notes).

Assessment

Each lesson includes 8 worked/practice tasks with complete step-by-step answers (40 lesson tasks total). Lesson 5 concludes with a **20-point End-of-Unit Synthesis Assessment** (5 multi-part problems covering binary conversion, pixel bit-rate ratios, a 2-gate truth table, packet routing/throughput, and even-parity error detection) with a full scoring rubric and worked answer key.

Sources

- Bell, T., Witten, I. H., & Fellows, M. (2015). CS Unplugged: An enrichment and extension programme for primary-aged students (openly licensed CC BY-NC-SA reference for dot-card and parity unplugged pedagogy; all text, numbers, diagrams, and problems in this unit are newly written from scratch under CC0): https://www.csunplugged.org/
- American National Standards Institute (1968/1986). ANSI X3.4 — Coded Character Sets — 7-Bit American National Standard Code for Information Interchange (7-Bit ASCII) (public-domain numerical mapping of standard printable characters `32–126`): https://www.rfc-editor.org/rfc/rfc20
- Postel, J. (Ed.) (1981). RFC 791 — Internet Protocol: DARPA Internet Program Protocol Specification (public-domain specification of packet headers, addressing, and datagram routing): https://www.rfc-editor.org/rfc/rfc791

Known gaps and cautions

- **Unreviewed AI draft:** This unit has not yet been reviewed by a human classroom teacher or tested with Grade 6 students.
- **Simplifications stated explicitly for students:**
- Lesson 2 uses standard 7-bit/8-bit ASCII for English letters (`A–Z`, `a–z`) and notes explicitly that modern software uses **UTF-8 / Unicode** so scripts and emoji from all languages can be represented.

- Lesson 4 models packet headers with three fields (`From`, `To`, `Packet # of Total`) rather than a full 20-byte IPv4 / TCP header.
- Lesson 5 introduces **single-bit even parity** and explicitly shows students why two simultaneous bit flips inside the same byte go undetected, motivating stronger checksums (like CRC and cryptographic hashes) in later grades.

Lesson 1: Counting with Switches — Repeated Doubling and 8-Bit Binary

Time: 50 minutes

Unit: Binary, Pixels, Logic Gates, and Packet Switching | **Next:** Lesson 2: Letters and Pictures in Bits

Goal

Convert whole numbers between `0` and `255` to and from 8-bit binary using repeated-doubling place-value cards (`128, 64, 32, 16, 8, 4, 2, 1`), and explain why adding one more card doubles the number of possible patterns.

Materials

- 8 index cards (or half-sheets of paper) per pair of students.
- Thick dark marker for drawing dots/numbers on one side of each card.
- Chalkboard or whiteboard for class place-value comparison.

Steps

✎ 1. Opening (5 min): Why Only Two States?

Hold up a classroom light switch (or draw a simple switch on the board). Ask: - "In everyday base-10 arithmetic, each digit slot can hold 10 different symbols (*`0`* through *`9`*). Why doesn't a computer wire use 10 different voltage levels (*`0V, 1V, 2V, ..., 9V`*) for each digit?" - Guide students to notice that electrical wires experience tiny fluctuations (heat, static, long cables). Distinguishing `10` close voltage levels is fragile, whereas distinguishing **two clearly separated states** — `OFF (0)` vs. `ON (1)` (or dark vs. bright, unmagnetized vs. magnetized) — is fast and reliable.

✎ 2. Main Activity Part A (15 min): Building the 8 Doubling Cards

Have each student pair make a row of 5 cards first, arranged **right to left** just like base-10 place value: 1. On the rightmost card, write **`1`** and draw `1` dot. 2. Move one slot left: **double** the previous card to write **`2`** (`2` dots). 3. Move one slot left: **double** `2` to write **`4`** (`4` dots). 4. Move one slot left: **double** `4` to write **`8`** (`8` dots). 5. Move one slot left: **double** `8` to write **`16`** (`16` dots).

Now extend the row to a full **byte (8 bits)** by doubling three more times to the left: **`32`**, **`64`**, and **`128`**:

Bit slot (left to right)	Card 8	Card 7	Card 6	Card 5	Card 4	Card 3	Card 2	Card 1
Place value (doubling right to left)	128	64	32	16	8	4	2	1

Explain the rule: - Face-up (`ON`) = **`1`** (add this card's value to your total). - Face-down (`OFF`) = **`0`** (do not add this card's value).

✎ Worked Example 1: Binary to Base-10 (`00101101`)

Suppose the 8 cards read `0 0 1 0 1 1 0 1`:

Card value	128	64	32	16	8	4	2	1
Bit (1 = ON, 0 = OFF)	0	0	1	0	1	1	0	1

Add only the cards switched `ON (1)`: `32 + 8 + 4 + 1 = 45`.

✎ Worked Example 2: Base-10 to 8-Bit Binary (`100`)

To build a target number like **`100`**, always work **from left (`128`) to right (`1`)** using the greedy subtraction check: 1. Does `128` fit into `100`? No (`128 > 100`) → write **`0`**, remainder is still `100`. 2. Does `64` fit into `100`? Yes (`64 <= 100`) → write **`1`**, subtract `100 - 64 = 36` remaining. 3. Does `32` fit into `36`? Yes (`32 <= 36`) → write **`1`**, subtract `36 - 32 = 4` remaining. 4. Does `16` fit into `4`? No → write **`0`**. 5. Does `8` fit into `4`? No → write **`0`**. 6. Does `4` fit into `4`? Yes (`4 <= 4`) → write **`1`**, subtract `4 - 4 = 0` remaining. 7. Does `2` fit into `0`? No → write **`0`**. 8. Does `1` fit into `0`? No → write **`0`**. Result: **`01100100`** (`64 + 32 + 4 = 100`).

✓ 3. Discussion Questions (10 min)

- **Question A:** "Look at the sum of all cards to the right of *`16`*: *`8 + 4 + 2 + 1 = 15`*. Look at all cards to the right of *`64`*: *`32 + 16 + 8 + 4 + 2 + 1 = 63`*. What pattern do you notice?"
- Example good answer: Every card is always **`1`** **more** than the sum of all smaller cards to its right! That means if your remaining target is `64` or more, you must turn on the `64` card because all smaller cards combined only reach `63`. This guarantees every whole number from `0` to `255` has **one and only one** 8-bit binary pattern.
- **Question B:** "How can you tell in one second whether an 8-bit binary number is odd or even?"
- Example good answer: Look only at the rightmost card (`1`). Every other card (`128, 64, 32, 16, 8, 4, 2`) is an even number, so the sum is odd if and only if the `1` bit is `1`.

✓ 4. Student Practice & Closing Check (20 min)

✎ Practice Tasks (1.1 – 1.8)

1. **Task 1.1:** Convert the 8-bit binary number `00010110` to a base-10 number.
2. **Task 1.2:** Convert the 8-bit binary number `10000101` to a base-10 number.
3. **Task 1.3:** Convert the 8-bit binary number `11111111` (all 8 cards turned ON) to a base-10 number.
4. **Task 1.4:** Write the base-10 number **`19`** in 8-bit binary.
5. **Task 1.5:** Write the base-10 number **`75`** in 8-bit binary.

6. **Task 1.6:** Write the base-10 number **200** in 8-bit binary.
7. **Task 1.7 (Pattern Reasoning):** Without converting both numbers to base-10 first, what happens to the value of `00001101` (`13`) when you shift every bit one slot to the left and put a `0` on the right (`00011010`)? Explain why.
8. **Task 1.8 (Capacity & Doubling):** A 4-bit system can represent `16` different numbers (`0` through `15`). How many different numbers can a 5-bit system represent, and what is the largest number it can hold?

Complete Answer Key (Tasks 1.1 – 1.8)

- **Answer 1.1:** `16 + 4 + 2 = 22`.
- **Answer 1.2:** `128 + 4 + 1 = 133`.
- **Answer 1.3:** `128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255` (which is `1` less than the next doubling value, `256`).
- **Answer 1.4:** `19 = 16 + 2 + 1` → **`00010011`**.
- **Answer 1.5:** `75 = 64 + 8 + 2 + 1` → **`01001011`**.
- **Answer 1.6:** `200 = 128 + 64 + 8` → **`11001000`**.
- **Answer 1.7:** Shifting left by one slot **doubles** the number from `13` (`8 + 4 + 1`) to `26` (`16 + 8 + 2`), because every ON bit moves to a card worth twice as much (just as appending a `0` in base-10 multiplies by `10`).
- **Answer 1.8:** Adding a 5th bit (the `16` card) **doubles** the number of patterns from `16` to **`32` different numbers** (`16` patterns with the `16`-card OFF, plus `16` patterns with the `16`-card ON). Because `0` uses one pattern, the range is `0` through **`31`** (`16 + 8 + 4 + 2 + 1 = 31`).

Vocabulary

- **bit (binary digit):** A single two-state switch or symbol that can be either `0` (`OFF`) or `1` (`ON`).
- **byte:** A group of `8` bits treated as a single unit, capable of representing `256` distinct patterns (`0` through `255`).
- **place value:** The numerical value assigned to a digit slot based on its position (`1, 2, 4, 8, 16, 32, 64, 128` from right to left in binary).

Differentiation

- **Extra support:** Start with only the 4 rightmost cards (`8, 4, 2, 1`) with pre-drawn dots (or large paper tokens, pointing, or verbal counting rather than small loose objects) so students can count the dots on each card directly before flipping cards face-up or face-down to build `1` through `15`.
- **Extra challenge:** Ask students how a computer could represent **negative numbers** or **fractions** using 8 bits, or challenge them to add two 8-bit binary numbers (`00010110 + 00001011`) column by column using the carry rule `1 + 1 = 2` (write `0`, carry `1` to the next doubled slot).

⚠ Safety

No physical hazards in this unplugged lesson.

Teacher notes

- **Common misconception 1 (Reading left-to-right vs. right-to-left):** Some students place `1` on the far left and `128` on the far right. Remind them that binary uses the exact same convention as base-10: the **smallest place value (`1`) is always on the far right**, and values grow larger as you move left.
- **Common misconception 2 (Forgetting that `00000000` counts as a pattern):** Students often ask why 8 bits (`256` patterns) maxes out at `255` instead of `256`. Point out that in base-10, two digits give `100` patterns (`00` through `99`), not `100`!

Lesson 2: Letters and Pictures in Bits — ASCII Text and Bitmap Pixels

Time: 50 minutes

Unit: Binary, Pixels, Logic Gates, and Packet Switching | **Previous:** Lesson 1 | **Next:** Lesson 3

Goal

Encode and decode text using an 8-bit ASCII lookup table, represent `8 × 8` images as 1-bit and 2-bit bitmaps, and calculate storage unit rates in **bits per character** and **bits per pixel**.

Materials

- Lesson 1 binary place-value cards (`128, 64, 32, 16, 8, 4, 2, 1`).
- Printed or board-drawn `8 × 8` blank pixel grids (2 grids per student).
- Colored pencils or shading styles (blank, light crosshatch, dark diagonal, solid black) for 2-bit palettes.

Steps

✎ 1. Opening (5 min): If Computers Only Store `0` and `1`, Where Are the Letters?

Write the byte `01001000` on the board. Ask: - "In Lesson 1, we decoded *`01001000`* as the number *`64 + 8 = 72`*. How can a text-messaging app use that exact same byte to display a capital letter instead of the number *`72`*?" - Introduce the idea of a **shared codebook (standard encoding)**: if every keyboard and screen agrees on which number stands for which letter, then storing numbers is enough to store words.

✎ 2. Main Activity Part A (15 min): Decoding and Encoding ASCII Text

Show the class how uppercase English letters are mapped in standard **ASCII** (American Standard Code for Information Interchange):

Character	Base-10 Code	8-Bit Binary	Character	Base-10 Code	8-Bit Binary
`(space)`	32	`00100000`	A	65	`01000001`
!	33	`00100001`	B	66	`01000010`
0 (digit)	48	`00110000`	C	67	`01000011`
1 (digit)	49	`00110001`	`...`	`...`	`...`
?	63	`00111111`	Z	90	`01011010`

Point out two helpful patterns: 1. Capital **`A` is `65` (`01000001`)**, so the `n`-th letter of the alphabet has code **`64 + n`**! Since `64` is the `01000000` bit, the last 5 bits directly tell you the letter's

position (1 to 26) in the alphabet. 2. Lowercase **a is 97 (01100001)**, which is 65 + 32 — toggling just the 32-bit switches any letter between uppercase and lowercase!

✎ Worked Example 1: Decoding a 3-Byte Word

Decode 01000011 01000001 01010100: - Byte 1 (01000011): 64 + 2 + 1 = 67 (3rd letter) → **C** - Byte 2 (01000001): 64 + 1 = 65 (1st letter) → **A** - Byte 3 (01010100): 64 + 16 + 4 = 84 (20th letter) → **T** Word: **CAT** (3 characters × 8 bits per character = 24 bits).

✎ 3. Main Activity Part B (15 min): Bitmap Grids and Bits-per-Pixel Ratios

Now ask: "How do we turn a grid of tiny squares (**pixels**) into *0*s and *1*s?" - **1-Bit Monochrome (1 bit per pixel):** Each square uses 1 bit: 0 = white (unshaded), 1 = black (shaded). - Example 8 × 8 icon (64 pixels total at 1 bit per pixel = 64 bits = 8 bytes of **raw uncompressed pixel data**, excluding any file headers or metadata): - Row 1: 00111100 - Row 2: 01000010 - Row 3: 10100101 - Row 4: 10000001 - Row 5: 10100101 - Row 6: 10011001 - Row 7: 01000010 - Row 8: 00111100 (Students shade the *1*s on their *8 × 8* grid to reveal a smiling face!) - **2-Bit Four-Color Palette (2 bits per pixel):** What if we want **4 colors** (like an early handheld game console: 00 = White, 01 = Light Gray, 10 = Dark Gray, 11 = Black)? - Each pixel now requires **2 bits**, so the unit rate doubles from 1 bit per pixel to **2 bits per pixel**. - For the same 8 × 8 (64-pixel) grid, raw pixel storage doubles from 64 bits (8 bytes) to **128 bits (16 bytes)** (excluding any metadata or compression).

4. Discussion & Student Practice (15 min)

✎ Practice Tasks (2.1 – 2.8)

1. **Task 2.1:** Using A = 65, find the base-10 ASCII code and 8-bit binary byte for uppercase **E** (the 5th letter) and **M** (the 13th letter).
2. **Task 2.2:** Decode the 4-byte ASCII message: 01000100 01000001 01010100 01000001.
3. **Task 2.3:** Look at the ASCII table: the character '1' has code 49 (00110001), while the number 1 in Lesson 1 was 00000001. Why does a computer distinguish between the text character '1' and the number 1?
4. **Task 2.4 (Unit Rate — Text):** A text file contains 150 characters (including spaces) encoded at a unit rate of 8 bits per character. How many total **bits** and how many **bytes** does the text file require?
5. **Task 2.5 (Bitmap Decoding):** Draw a 4 × 4 grid and shade the 1s for these four 4-bit rows: Row 1 = 0110, Row 2 = 1111, Row 3 = 0110, Row 4 = 0110. What symbol appears?
6. **Task 2.6 (Color Depth Ratio):** An 8 × 8 sprite uses a 16-color palette where each pixel uses 4 bits (0000 through 1111). What is the unit rate in **bits per pixel**, and how many **raw pixel bits** (and **bytes**, excluding headers/compression) are needed for all 64 pixels?
7. **Task 2.7 (Aspect Ratio):** A game banner measures 120 pixels wide by 80 pixels tall. Write the width-to-height ratio 120 : 80 in simplest whole-number form. If you scale the banner to 300 pixels wide while keeping the exact same aspect ratio, how many pixels tall must it be?

8. **Task 2.8 (Metadata Reasoning):** Suppose you receive a raw stream of `64` bits (`0`s and `1`s) for a 1-bit black-and-white picture, but nobody tells you whether the grid is `8 × 8`, `16 × 4`, or `32 × 2`. Why does the receiver need to know the **width and height** (whether stored in a file header, sent by a communication protocol, or fixed by a prior external agreement) to display the pixel bits correctly?

Complete Answer Key (Tasks 2.1 – 2.8)

- **Answer 2.1:** `E` is the 5th letter: `64 + 5 = 69` → **`01000101`**. `M` is the 13th letter: `64 + 13 = 77` (`64 + 8 + 4 + 1`) → **`01001101`**.
- **Answer 2.2:** `01000100` = `64 + 4 = 68` (`D`); `01000001` = `65` (`A`); `01010100` = `64 + 16 + 4 = 84` (`T`); `01000001` = `65` (`A`). Word: **`DATA`**.
- **Answer 2.3:** When `'1'` appears inside a sentence or street address (like `"12 Oak St"`), the screen needs to look up the font glyph for the symbol `'1'` alongside letters and punctuation; when storing a quantity for arithmetic, storing pure binary (`00000001`) lets the processor add and subtract directly.
- **Answer 2.4:** `150 characters × 8 bits/character = 1,200 bits`, which is **`150 bytes`** (`1 byte per character`).
- **Answer 2.5:** Shading the middle two columns on rows 1, 3, and 4 and all four columns on row 2 forms a **cross / plus (+) or sword/tree icon**.
- **Answer 2.6:** Unit rate = **`4 bits per pixel`**. Total storage = `64 pixels × 4 bits/pixel = 256 bits`, which is `256 ÷ 8 =` **`32 bytes`**.
- **Answer 2.7:** Dividing both `120` and `80` by their greatest common factor (`40`) gives a **`3 : 2`** width-to-height ratio. Since `300 ÷ 3 = 100`, the scaled height must be `2 × 100 =` **`200` pixels tall**.
- **Answer 2.8:** Without knowing how many pixels belong in each row (the width—supplied via a file header, a protocol message, or a fixed format agreement), the receiver does not know where to wrap to the next line; wrapping every `16` bits instead of every `8` bits scrambles the picture into unrecognizable stripes.

Vocabulary

- **ASCII:** A standard character-encoding table that assigns numbers (`0` to `127`) to English letters, digits, and punctuation marks.
- **pixel (picture element):** One individual colored square in a grid-based digital image.
- **bitmap:** An image represented as a row-by-row grid of pixel color codes (whose width and height are supplied by a file header, protocol, or format specification).
- **color depth (bits per pixel):** The unit rate measuring how many bits are used to represent the color of each single pixel.

Differentiation

- **Extra support:** Provide a printed alphabet strip listing `A = 65` through `Z = 90` with their 5-bit endings (`00001` to `11010`) so students can focus on matching patterns before doing mental subtraction.
- **Extra challenge:** Introduce **Run-Length Encoding (RLE)** compression: in Row 1 of the smiley face (`00111100`), instead of listing every bit, write `(2 white, 4 black, 2 white)`. Ask students to test which `8 × 8` drawings shrink with RLE and which pattern (such as an alternating checkerboard `01010101`) actually gets larger!

⚠ Safety

No physical hazards in this unplugged lesson.

Teacher notes

- **Important scope note on Unicode / UTF-8:** Make sure students know that 7-bit/8-bit ASCII only had room for English characters. Modern operating systems and web browsers use **UTF-8 (Unicode)**, which stays backwards-compatible with 1-byte ASCII for `0–127` while using `2`, `3`, or `4` bytes per character to represent over `149,000` characters across global writing systems and emoji.

Lesson 3: Making Decisions with Switches — NOT, AND, and OR Logic Gates

Time: 50 minutes

Unit: Binary, Pixels, Logic Gates, and Packet Switching | **Previous:** Lesson 2 | **Next:** Lesson 4

Goal

Complete truth tables for `NOT`, `AND`, and `OR` logic gates, model series and parallel switch paths, and trace the output of a 2-gate combination circuit for all four input pairs.

Materials

- Chalk or masking tape on the classroom floor to draw two circuit pathways (**Series Path** and **Parallel Path**).
- Index cards labeled `0 (OPEN / FALSE)` and `1 (CLOSED / TRUE)` for student gatekeepers.
- Student notebooks or worksheet tables for truth tables.

Steps

✎ 1. Opening (5 min): How Do `0`s and `1`s Make Decisions?

Ask the class: - "Lessons 1 and 2 showed how *0*s and *1*s store numbers, letters, and pictures. But how does a computer actually **decide** something — like turning on a terrarium misting sprayer only when the soil is dry **AND** the lid is closed?" - Explain that inside a processor chip, billions of microscopic electronic switches (**logic gates**) take `0`/`1` inputs and produce a `0`/`1` output according to three fundamental rules: **`NOT`**, **`AND`**, and **`OR`**.

✎ 2. Main Activity Part A (15 min): Acting Out the Three Fundamental Logic Gates

Select student volunteers to act as **Switch Gatekeepers** along chalk paths drawn on the floor (or trace the paths at desks using a tabletop diagram, paper switch cards, or pointing so standing/walking is optional): - **`1` = Gate Closed (Bridge Down / True):** The messenger can walk across. - **`0` = Gate Open (Bridge Up / False):** The path is blocked.

1. The `NOT` Gate (Inverter) — 1 Input

Flips the input bit to its opposite:

Input `A`	Output `NOT A`
`0`	**`1`**
`1`	**`0`**

Everyday example: A refrigerator interior light switch that turns the bulb **ON (1)** when the door is **NOT closed (0)**.

2. The AND Gate (Series Path) — 2 Inputs

Draw two switches A and B **in a single straight line (in series)**. To walk from the battery to the lightbulb, the messenger must cross **both** A and B:

Input A	Input B	Output A AND B
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Rule: A AND B outputs 1 **only when both inputs are 1**.

3. The OR Gate (Parallel Path) — 2 Inputs

Draw a path that **splits into two side-by-side branches** (Switch A on the top branch, Switch B on the bottom branch) and rejoins before the lightbulb. The messenger can reach the lightbulb if **either** A is closed, **or** B is closed, **or both**:

Input A	Input B	Output A OR B
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

Rule: A OR B outputs 1 **whenever at least one input is 1**.

✎ 3. Main Activity Part B (15 min): Combining Gates in a Terrarium Controller

Connect to the Grade 6 Science unit (Fungi and Decomposition): suppose your class is modeling an **ideal Boolean simulation** of a mushroom decomposition terrarium controller with three idealized binary inputs: - **D (Dry):** 1 if humidity is too low, 0 if moist enough. - **L (Lid Closed):** 1 if the terrarium lid is safely closed, 0 if open. - **M (Manual Override Button):** 1 if a student presses the test-spray button, 0 otherwise.

✎ Worked Example: Tracing (D AND L) OR M

Let's trace when the **Sprayer (S)** turns ON (S = 1) for different sensor readings by adding an intermediate column X = (D AND L) first:

D (Dry)	L (Lid)	M (Manual)	Step 1: X = D AND L	Step 2: S = X OR M
1	0	0	1 AND 0 = 0	0 OR 0 = **0 (OFF — lid is open!)**
1	1	0	1 AND 1 = 1	1 OR 0 = **1 (ON — dry and lid closed)**

D (Dry)	L (Lid)	M (Manual)	Step 1: X = D AND L	Step 2: S = X OR M
0	0	1	0 AND 0 = 0	0 OR 1 = **1 (ON — manual button pressed)**

4. Discussion & Student Practice (15 min)

✎ Practice Tasks (3.1 – 3.8)

1. **Task 3.1:** Evaluate `1 AND 0`, `1 OR 0`, and `NOT 0`.
2. **Task 3.2:** Evaluate `(1 OR 0) AND (NOT 1)`. Show the intermediate value of each parentheses group first.
3. **Task 3.3:** Complete all 4 rows of the truth table for the circuit `Output = (NOT A) AND B` (where `(A, B)` takes values `(0,0)`, `(0,1)`, `(1,0)`, `(1,1)`).
4. **Task 3.4:** Complete all 4 rows of the truth table for the circuit `Output = NOT (A OR B)`.
5. **Task 3.5 (Safety Bug Analysis in Our Boolean Simulation):** In the simulated mushroom terrarium controller `S = (D AND L) OR M`, notice Row 3 of the worked example (`D=0, L=0, M=1`): if the lid is **open (`L = 0`)** and someone presses the **Manual button (`M = 1`)**, the sprayer output is still `1`! Rewrite the Boolean formula using `D`, `L`, and `M` so that in this ideal logic model the sprayer output is `1` when dry **or** when the manual button is pressed, **but ONLY if the lid input `L` is closed (`L = 1`)**.
6. **Task 3.6 (Comparing Circuits):** Compute all 4 rows of `(NOT A) AND (NOT B)` and compare the output column to Task 3.4 (`NOT (A OR B)`). What do you discover?
7. **Task 3.7 (Bitwise Masking):** Computers often apply an `AND` gate to two 4-bit numbers **column by column**. Compute `1101 AND 0110` bit by bit.
8. **Task 3.8 (Ratio of True Outputs):** Out of the `4` possible input rows in a 2-input truth table (`00, 01, 10, 11`), what is the ratio of `1` (TRUE) outputs to total rows for an `AND` gate vs. an `OR` gate?

Complete Answer Key (Tasks 3.1 – 3.8)

- **Answer 3.1:** `1 AND 0` = **0**; `1 OR 0` = **1**; `NOT 0` = **1**.
- **Answer 3.2:** First group: `(1 OR 0)` = 1. Second group: `(NOT 1)` = 0. Final gate: `1 AND 0` = **0**.
- **Answer 3.3 (`(NOT A) AND B`):**
- `(0, 0)`: `NOT 0` = 1 → `1 AND 0` = **0**
- `(0, 1)`: `NOT 0` = 1 → `1 AND 1` = **1**
- `(1, 0)`: `NOT 1` = 0 → `0 AND 0` = **0**
- `(1, 1)`: `NOT 1` = 0 → `0 AND 1` = **0** (Only row *`(0, 1)`* outputs *1*.)
- **Answer 3.4 (`NOT (A OR B)`):**
- `(0, 0)`: `0 OR 0` = 0 → `NOT 0` = **1**
- `(0, 1)`: `0 OR 1` = 1 → `NOT 1` = **0**
- `(1, 0)`: `1 OR 0` = 1 → `NOT 1` = **0**
- `(1, 1)`: `1 OR 1` = 1 → `NOT 1` = **0**
- **Answer 3.5:** Put `D OR M` inside parentheses first, then require `AND L`: **`S = (D OR M) AND L`**. In this **ideal Boolean model**, whenever `L = 0` (lid open), `(D OR M) AND 0 = 0`, so the logic output stays `0` while `L = 0`. (Note that in a real physical device, a stuck lid sensor or faulty valve could still cause a malfunction even when the Boolean formula is correct!)
- **Answer 3.6:** For `(NOT A) AND (NOT B)`: `(0,0) -> 1 AND 1 = 1`; `(0,1) -> 1 AND 0 = 0`; `(1,0) -> 0 AND 1 = 0`; `(1,1) -> 0 AND 0 = 0`. The output column (`1, 0, 0, 0`) is **identical** to `NOT (A OR B)` in Task 3.4! ("Neither A nor B is ON" means the exact same thing as "A is OFF and B is OFF".)

- **Answer 3.7:** Column by column from left to right: `1 AND 0 = 0`, `1 AND 1 = 1`, `0 AND 1 = 0`, `1 AND 0 = 0` → **`0100`**.
- **Answer 3.8:** An `AND` gate outputs `1` in **`1 : 4`** (`1` out of `4` rows, or `25%`), whereas an `OR` gate outputs `1` in **`3 : 4`** (`3` out of `4` rows, or `75%`).

Vocabulary

- **logic gate:** A physical or digital switch circuit that takes one or more binary inputs (`0` or `1`) and produces a single binary output according to a fixed rule.
- **truth table:** A complete table listing every possible combination of input bits and the resulting output bit.
- **series circuit (`AND`):** Switches wired one after another along a single path so all switches must be closed (`1`) for current to flow.
- **parallel circuit (`OR`):** Switches wired on side-by-side branches so closing any one branch (`1`) allows current to flow.

Differentiation

- **Extra support:** Have students physically write the `0` or `1` output above each gate wire on a printed circuit diagram before filling in the table row.
- **Extra challenge:** Introduce the **Exclusive OR (`XOR`)** gate, which outputs `1` when `A` and `B` are **different** (`01` or `10`), and outputs `0` when both are the same (`00` or `11`). Ask students to build `A XOR B` using only `AND`, `OR`, and `NOT` gates: `(A OR B) AND NOT (A AND B)`.

⚠ Safety

Keep floor pathways clear of backpacks and chairs during the human-circuit walkthrough.

Teacher notes

- **Common misconception ("Everyday English OR vs. Logical OR"):** In everyday conversation, "You can have soup OR salad" often means one or the other, not both (`XOR`). Emphasize that in mathematics and computing, **`OR` always means "at least one (including both)"** (`1 OR 1 = 1`).

Lesson 4: Postcards on the Wire — Packet Switching, Routing, and Throughput Rates

Time: 50 minutes

Unit: Binary, Pixels, Logic Gates, and Packet Switching | **Previous:** Lesson 3 | **Next:** Lesson 5

Goal

Simulate splitting a message into numbered packets, route packets around a failed network link, reassemble out-of-order packets using header sequence numbers, and calculate network unit rates (**packets per second** and **bits per second**).

Materials

- 6 index cards ("Packet Postcards") per group of 6 students.
- Yarn or string pieces connecting 6 student desks as a 6-node router mesh (`Node A` through `Node F`).
- Scissors or a red `"LINK DOWN"` card to simulate a cut fiber-optic cable during the activity.

Steps

✎ 1. Opening (5 min): One Giant Scroll vs. Numbered Postcards

Ask the class: - "Suppose you want to mail a 300-page book to a friend, but the postal tube can only hold 1 page at a time, and 20 other classmates are also sharing the same highway. What happens if one person reserves the entire highway for 10 minutes to send their whole book in one piece?" - Contrast **circuit reservation** (everyone else waits in line; if a tree falls across the road halfway through, the whole book fails) with **packet switching** (chopping the data into small numbered blocks called **packets** that take turns sharing the links and can detour around a broken road).

✎ 2. Main Activity (20 min): The 6-Node Classroom Router Mesh

Arrange 6 desks (`A, B, C, D, E, F`) with yarn links matching this network map:

```
   [Node B] -------- [Node D]
   /      \          /      \
[Node A]   (link)  (link)   [Node F]
   \      /          \      /
   [Node C] -------- [Node E]
```

Links available: - `A–B`, `A–C` - `B–C`, `B–D` - `C–D`, `C–E` - `D–F`, `E–F`

Step-by-Step Simulation

1. **Sender (`Node A`)** wants to send the 16-character message `"FUNGI GROW FAST!"` to **Receiver (`Node F`)**, but each packet payload can hold **at most `4` characters** (`32 bits`).
2. `Node A` splits the message into `4` packets and fills out a **Header** on each card:
3. **Packet 1:** `From: A | To: F | Packet: 1 of 4 | Payload: "FUNG"`
4. **Packet 2:** `From: A | To: F | Packet: 2 of 4 | Payload: "I GR"`
5. **Packet 3:** `From: A | To: F | Packet: 3 of 4 | Payload: "OW F"`
6. **Packet 4:** `From: A | To: F | Packet: 4 of 4 | Payload: "AST!"`
7. **Round 1 (Normal Routing — `1` Hop per Step):** Assume a synchronized step-by-step scenario where each link hop takes **1 step (1 second)** and `Node F` logs simultaneous arrivals from `Node D` before `Node E`. `Node A` sends Packet 1 along `A -> B -> D -> F` (`3` steps) and Packet 2 along `A -> C -> E -> F` (`3` steps), so Packets 1 and 2 arrive at `Node F` first (logged as `Packet 1, Packet 2`).
8. **Round 2 (Sending Packets 3 & 4 + Link Failure!):** Next, `Node A` sends **Packet 3** to `Node B` and **Packet 4** directly to `Node C`. Just as Packet 3 arrives at `Node B`, place the red `"LINK DOWN"` card on link **`B–D`**!
9. Does `Node B` throw Packet 3 away? **No!** `Node B` inspects its remaining working links, sees `B–C` is still up, and reroutes Packet 3 along `B -> C -> E -> F` (`4` hops total from `A`). Meanwhile, **Packet 4** travels directly along `A -> C -> E -> F` (`3` hops total, with no queue delay at `C` or `E` because it is 1 step ahead of rerouted Packet 3).
10. **Out-of-Order Arrival at `Node F` (in This 1-Second-per-Hop Scenario):** In this scenario where every hop takes 1 second, Packet 4 completes its 3-hop path (`A -> C -> E -> F`) one step before rerouted Packet 3 completes its 4-hop detour (`A -> B -> C -> E -> F`). (Note: On real networks, arrival order depends on link delays, bandwidth, and router queues—not hop count alone!) Thus `Node F` receives the cards in the order: **Packet 1, Packet 2, Packet 4, Packet 3** (`"FUNGI GRAST!OW F"` if read blindly!).
11. `Node F` uses the **`Packet # of 4`** header field to sort them back into `1, 2, 3, 4` → **`"FUNGI GROW FAST!"`**.

3. Connecting to Grade 6 Ratios & Unit Rates: Network Throughput (10 min)

Work through a concrete rate calculation on the board: - Suppose each packet carries **`32` bits of header** + **`32` bits of payload** = **`64` bits per packet**. - If a router link transmits **`300` packets in `6` seconds**: 1. **Packet rate (unit rate):** `300 packets ÷ 6 seconds` = **`50 packets per second`**. 2. **Bit rate (throughput):** `50 packets/second × 64 bits/packet` = **`3,200 bits per second`**. 3. **Header overhead ratio:** In every `64`-bit packet, `32` bits are header and `32` bits are payload (`32 : 64 = 1 : 2`, or `50%` header overhead). If we increase the payload to `192` bits while keeping the same `32`-bit header (`224` bits total), the header overhead drops to `32 : 224` = **`1 : 7`**!

✓ 4. Student Practice & Closing Check (15 min)

✎ Practice Tasks (4.1 – 4.8)

1. **Task 4.1:** A `20`-character message (`"MEET AT THE LIBRARY!"`) is split into packets that hold at most `4` characters per payload. How many packets are needed, and what is the exact `4`-character payload of **Packet 3 of 5**? (Count spaces as characters!)
2. **Task 4.2:** Receiver `Node F` receives five packets in the arrival order:

3. `Packet 4 of 5: "RKS!"`
4. `Packet 2 of 5: "ING "`
5. `Packet 5 of 5: " :)"`
6. `Packet 1 of 5: "ROUT"`
7. `Packet 3 of 5: "WO"` Reassemble the original message in order.
8. **Task 4.3:** What would happen at `Node F` if `Packet 3 of 5` was accidentally dropped on the floor and never arrived? How does the header `"Packet # of 5"` help `Node F` know specifically which piece to request again?
9. **Task 4.4 (Hop Count on the Network Map):** Using the 6-node map (`A–B, A–C, B–C, B–D, C–D, C–E, D–F, E–F`), what is the minimum number of link hops from `A` to `F` when all links are working? List all `3`-hop paths from `A` to `F`.
10. **Task 4.5 (Rerouting Around Failure):** If **both** link `B–D` and link `C–E` fail at the same time on the 6-node map, is there still a working path from `A` to `F`? If so, write the exact node sequence and count its hops.
11. **Task 4.6 (Unit Rate — Packets per Second):** A classroom switch forwards `420` packets in `14` seconds. What is its unit rate in **packets per second**? At that same rate, how many packets will it forward in `25` seconds?
12. **Task 4.7 (Unit Rate — Bits per Second):** Each packet on a sensor network is `80` bits long (`16` header bits + `64` payload bits). If the sensor transmits `40` packets per second, what is the total transmission rate in **bits per second**? How many of those bits per second are **actual sensor payload**?
13. **Task 4.8 (Header-to-Total Ratio):** In Task 4.7 (`16` header bits and `64` payload bits, `80` bits total per packet), write the ratio of **payload bits to total bits** in simplest whole-number form.

Complete Answer Key (Tasks 4.1 – 4.8)

- **Answer 4.1:** `20 characters ÷ 4 characters/packet =` **`5` packets**. Splitting into 4-character blocks: Packet 1 = `"MEET"`, Packet 2 = `" AT "`, **Packet 3 = `"THE "`**, Packet 4 = `"LIBR"`, Packet 5 = `"ARY!"`.
- **Answer 4.2:** Sorting by sequence number `1, 2, 3, 4, 5`: `"ROUT"` + `"ING "` + `"WO"` + `"RKS!"` + `" :)"` = **`"ROUTING WORKS! :)"`**.
- **Answer 4.3:** Because `Node F` sees packets `1, 2, 4, 5` of `5`, it immediately detects that **`Packet 3` is missing** and only needs to ask `Node A` to re-send **Packet 3**, rather than re-sending the entire message.
- **Answer 4.4:** The minimum hop count is **`3` hops**. There are **three** `3`-hop paths from `A` to `F`:
- `A -> B -> D -> F`
- `A -> C -> D -> F`
- `A -> C -> E -> F`
- **Answer 4.5: Yes!** Even with both `B–D` and `C–E` broken, `Node C` connects to `Node D` via `C–D`. So a packet can travel **`A -> C -> D -> F`** (`3` hops) or **`A -> B -> C -> D -> F`** (`4` hops).
- **Answer 4.6:** Unit rate: `420 ÷ 14 =` **`30 packets per second`**. In `25` seconds: `30 × 25 =` **`750 packets`**.
- **Answer 4.7:** Total rate: `40 packets/sec × 80 bits/packet =` **`3,200 bits per second`**. Actual payload rate: `40 packets/sec × 64 payload bits/packet =` **`2,560 payload bits per second`**.
- **Answer 4.8:** Payload to total is `64 : 80`. Dividing both by `16` gives **`4 : 5`** (`4` out of every `5` bits sent are payload).

Vocabulary

- **packet:** A small, fixed-limit chunk of data paired with a header so it can travel independently across a shared network.
- **header:** Metadata attached to the front of a packet listing its sender address, destination address, and sequence number (`Packet k of N`).
- **payload:** The actual slice of user data (text, image bits, sensor reading) carried inside a packet.
- **router:** A network node that inspects a packet's destination address and forwards it along a working link toward its destination.
- **throughput (bit rate):** The unit rate measuring how many bits pass through a network link per second (`bits per second`).

Differentiation

- **Extra support:** Give each student desk a printed sign (`Node A` through `Node F`) and have students slide the index cards along the yarn strings—or trace the routes on a shared tabletop map/diagram by pointing or dictating moves so standing, walking, or handling yarn is optional.
- **Extra challenge:** Assign different **link delays** to each yarn piece (for example, `C–D` takes `1` second, while `B–D` takes `4` seconds) and ask students to find the path with the **lowest total travel time** even when that path has more hops!

⚠ Safety

Ensure yarn links are taped across desk tops rather than stretched across floor walkways where students could trip.

Teacher notes

- **Scope note on TCP/IP:** Explain briefly that on the real Internet, **IP (Internet Protocol)** handles addressing and hop-by-hop routing (like the `From` and `To` fields on our cards), while **TCP (Transmission Control Protocol)** handles the sequence numbering (`Packet k of N`) and requesting re-transmission when a packet is lost.

Lesson 5: Catching Flipped Bits — Noise, Even Parity Checks, and End-of-Unit Assessment

Time: 50 minutes

Unit: Binary, Pixels, Logic Gates, and Packet Switching | **Previous:** Lesson 4

Goal

Add an even-parity check bit to 8-bit bytes, detect single-bit transmission errors, explain why two simultaneous bit flips in the same byte escape 1-bit parity, and complete the 5-part End-of-Unit Assessment.

Materials

- Large two-sided paper cards or tokens (`Light = 1`, `Dark = 0`), 9 cards per pair (or a printed 9-box row where students can point, dictate, or write `0` and `1` instead of handling small objects).
- Printed or board-copied End-of-Unit Assessment (20 points total; 20 minutes at the end of this 50-minute lesson, or administered in a separate quiz period).

Steps

✎ 1. Opening (5 min): What If Static Flips a `0` to a `1`?

Ask the class: - "Suppose a hospital or spacecraft sends the 8-bit number *`00000101`* (*`5`* milligrams or *`5`* degrees), and a burst of radio static on the wire flips the leftmost bit from *`0`* to *`1`* (*`10000101`* = *`133`*!). How can the receiver tell that a bit was flipped without already knowing the secret number?"

✎ 2. Main Activity (15 min): The Even-Parity Check Bit

Demonstrate how adding **one extra check bit** (a **9th bit** called the **Even Parity Bit**) catches any single-bit flip (or any odd number of bit flips):

The Even-Parity Rule

Before sending an 8-bit data byte, the sender counts how many **`1`s** are in the 8 data bits: - If the number of `1`s in the 8 data bits is already **even** (`0, 2, 4, 6, or 8`), set the 9th **Parity Bit = `0`** so the total number of `1`s across all 9 bits stays **even**. - If the number of `1`s in the 8 data bits is **odd** (`1, 3, 5, or 7`), set the 9th **Parity Bit = `1`** so the total number of `1`s across all 9 bits becomes **even**.

✎ Worked Example 1: Computing the Parity Bit

8 Data Bits	Count of 1s in Data	Even or Odd?	9th Bit (Even Parity Bit)	Complete 9-Bit Frame Sent	Total 1s Sent
01000001 ('A')	2 ones	Even	**0**	01000001 **0**	2 (Even)
01000011 ('C')	3 ones	Odd	**1**	01000011 **1**	4 (Even)

✎ Worked Example 2: Checking Incoming Frames at the Receiver

When the receiver gets a 9-bit frame, it counts the total number of 1s across **all 9 bits**: - Received Frame X: 01001001 0 → Count of 1s = 3 (**Odd!**). Since every valid frame was sent with an **even** number of 1s, an odd total signals that **an odd number of bits (1, 3, 5...) flipped** during transmission: **ERROR DETECTED — discard and request re-send!** - **Crucial limitation ("No Error Detected" ≠ Proof of Zero Errors):** What if lightning flips **two** (or 4, 6, 8) bits inside the same 9-bit frame? Changing an even count by +2, 0, or -2 keeps the total **even**, so a single parity bit **cannot detect an even number of bit flips**. Therefore, a passing (even) parity check means **no error was detected**, not absolute proof that zero bits changed!

✎ 3. Student Practice Tasks (5.1 – 5.8) (10 min)

1. **Task 5.1:** Find the Even Parity Bit (0 or 1) to append to the 8-bit data byte 00110101.
2. **Task 5.2:** Find the Even Parity Bit (0 or 1) to append to the 8-bit data byte 10110101.
3. **Task 5.3:** A receiver uses even parity and receives these four 9-bit frames. Which ones have a detected error?
4. Frame A: 01010101 0
5. Frame B: 11001010 1
6. Frame C: 00000000 0
7. Frame D: 10000000 0
8. **Task 5.4:** What if the **Parity Bit itself** (the 9th bit) gets flipped by static during transmission while the 8 data bits are untouched? Will the receiver still detect an error? Explain why.
9. **Task 5.5:** Assuming at most **one** bit flipped in a 9-bit frame, explain why a single 9th parity bit can tell you **that** a bit flipped, but cannot tell you **which** of the 9 bits flipped.
10. **Task 5.6 (2D Grid Parity Puzzle):** Arrange three 4-bit data rows into a grid with an even **Row Parity bit** at the right end of each row and an even **Column Parity bit** at the bottom of each data column:
11. Row 1: 1 0 1 0 | 0
12. Row 2: 0 1 1 0 | 1
13. Row 3: 1 1 0 0 | 0
14. Col Parity: 0 0 1 0 Assume that **at most one bit** in the entire diagram flipped during transmission, and in fact **exactly one data bit** inside the 3 × 4 data grid flipped. Check the row sums and column sums to pinpoint the exact (Row, Column) of the flipped bit.

15. **Task 5.7 (Error Rate Ratio):** Out of `500` transmitted packets, `15` packets fail their parity check and must be re-sent. Write the error rate as a simplified ratio (`failed : total`) and as a rate **per `100` packets** (percentage).
16. **Task 5.8 (Overhead Comparison):** Adding `1` parity bit to every `8` data bits creates `9` total bits per byte. What is the ratio of **check bits to data bits** (`1 : 8`) and **data bits to total bits** (`8 : 9`)?

Complete Answer Key (Tasks 5.1 – 5.8)

- **Answer 5.1:** `00110101` has `4` ones (already even) → Parity Bit = **`0`** (`00110101 0`).
- **Answer 5.2:** `10110101` has `5` ones (odd) → Parity Bit = **`1`** (`10110101 1`, making `6` ones total).
- **Answer 5.3:**
- Frame A (`01010101 0`): `4` ones (Even) → **Passes check (no error detected)**.
- Frame B (`11001010 1`): `5` ones (Odd) → **ERROR DETECTED (odd number of bit flips)**.
- Frame C (`00000000 0`): `0` ones (`0` is an even number!) → **Passes check (no error detected)**.
- Frame D (`10000000 0`): `1` one (Odd) → **ERROR DETECTED (odd number of bit flips)**.
- **Answer 5.4: Yes!** Flipping any single bit among the 9 bits — including the 9th parity bit itself — changes the total count of 1s by `+1` or `-1`, turning an even total into an odd total.
- **Answer 5.5:** Under the single-bit-flip assumption, flipping any one of the 9 bit positions changes the total parity from even to odd in the exact same way, so an odd total tells the receiver that a bit flipped (or, more generally, that an odd number of bits flipped), but not which slot (`1` through `9`) changed.
- **Answer 5.6:** Count the 1s in each row (including its right-hand Row Parity bit):
- Row 1 (`1 0 1 0 | 0`): `2` ones (Even — passes check).
- **Row 2 (`0 1 1 0 | 1`):** `3` ones (**Odd — error detected in Row 2!**).
- Row 3 (`1 1 0 0 | 0`): `2` ones (Even — passes check). Now count the 1s in each data column (including its bottom Column Parity bit):
- Column 1 (`1, 0, 1 | 0`): `2` ones (Even — passes check).
- Column 2 (`0, 1, 1 | 0`): `2` ones (Even — passes check).
- **Column 3 (`1, 1, 0 | 1`):** `3` ones (**Odd — error detected in Column 3!**).
- Column 4 (`0, 0, 0 | 0`): `0` ones (Even — passes check). Because Row 2 and Column 3 both fail their even-parity check, the single flipped bit lies at their intersection: **`(Row 2, Column 3)`** (changing that `1` back to `0` restores both Row 2 and Column 3 to `2` ones).
- **Answer 5.7:** `15 : 500` simplifies (dividing by `5`) to **`3 : 100`**, which is **`3` failed packets per `100` packets (`3%`)**.
- **Answer 5.8:** Check bits to data bits = **`1 : 8`**; data bits to total transmitted bits = **`8 : 9`**.

End-of-Unit Synthesis Assessment (20 Points Total — 20 Minutes)

Instructions for Students

Answer all 5 problems (`4` points each = `20` points total). Show your place-value additions, intermediate gate values, or unit-rate steps clearly.

✎ Problem 1: Doubling Cards & 8-Bit Binary (4 points)

- **(a) [1 pt]** List the 8 binary place-value cards in order from left (`8`th bit) to right (`1`st bit).

- **(b) [1 pt]** Convert the 8-bit binary byte `01011010` to a base-10 number.
- **(c) [2 pts]** Convert the base-10 number **`150`** into an 8-bit binary byte. Show which place-value cards are switched ON (`1`).

✎ Problem 2: ASCII Characters & Bitmap Pixel Ratios (4 points)

- **(a) [2 pts]** In standard ASCII, uppercase `'A'` is `65` (`01000001`). Decode the 2-byte word `01001000 01001001`.
- **(b) [2 pts]** A small `10 × 10` game icon (`100` pixels total) is upgraded from a 2-color palette (`1 bit per pixel`) to an 8-color palette (`3 bits per pixel`). Counting **raw uncompressed pixel data** (excluding any headers), how many **bits** did the icon take before the upgrade, how many **bits** does it take after the upgrade, and what is the ratio of old raw pixel size to new raw pixel size?

✎ Problem 3: Logic Gates & Truth Table (4 points)

A solar battery charger turns ON (`Output = 1`) according to the logic circuit: `Output = S AND (NOT F)` where **`S`** = `1` when Sunlight is detected (`0` when dark), and **`F`** = `1` when the Battery is already Full (`0` when not full). - **(a) [3 pts]** Write out the 4-row truth table for `(S, F)` = `(0,0)`, `(0,1)`, `(1,0)`, `(1,1)`, showing the intermediate column `NOT F` and the final `Output` column. - **(b) [1 pt]** Explain in plain English which single row turns the charger ON and why.

✎ Problem 4: Packet Switching & Network Unit Rates (4 points)

- **(a) [2 pts]** Three packets arrive at a receiver in the order:
- `Packet 3 of 3: "ONE!"`
- `Packet 1 of 3: "WELL"`
- `Packet 2 of 3: " D"` Write the reassembled message and explain why packets from the same message can arrive out of order on a packet-switched network.
- **(b) [2 pts]** A network link transfers `450` packets in `9` seconds. Each packet is `100` bits long (`20` header bits + `80` payload bits). Find the link's rate in **packets per second** and its total throughput in **bits per second**.

✎ Problem 5: Even-Parity Error Detection (4 points)

- **(a) [1 pt]** What Even Parity Bit (`0` or `1`) should be attached to the 8-bit byte `11010011`?
- **(b) [2 pts]** A receiver using even parity receives Frame 1 (`10101010 0`) and Frame 2 (`11100011 0`). Which frame has a detected error? Show your count of `1`s for each frame.
- **(c) [1 pt]** Explain why even parity fails to detect an error if exactly **two** bits flip inside the same 9-bit frame.

End-of-Unit Assessment Answer Key & Scoring Guide (20 Points)

- **Problem 1 (4 pts):**
- **(a) [1 pt]:** `128, 64, 32, 16, 8, 4, 2, 1`.

- **(b) [1 pt]:** `64 + 16 + 8 + 2 =` **`90`**.
- **(c) [2 pts]:** `150 = 128 + 16 + 4 + 2` → **`10010110`** (1 pt for correct card decomposition, 1 pt for exact 8-bit string).
- **Problem 2 (4 pts):**
- **(a) [2 pts]:** Byte 1 (`01001000`) = `64 + 8 = 72` (8th letter = `'H'`). Byte 2 (`01001001`) = `64 + 8 + 1 = 73` (9th letter = `'I'`). Decoded word: **`"HI"`**.
- **(b) [2 pts]:** Before upgrade: `100 pixels × 1 bit/pixel = 100 bits`. After upgrade: `100 pixels × 3 bits/pixel = 300 bits`. Ratio of old to new storage = **`1 : 3`**.
- **Problem 3 (4 pts):**
- **(a) [3 pts]:**
  - `(S=0, F=0)` → `NOT F = 1` → `0 AND 1 =` **`0`**
  - `(S=0, F=1)` → `NOT F = 0` → `0 AND 0 =` **`0`**
  - `(S=1, F=0)` → `NOT F = 1` → `1 AND 1 =` **`1`**
  - `(S=1, F=1)` → `NOT F = 0` → `1 AND 0 =` **`0`**
- **(b) [1 pt]:** Row `(S=1, F=0)` turns the charger ON (`1`) because there is sunlight (`S=1`) **and** the battery is **not** full (`F=0`, so `NOT F = 1`).
- **Problem 4 (4 pts):**
- **(a) [2 pts]:** Reassembled message (`1, 2, 3`): **`"WELL DONE!"`**. Packets can arrive out of order because different packets may take different router paths (with different hop counts or congestion delays) around busy or broken links.
- **(b) [2 pts]:** Packet rate: `450 packets ÷ 9 seconds =` **`50 packets per second`**. Total throughput: `50 packets/sec × 100 bits/packet =` **`5,000 bits per second`**.
- **Problem 5 (4 pts):**
- **(a) [1 pt]:** `11010011` has `5` ones (odd) → Even Parity Bit = **`1`**.
- **(b) [2 pts]:** Frame 1 (`10101010 0`) has `4` ones (even → **passes check, no error detected**). Frame 2 (`11100011 0`) has `5` ones (odd → **ERROR DETECTED in Frame 2**).
- **(c) [1 pt]:** Flipping two bits changes the number of `1`s by `+2`, `0`, or `-2`, which keeps an even total **even**, so the receiver sees an even count and cannot tell that two bits changed.

Vocabulary

- **noise / interference:** Unwanted electrical or radio disturbances that can accidentally flip a `0` bit to a `1` (or `1` to `0`) during transmission.
- **even parity bit:** An extra check bit (`0` or `1`) appended to a group of data bits so the total number of `1`s across the entire frame is always an even number.

Differentiation

- **Extra support:** Allow students to keep their 8 physical place-value cards (`128, 64, 32, 16, 8, 4, 2, 1`) on their desk during the assessment.
- **Extra challenge:** Have students complete the 2D `3 × 4` grid parity puzzle (Task 5.6) where they intentionally flip one data coin while their partner looks away, and see if their partner can find the exact flipped coin in one try using only row and column parity!

⚠ Safety

No physical hazards in this unplugged lesson.

Teacher notes

- **Zero-is-even reminder:** Remind students before Task 5.3 that **`0` is an even number**, so an all-zero byte `00000000` already has an even count of `1`s (`0` ones) and receives a `0` parity bit (`00000000 0`).

Unit: Data Structures, Graph Algorithms, and Shortest Paths (Computing, Grade 10)

> **Unreviewed AI draft.** Written by Gemini 4 Argon (2026-11-05). No human teacher has reviewed this unit. All content is CC0 1.0 Public Domain.

Overview

Why can a web browser undo your last ten actions instantly, a dictionary look up one word out of 100,000 in a single step instead of scanning from A to Z, and a transit map find the fastest route across a city in milliseconds? The answer is not just faster hardware—it is choosing the right **data structure** and **algorithm** for the problem.

Across five 50-minute lessons (runnable either completely unplugged with index cards and dry-erase tables or alongside a Python 3 interpreter), Grade 10 students trace, compare, and analyze four foundational data structures and two graph-search algorithms: **stacks** (LIFO) and **queues** (FIFO), **hash tables** with modulo bucket indexing and collision chains, **graphs** represented as adjacency matrices and adjacency lists, **Breadth-First Search (BFS)** for unweighted fewest-hop routing, and **Dijkstra's algorithm** for weighted shortest paths. Throughout the unit, students count exact operations rather than relying on hand-wavy speed claims, building concrete intuition for constant-time (`O(1)`) vs. linear-time (`O(n)`) cost.

Prior knowledge

- Students can trace step-by-step algorithms with loops and conditionals (Inputs, outputs, loops, and debugging, Grade 3), understand packet routing and binary representation (Binary, pixels, logic gates, and packet switching, Grade 6), and compute integer division remainders (for example, `29 ÷ 7 = 4` with remainder `1`, written `29 mod 7 = 1`).
- Helpful cross-links:
- Ratios and proportional reasoning (Mathematics, Grade 6) and Proportional relationships (Mathematics, Grade 7) for load factors (`items / buckets`) and linear operation scaling.
- Ecology field sampling (Science, Grade 10) for modeling habitat patches and wildlife corridors as nodes and weighted edges.
- Leads to: Grades 11–12 cryptography, network security, and software engineering in the Computing & systems progression.

Learning goals

By the end of this unit, a student can:

1. Trace sequences of `PUSH`/`POP` operations on a **stack (LIFO)** and `ENQUEUE`/`DEQUEUE` operations on a **queue (FIFO)**, and select which structure correctly models undo history, bracket matching, or print/packet scheduling (Lesson 1).
2. Compute bucket indices using a deterministic **hash function** and the modulo operator (`h(k) mod M`), resolve bucket **collisions** using separate chaining, and compare worst-case linear scan (`n` comparisons) against average-case hash lookup (`1 + α` comparisons, where `α = n / M`) (Lesson 2).

3. Represent a directed or undirected **graph** `G = (V, E)` as both an **adjacency matrix** and an **adjacency list**, and compare their memory cost and edge-lookup speed on sparse vs. dense networks (Lesson 3).
4. Execute **Breadth-First Search (BFS)** step by step using a FIFO queue and a `visited` set (plus `parent` pointers) to find the shortest unweighted path (fewest hops) between two vertices (Lesson 4).
5. Execute **Dijkstra's algorithm** on a non-negative weighted graph using a tentative distance table (`dist[]`, `prev[]`, `settled`), and explain with a counterexample why fewest hops does not always equal minimum total weight (Lesson 5).

Lesson sequence

Lesson	Title	Time	Core idea
Lesson 1	Who Goes Next? Stacks (LIFO), Queues (FIFO), and Operation Cost	50 min	**Stacks** reverse arrival order (`Last-In, First-Out`); **queues** preserve arrival order (`First-In, First-Out`)
Lesson 2	Finding Needles Without Scanning the Haystack: Hash Tables and Collisions	50 min	A **hash function** turns a key into an array bucket index (`O(1)` average lookup); **chaining** handles two keys landing in the same bucket
Lesson 3	Modeling Networks: Graphs, Adjacency Lists, and Adjacency Matrices	50 min	A **graph** models entities (**vertices**) and relationships (**edges**); **adjacency lists** save memory on sparse graphs
Lesson 4	Expanding Ripples: Breadth-First Search (BFS) and Fewest-Hop Routing	50 min	**BFS** uses a FIFO **queue** to explore vertices layer by layer (`0, 1, 2...` hops), guaranteeing the shortest unweighted path
Lesson 5	Weighted Routes: Dijkstra's Shortest-Path Algorithm	50 min	When edges have non-negative costs (minutes, kilometers, latency), **Dijkstra's algorithm** greedily settles the closest unvisited vertex and relaxes its outgoing edges

Materials for the whole unit

- Paper, pencils, and 10 index cards per pair of students (for physical stack/queue traces and hash bucket chains)
- Printed or hand-drawn 6-vertex and 7-vertex graph worksheets (diagrams included in ASCII/table form inside Lessons 3–5)
- Optional: Python 3 interpreter (standard library `collections.deque` and `heapq`) if running in a computer lab

Assessment summary

Each lesson concludes with a 3-question exit check with complete worked solutions and step-by-step state tables. Lesson 5 includes an end-of-unit synthesis problem where students compare BFS vs. Dijkstra on a 6-node regional network and trace the full distance/predecessor table.

Sources

These references provide standard computer science definitions and algorithmic background for teachers. All links were checked to load on 2026-11-05.

- Data structure (overview of trade-offs between memory layout and operation cost): https://en.wikipedia.org/wiki/Data_structure — background for Lessons 1–3.
- Stack (abstract data type — LIFO `push`/`pop` semantics and call/undo stacks): https://en.wikipedia.org/wiki/Stack_(abstract_data_type) — background for Lesson 1.
- Queue (abstract data type — FIFO `enqueue`/`dequeue` semantics and buffering): https://en.wikipedia.org/wiki/Queue_(abstract_data_type) — background for Lessons 1 and 4.
- Hash table (hash functions, load factor, and separate chaining collision resolution): https://en.wikipedia.org/wiki/Hash_table — background for Lesson 2.
- Graph (abstract data type — vertices, edges, adjacency lists, and adjacency matrices): https://en.wikipedia.org/wiki/Graph_(abstract_data_type) — background for Lesson 3.
- Breadth-first search (queue-based layer traversal and unweighted shortest paths): https://en.wikipedia.org/wiki/Breadth-first_search — background for Lesson 4.
- Dijkstra, E. W. (1959), A note on two problems in connexion with graphs, via Wikipedia summary of Dijkstra's algorithm: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm — background for Lesson 5.

Known gaps and cautions

- **Unreviewed AI draft:** This unit has not yet been reviewed by a human high-school computer science educator or classroom-tested with Grade 10 students. (Peer-checked by Claude Haiku 5.5 on Lessons 2 and 5, and by Claude Fable 5.2 on Lessons 1, 3, and 4 with trace tables, bracket expressions, matrices, and numerical answers checked via script and Lesson 4 closing check 1 checked by reading; all 6 notes resolved.)
- **Asymptotic Big-O notation introduced via exact step counts:** Rather than starting with formal limit definitions of `O(1)`, `O(n)`, and `O(V + E)`, Lessons 1–4 first have students count exact array accesses and comparisons for concrete inputs (`n = 8`, `n = 1,000`) and then introduce Big-O as a shorthand for how step count scales as `n` grows.
- **Toy vs. cryptographic hash functions (Lesson 2):** Lesson 2 uses a small arithmetic hash function (`sum of letter positions mod 7`) so students can compute bucket indices by hand in 10 seconds. Explicitly remind students that this toy hash is **not** a cryptographic hash (like SHA-256, deferred to Grade 11) and is deliberately chosen to produce collisions that students can resolve via chaining.
- **Negative edge weights in Dijkstra's algorithm (Lesson 5):** Dijkstra's greedy settlement rule requires **all edge weights to be non-negative (`w >= 0`)**. Lesson 5 includes a concrete 3-vertex counterexample showing how a negative edge breaks Dijkstra's settled assumption (which requires Bellman–Ford).

Lesson 1: Who Goes Next? — Stacks (LIFO), Queues (FIFO), and Operation Cost

- **Unit:** Data Structures, Graph Algorithms, and Shortest Paths (Computing, Grade 10)
- **Time:** 50 minutes
- **Grouping:** Whole class and pairs

Learning objective

Students will trace `PUSH` / `POP` / `PEEK` operations on a **stack (LIFO)** and `ENQUEUE` / `DEQUEUE` operations on a **queue (FIFO)**, count the exact steps required for each operation, and use a stack to verify whether a string of nested brackets is balanced.

Materials

- 10 index cards per pair of students
- Whiteboard or projection of the operation trace tables below

Vocabulary

- **Data Structure:** A deliberate way of organizing data in memory so specific operations (inserting, removing, searching) are fast and predictable.
- **Stack (LIFO — Last-In, First-Out):** A linear collection where items are added (`PUSH`), removed (`POP`), and inspected without removal (`PEEK`) at the **same end** (the `top`).
- **Queue (FIFO — First-In, First-Out):** A linear collection where items are added (`ENQUEUE`) at the `back` (tail) and removed (`DEQUEUE`) from the `front` (head).
- **Constant Time (`O(1)`):** An operation whose step count stays the same whether the collection holds 5 items or 5,000,000 items.

Lesson flow

✎ 1. Opening: Two Ways to Handle a Pile of Tasks (8 min)

1. Contrast two everyday situations:
2. **Situation A (Cafeteria Tray Dispenser / Text Editor Undo):** You type `"cat"`, then `"dog"`, then `"bird"`. When you press `Ctrl+Z` (**Undo**), which word should disappear first? (`"bird"`—the **last** action entered must be the **first** action undone!)
3. **Situation B (Cafeteria Lunch Line / Network Router Packet Buffer):** Three packets `P1`, `P2`, `P3` arrive at a router (Grade 6 Computing) in that order. Which packet should leave the router first? (`P1`—the **first** packet in line must be the **first** packet sent!)
4. Introduce the names: **Situation A** is a **Stack (LIFO)**; **Situation B** is a **Queue (FIFO)**.

2. Direct instruction: Tracing Stack vs. Queue State Tables (15 min)

Walk through the exact same arrival sequence (`A`, then `B`, then `C`, followed by two removals, then `D`, then one removal) on both structures:

Step	Stack Command	Stack State (`bottom → top`)	Returned Value	Queue Command	Queue State (`front → back`)	Returned Value
1	`PUSH(A)`	`[A]`	—	`ENQUEUE(A)`	`[A]`	—
2	`PUSH(B)`	`[A, B]`	—	`ENQUEUE(B)`	`[A, B]`	—
3	`PUSH(C)`	`[A, B, C]`	—	`ENQUEUE(C)`	`[A, B, C]`	—
4	`POP()`	`[A, B]`	**`C`**	`DEQUEUE()`	`[B, C]`	**`A`**
5	`POP()`	`[A]`	**`B`**	`DEQUEUE()`	`[C]`	**`B`**
6	`PUSH(D)`	`[A, D]`	—	`ENQUEUE(D)`	`[C, D]`	—
7	`POP()`	`[A]`	**`D`**	`DEQUEUE()`	`[D]`	**`C`**

Highlight the output order: - The **Stack** popped **`C, B, D`** (always removing the newest item on top). - The **Queue** dequeued **`A, B, C`** (always removing the oldest item waiting at the front).

✎ 3. Main activity: Bracket-Matching Compiler Check with a Stack (17 min)

1. Explain how code editors and compilers check whether parentheses, square brackets, and curly braces—`()`, `[]`, `{}`—are properly nested:
2. **Algorithm:** Scan the string left to right:
    1. If you see an **opening bracket** (`(`, `[`, or `{`), **`PUSH`** it onto the stack.
    2. If you see a **closing bracket** (`)`, `]`, or `}`):
        - If the stack is **empty**, fail immediately (**Unmatched closing bracket**).
        - Otherwise, **`POP`** the top opening bracket and check if it matches the closing bracket's type. If not, fail (**Mismatched bracket type**).
    3. At the end of the string, if the stack is **empty**, return **`VALID`**; if any brackets remain on the stack, fail (**Unclosed opening bracket**).
3. Student pairs use their index cards as a physical stack to trace three expressions and record the exact step where invalid strings fail:
4. **Expression 1:** `{ [ ( ) ] ( ) }` → Stack ends empty → **`VALID`**.
5. **Expression 2:** `[ ( ] )` → Push `[`, Push `(`, next character is `]`, pop `(` → mismatch (`(` does not match `]`) at character 3 → **`INVALID`**.
6. **Expression 3:** `( [ { } ]` → Ends with `(` still on the stack → **`INVALID`** **(unclosed `()`**.

4. Discussion & Operation Cost Check (5 min)

- **Question:** "If an array holds *`1,000,000`* items in slots *`0`* through *`999,999`*, why is *`PUSH`* / *`POP`* at the **end** of the array (*`slot 999,999`*) a 1-step *`O(1)`* operation, whereas removing from *`slot 0`* and sliding all *`999,999`* remaining items one slot left takes *`999,999`* steps (*`O(n)`*)?"
- Example good answer: Adding or removing at the end only touches one slot (`top`), whereas shifting every item left touches every remaining element in the array. (To make a Queue `O(1)` for both `ENQUEUE` and `DEQUEUE`, languages use a circular buffer with a moving `front` index or a doubly linked list like Python's `collections.deque`.)

✓ 5. Closing check (5 min)

1. Start with an empty stack `S`. Execute: `PUSH(10)`, `PUSH(20)`, `POP()`, `PUSH(30)`, `PUSH(40)`, `POP()`. What value is returned by the second `POP()`, and what items remain in `S` from bottom to top?

2. Start with an empty queue `Q`. Execute: `ENQUEUE(10)`, `ENQUEUE(20)`, `DEQUEUE()`, `ENQUEUE(30)`, `ENQUEUE(40)`, `DEQUEUE()`. What value is returned by the second `DEQUEUE()`, and what items remain in `Q` from front to back?
3. Trace the bracket-matching algorithm on `( [ ) ]`. What is on the stack right before the first closing bracket `)` is read, and why does the check fail?

Worked answers for closing check

1. First `POP()` removes `20` (leaving `[10]`). Then `30` and `40` are pushed (`[10, 30, 40]`). The second `POP()` returns **`40`**, leaving **`[10, 30]`** (`10` at bottom, `30` at top).
2. First `DEQUEUE()` removes `10` (leaving `[20]`). Then `30` and `40` are enqueued (`[20, 30, 40]`). The second `DEQUEUE()` returns **`20`**, leaving **`[30, 40]`** (`30` at front, `40` at back).
3. Right before `)` is read, the stack holds **`[ '(', '[' ]`** (with `'['` on top). When `)` is read, `POP()` returns `'['`, which does not match `')'`, so the algorithm immediately reports a mismatch failure.

Differentiation

- **Extra support:** Have students write the index number (`1, 2, 3...`) on each card as it enters and physically stack cards vertically for a Stack vs. slide cards horizontally through a paper tunnel for a Queue.
- **Extra challenge:** Show how a browser's **Back** and **Forward** buttons use **two stacks** (`back_stack` and `forward_stack`): clicking **Back** pops the current page from `back_stack` and pushes it onto `forward_stack`, while visiting a brand-new link clears `forward_stack`.

Teacher notes

- **Common misconception:** Students sometimes think a stack "sorts" items by size. Emphasize that stacks and queues care **only about arrival time**, never about whether `10` is smaller or larger than `40`.

Lesson 2: Finding Needles Without Scanning the Haystack — Hash Tables and Collisions

- **Unit:** Data Structures, Graph Algorithms, and Shortest Paths (Computing, Grade 10)
- **Time:** 50 minutes
- **Grouping:** Whole class and pairs

Learning objective

Students will compute array bucket indices using a deterministic **hash function** (`h(k) mod M`), build a 7-bucket **hash table** with **separate chaining** to resolve collisions, and compare the number of comparisons needed for an unsorted list scan vs. a hash table lookup.

Materials

- Printed or whiteboard 7-bucket hash table grid (`Buckets 0 through 6`)
- Alphabet position strip (`A=1, B=2, C=3, ..., Z=26`) for quick arithmetic hashing by hand

Vocabulary

- **Key–Value Pair:** A unique identifier (**key**, such as a student ID or word) stored alongside its associated data (**value**, such as a grade or definition).
- **Hash Function (`h(k)`):** A deterministic math rule that converts any key `k` into an integer bucket index `0, 1, ..., M - 1`.
- **Collision:** When two different keys (`k1 != k2`) produce the **exact same bucket index** (`h(k1) == h(k2)`).
- **Separate Chaining:** Storing a short linked list (chain) of key–value pairs inside each bucket so colliding keys can share a bucket without overwriting each other.
- **Load Factor (`α = n / M`):** The ratio of stored items (`n`) to total buckets (`M`), connecting directly to Grade 6 Mathematics: Ratios.

Lesson flow

✎ 1. Opening: Why Not Just Scan a List? (8 min)

1. Suppose a school library database stores `n = 70,000` book barcodes in an unsorted list.
2. To check if barcode `B-94821` is checked out, a **linear scan** checks item 1, item 2, item 3... taking `35,000` comparisons on average and `70,000` comparisons in the worst case (`O(n)`).
3. Ask: "What if, instead of searching every shelf, we had a math formula that looked at the letters of a key and told us the exact shelf number in 1 step?"
4. That is a **Hash Table** (called a `dict` in Python or `Map` in JavaScript).

2. Direct instruction: Computing a Toy Hash Function & Modulo Indexing (15 min)

1. Define a simple, hand-computable hash rule for a table with `M = 7` **buckets** numbered `0` through `6`:
2. **Step A:** Add the alphabet positions (`A=1, B=2, ..., Z=26`) of the letters in the key (`sum`).
3. **Step B:** Divide `sum` by `7` and take the **remainder**: `bucket = sum mod 7` (the remainder is always `0, 1, 2, 3, 4, 5, or 6`!).
4. Work through 6 animal keys together on the board:

Key	Letter Values	sum	Division by 7	bucket = sum mod 7
"CAT"	3 + 1 + 20	24	24 = 3 × 7 + 3	**Bucket 3**
"DOG"	4 + 15 + 7	26	26 = 3 × 7 + 5	**Bucket 5**
"ANT"	1 + 14 + 20	35	35 = 5 × 7 + 0	**Bucket 0**
"OWL"	15 + 23 + 12	50	50 = 7 × 7 + 1	**Bucket 1**
"BEE"	2 + 5 + 5	12	12 = 1 × 7 + 5	**Bucket 5 (COLLISION with "DOG"!)**
"FOX"	6 + 15 + 24	45	45 = 6 × 7 + 3	**Bucket 3 (COLLISION with "CAT"!)**

1. Show why **Separate Chaining** is essential:
2. When `"BEE"` lands in **Bucket `5`**, we must **not** erase `"DOG"`. Instead, Bucket `5` holds a short chain: `Bucket 5 → [ ("DOG", val), ("BEE", val) ]`.
3. When looking up `"BEE"`, the computer computes `12 mod 7 = 5`, jumps straight to Bucket `5`, checks `"DOG"` (not a match), and finds `"BEE"` on the 2nd comparison—still only **2 comparisons** instead of scanning all 6 items!

✎ 3. Main activity: Pair Hash-Table Build & Worst-Case Adversarial Attack (17 min)

1. **Part A — Insert 3 More Keys (10 min):** Student pairs draw the 7-bucket table (`0..6`) with the 6 animals above already placed, then compute and insert three more keys:
2. `"ELK"` (`5 + 12 + 11 = 28`; `28 mod 7 = 0` → chains after `"ANT"` in **Bucket `0`**)
3. `"HEN"` (`8 + 5 + 14 = 27`; `27 mod 7 = 6` → lands in empty **Bucket `6`**)
4. `"COD"` (`3 + 15 + 4 = 22`; `22 mod 7 = 1` → chains after `"OWL"` in **Bucket `1`**)
5. **Part B — Count Comparisons & Load Factor (7 min):**
6. Total items `n = 9`, total buckets `M = 7` → **Load factor `α = 9 / 7 ≈ 1.29` items per bucket**.
7. What is the maximum number of key comparisons needed to find any of the 9 animals in our hash table? (Only **2 comparisons**, because the longest chain has length 2!)
8. **Adversarial thought experiment:** Why doesn't our hash function only check the first letter of a word? (Because in an English dictionary, thousands of words start with `'S'`, piling into one giant chain while `'X'` is nearly empty!)

✓ 4. Discussion questions (5 min)

- **Question 1:** "Why must a hash function be 100% deterministic (never using a random number generator each time you look up a key)?"

- Example good answer: If `h("CAT")` returned Bucket `3` when inserting and a random Bucket `6` when looking it up later, you would look in the wrong bucket and never find `"CAT"` again!
- **Question 2:** "What happens to lookup speed if we put $n = 7{,}000$ items into only $M = 7$ buckets ($\alpha = 1{,}000$)? How do real hash tables prevent that?"
- Example good answer: Each bucket chain would average `1,000` items long, slowing lookups back down to linear scans. Real hash tables **resize (rehash)**—doubling `M` whenever the load factor `α = n / M` passes a threshold like `0.75`.

✓ 5. Closing check (5 min)

Using the same rule (`letter sum mod 7`, with `A=1 ... Z=26`):

1. Compute the letter sum and bucket index (`0..6`) for the key **`"BAT"`** (`B=2, A=1, T=20`).
2. Does **`"BAT"`** collide with any key in our 9-animal table from the lesson? If so, which bucket does it join, and which keys are already in that bucket?
3. A hash table has `M = 50` buckets and currently stores `n = 35` keys. What is its **load factor** **`α`**, and why is a low load factor desirable?

Worked answers for closing check

1. `B(2) + A(1) + T(20) = 23`. Dividing by 7 gives `23 = 3 × 7 + 2`, so `23 mod 7 = 2` (**Bucket `2`**).
2. **No collision!** In our 9-animal table, Buckets `0, 1, 3, 5, 6` had items, while **Bucket `2`** and **Bucket `4`** were empty. So `"BAT"` is the first item in Bucket `2`.
3. **Load factor `α = 35 / 50 = 0.70`**. A low load factor means most buckets have 0 or 1 item, keeping lookups close to 1 comparison (`O(1)`).

Differentiation

- **Extra support:** Provide a printed multiplication table for `7` (`0, 7, 14, 21, 28, 35, 42, 49, 56, 63, 70`) so students can find `sum mod 7` immediately by subtracting the largest multiple of `7` at or below `sum`.
- **Extra challenge:** Have students find an **anagram collision** (such as `"ACT"`, `"CAT"`, and `"TAC"`): why do all anagrams collide under an addition-only hash function, and how does multiplying each letter by a position weight (`1×L1 + 31×L2 + 31²×L3`) break anagram collisions?

Teacher notes

- **Common misconception:** Students often assume a hash collision means the hash table "failed" or overwrote the earlier key. Emphasize that collisions are normal and expected (by the Pigeonhole Principle / Birthday Paradox), and that separate chaining stores both keys safely in the same bucket list.

Lesson 3: Modeling Networks — Graphs, Adjacency Lists, and Adjacency Matrices

- **Unit:** Data Structures, Graph Algorithms, and Shortest Paths (Computing, Grade 10)
- **Time:** 50 minutes
- **Grouping:** Whole class and pairs

Learning objective

Students will model real-world systems as **graphs** `G = (V, E)` (choosing directed vs. undirected and weighted vs. unweighted edges), convert a graph between its **adjacency matrix** and **adjacency list** representations, and compare their memory and lookup trade-offs.

Materials

- Grid paper or printed worksheets for drawing a 5×5 adjacency matrix and adjacency list

Vocabulary

- **Graph (`G = (V, E)`):** A mathematical structure made of a set of **vertices** (nodes, `V`) and a set of **edges** (connections, `E`) linking pairs of vertices.
- **Directed vs. Undirected Edge:** An **undirected** edge `{u, v}` is a two-way connection (`u ↔ v`); a **directed** edge `(u → v)` is a one-way arrow from `u` to `v`.
- **Degree of a Vertex:** The number of edges connected to that vertex (in a directed graph, split into **in-degree** and **out-degree**).
- **Adjacency Matrix:** A `|V| × |V|` grid where row `u`, column `v` stores `1` (or edge weight `w`) if edge `(u, v)` exists, and `0` otherwise.
- **Adjacency List:** A dictionary/array where each vertex `u` stores a list of only its direct neighbors.

Lesson flow

✎ 1. Opening: Four Different Worlds, One Same Shape (8 min)

1. Put four scenarios on the board and ask what they have in common:
2. **Packet routers** connected by fiber-optic cables (Grade 6 Computing)
3. **Meadow habitat patches** connected by wildlife movement corridors (Grade 10 Science: Ecology Field Sampling)
4. **Web pages** connected by clickable hyperlinks (Grade 8 Society: Information Literacy)
5. **Cities** connected by rail lines
6. In every case, we have **nodes (vertices)** and **connections (edges)**. Once we represent any of these as a **graph**, the exact same algorithms can analyze all four!
7. Ask: "Which of those four has **one-way (directed)** edges by default?"
8. **Web hyperlinks!** Page `A` can link to Page `B` (`A → B`) without Page `B` linking back to Page `A`.

2. Direct instruction: Adjacency Matrix vs. Adjacency List (15 min)

Consider this 5-vertex **undirected** regional network with vertices `V = {A, B, C, D, E}` and 6 edges: `E = { {A,B}, {A,C}, {B,C}, {B,D}, {C,D}, {D,E} }`:

```
   A --- B
   |   / |
   |  /  |
   C --- D --- E
```

Build both computer representations side by side:

Representation 1: 5×5 Adjacency Matrix (`|V|² = 25` cells)

From \ To	A	B	C	D	E	Row Sum (Degree)
A	0	1	1	0	0	2
B	1	0	1	1	0	3
C	1	1	0	1	0	3
D	0	1	1	0	1	3
E	0	0	0	1	0	1

- Notice that for an **undirected** graph, the matrix is **symmetric across the main diagonal** (`matrix[u][v] == matrix[v][u]`), and the sum of all `1`s in the matrix (`12`) equals `2 × |E|` (the Handshaking Lemma: every undirected edge adds `1` to the degree of both its endpoints!).

Representation 2: Adjacency List (`|V| + 2|E|` entries)

```
graph = {
    "A": ["B", "C"],
    "B": ["A", "C", "D"],
    "C": ["A", "B", "D"],
    "D": ["B", "C", "E"],
    "E": ["D"]
}
```

✎ 3. Main activity: Sparse vs. Dense Trade-Off Analysis & Directed Conversion (17 min)

1. **Part A — Directed Graph Practice (9 min):** Pairs are given a 4-page mini-web of directed hyperlinks:
2. Page `1` links to `2` and `3` (`1 → 2`, `1 → 3`)
3. Page `2` links to `3` (`2 → 3`)
4. Page `3` links to `1` and `4` (`3 → 1`, `3 → 4`)
5. Page `4` links to `2` (`4 → 2`) Students write:
6. The 4×4 **directed adjacency matrix** (and verify it is **not** symmetric: `matrix[1][2] = 1`, `matrix[2][1] = 0`).
7. The **in-degree** (column sum = incoming links) and **out-degree** (row sum = outgoing links) of every page.
8. **Part B — Memory Calculation for a Real Road Map (8 min):**
9. Suppose a country road map has `|V| = 100,000` intersections, and each intersection meets an average of `4` roads (`200,000` undirected edges).

10. **Adjacency Matrix size:** `100,000 × 100,000 = 10,000,000,000` (`10 billion` cells—over `99.996%` of them filled with `0`s!).
11. **Adjacency List size:** `100,000` vertex buckets holding `2|E| = 400,000` neighbor entries (`25,000×` fewer neighbor entries than the matrix, or `|V| + 2|E| = 500,000` total entries including the vertex headers—`20,000×` smaller overall!).

✓ 4. Discussion questions (5 min)

- **Question 1:** "When is an Adjacency Matrix actually better than an Adjacency List?"
- Example good answer: When the graph is small or **dense** (nearly every vertex connects to every other vertex), or when you frequently need to check "Does edge *(u, v)* exist?" in a single array lookup `matrix[u][v]` (`O(1)`).
- **Question 2:** "Why is the sum of the degrees of all vertices in any undirected graph always an **even number**?"
- Example good answer: Because every single edge connects two endpoints, adding `+2` to the total degree sum (`Sum of degrees = 2 × |E|`).

✓ 5. Closing check (5 min)

1. An undirected graph has `|V| = 6` vertices and `|E| = 8` edges. How many total cells are in its **adjacency matrix**, how many of those cells contain a `1`, and how many contain a `0`?
2. Look at this adjacency list for a **directed** graph:
3. `X: [Y, Z]`
4. `Y: [Z]`
5. `Z: []` What are the **out-degree** and **in-degree** of vertex `Z`?
6. Would you use an **adjacency matrix** or an **adjacency list** to store a social network with `1,000,000` users where the average user has `150` connections? Explain why using numbers.

Worked answers for closing check

1. A `6 × 6` adjacency matrix has **`36` total cells**. Since each of the `8` undirected edges places two `1`s in the matrix (`matrix[u][v]` and `matrix[v][u]`), **`16` cells contain `1`** and `36 − 16 = 20` **cells contain `0`**.
2. Vertex `Z` has **out-degree `0`** (its outgoing list `Z: []` is empty) and **in-degree `2`** (both `X` and `Y` point to `Z`).
3. An **adjacency list**, because it stores only `1,000,000 × 150 = 150,000,000` connection entries (~1.5×10^8), whereas an adjacency matrix would require `1,000,000 × 1,000,000 = 1,000,000,000,000` (10^{12}, one trillion) cells—nearly all zeros.

Differentiation

- **Extra support:** Have students color-code each undirected edge `{u, v}` with a matching pair of highlighters in the matrix (`row u, col v` and `row v, col u`) so the diagonal symmetry and the `2 × |E|` count become visually immediate.

- **Extra challenge:** Ask students how the adjacency matrix and adjacency list change if we allow **self-loops** (an edge from a vertex to itself, like a web page linking to itself) and **parallel edges** (two separate roads between the same two towns).

Teacher notes

- **Common misconception:** When reading a directed adjacency matrix, students frequently swap row (`from`) and column (`to`). Keep the header label **`From (Row) \ To (Column)`** visible on the board so out-degree is always the row sum and in-degree is always the column sum.

Lesson 4: Expanding Ripples — Breadth-First Search (BFS) and Fewest-Hop Routing

- **Unit:** Data Structures, Graph Algorithms, and Shortest Paths (Computing, Grade 10)
- **Time:** 50 minutes
- **Grouping:** Whole class and pairs

Learning objective

Students will execute **Breadth-First Search (BFS)** step by step on an unweighted graph using a FIFO **queue**, a `visited` set, a `dist` (hop count) table, and `parent` pointers to reconstruct the shortest unweighted path between two vertices.

Materials

- Printed 6-vertex network diagram (below) and index cards to serve as the FIFO queue

Vocabulary

- **Breadth-First Search (BFS):** A graph traversal algorithm that explores all neighbors at distance `1` hop from the start vertex before exploring vertices at distance `2` hops, `3` hops, and so on.
- **Visited Set:** A record of vertices already discovered (enqueued) so the algorithm never gets trapped in an infinite loop around a cycle.
- **Parent (Predecessor) Pointer:** For each discovered vertex `v`, the vertex `u` that first discovered `v` (`parent[v] = u`), allowing us to trace the shortest path backward from the goal to the start.

Lesson flow

✎ 1. Opening: Dropping a Stone in a Pond (7 min)

1. Consider this 6-router network (`A` through `F`) where every link takes 1 hop: `A --- B --- E | | | C --- D --- F` Edges (alphabetical neighbor order):
2. `A: [B, C]`
3. `B: [A, D, E]`
4. `C: [A, D]`
5. `D: [B, C, F]`
6. `E: [B, F]`
7. `F: [D, E]`
8. Ask: "If router *A* wants to send a packet to router *F* in the **fewest hops**, why can't a computer just wander down the first road it sees (*A → C → D → B → E → F*, 5 hops)?"
9. Explain the ripple idea: If we explore in expanding rings—**Ring 0** (`A`), **Ring 1** (all 1-hop neighbors of `A`), **Ring 2** (all 2-hop neighbors), **Ring 3** (all 3-hop neighbors)—then the very **first time** a ring touches `F`, we are guaranteed to have found a path with the minimum possible number of hops!

10. Ask: "Which data structure from Lesson 1 processes items in exact First-In, First-Out order so Ring 1 finishes before Ring 2?" → **A Queue (FIFO)!**

2. Direct instruction: Complete Step-by-Step BFS Trace from `A` to `F` (16 min)

Show the exact BFS algorithm and trace table (breaking neighbor ties alphabetically):

```
from collections import deque

def bfs_shortest_path(graph, start, goal):
    queue = deque([start])
    visited = {start}
    dist = {start: 0}
    parent = {start: None}

    while queue:
        u = queue.popleft()          # DEQUEUE front vertex
        if u == goal:
            break
        for v in graph[u]:           # Inspect each neighbor of u
            if v not in visited:
                visited.add(v)       # Mark visited IMMEDIATELY upon enqueue!
                dist[v] = dist[u] + 1
                parent[v] = u
                queue.append(v)      # ENQUEUE v at back
    return dist, parent
```

Trace every iteration on the board:

Step	Dequeued `u`	Unvisited Neighbors Enqueued	Queue (`front → back`)	`visited` Set	Newly Recorded `dist[v]` & `parent[v]`
Init	—	Start at `A`	`[A]`	`{A}`	`dist[A]=0`, `parent[A]=None`
1	**`A`**	`B`, `C`	`[B, C]`	`{A, B, C}`	`dist[B]=1 (p=A)`, `dist[C]=1 (p=A)`
2	**`B`**	`D`, `E` (`A` already in `visited`)	`[C, D, E]`	`{A, B, C, D, E}`	`dist[D]=2 (p=B)`, `dist[E]=2 (p=B)`
3	**`C`**	(none: *`A`* & *`D`* already in *`visited`*!)	`[D, E]`	`{A, B, C, D, E}`	—
4	**`D`**	`F` (`B` & `C` already in `visited`)	`[E, F]`	`{A, B, C, D, E, F}`	`dist[F]=3 (p=D)`
5	**`E`**	(none: *`B`* & *`F`* already in *`visited`*)	`[F]`	`{A, B, C, D, E, F}`	—
6	**`F`**	Goal reached!	`[]`	`{A, B, C, D, E, F}`	Stop

Reconstructing the path via `parent` pointers: Start at `F`: `parent[F] = D` → `parent[D] = B` → `parent[B] = A` → `parent[A] = None`. Reverse it: **`A → B → D → F`** (3 hops!). (Note: `A → B → E → F` and `A → C → D → F` are also tied at `3` hops; alphabetical tie-breaking discovered `D` via `B` first, and `F` via `D` first.)

> **Critical bug to highlight:** Look at Step 2 and Step 3. Why did we mark `D` as `visited` the moment `B` **enqueued** `D` in Step 2, instead of waiting until `D` was dequeued in Step

4? Because in Step 3 (`u = C`), `C` also has an edge to `D`! If `D` were not already in `visited`, `C` would enqueue a duplicate copy of `D` into the queue. Always mark vertices `visited` **at enqueue time**.

✎ 3. Main activity: Pair BFS Trace on a 7-Node Campus Graph (17 min)

Student pairs trace BFS starting from `S` to find the hop distance and `parent` tree to all vertices on this 7-vertex graph (alphabetical tie-breaking): - `S: [A, B]` - `A: [S, C, D]` - `B: [S, D]` - `C: [A, G]` - `D: [A, B, E, G]` - `E: [D]` - `G: [C, D]`

Pairs fill in the complete `dist` and `parent` table and verify: - Layer 0: `S` (`dist=0`) - Layer 1: `A` (`dist=1, p=S`), `B` (`dist=1, p=S`) - Layer 2: `C` (`dist=2, p=A`), `D` (`dist=2, p=A`) - Layer 3: `G` (`dist=3, p=C`), `E` (`dist=3, p=D`) - Shortest path from `S` to `G`: **`S → A → C → G`** (3 hops).

✓ 4. Discussion questions (5 min)

- **Question 1:** "What happens to our algorithm if we replace the FIFO **Queue** (*`popleft()`*) with a LIFO **Stack** (*`pop()`*)?"
- Example good answer: It explores in a **depth-first** style instead of expanding in layer rings (`1 hop, 2 hops, 3 hops`), plunging deep down one branch before backtracking. Because it does not finish Layer 1 and Layer 2 before exploring deeper nodes, the first path it records is **not** guaranteed to be the shortest path (for example, on our 7-node campus graph, popping from the end visits `S, B, D, G, C` and records a `4`-hop path `S → B → D → G → C` to `C` instead of the `2`-hop shortest path `S → A → C`; on the 6-router network from `A`, it records a `4`-hop path `A → C → D → F → E` to `E` instead of `2` hops `A → B → E`).
- **Question 2:** "How many times does BFS enqueue each vertex and inspect each edge in a connected graph *`G = (V, E)`*?"
- Example good answer: Each vertex enters the queue at most **once** (`|V|` dequeues), and each edge is inspected at most **twice** (once from each endpoint), giving a linear time cost of **`O(|V| + |E|)`**.

✓ 5. Closing check (5 min)

1. In the 7-vertex campus graph (`S → A, B` above), both `A` and `B` have an edge to `D`. Why did `parent[D]` get set to `A` instead of `B`?
2. What would happen if you ran BFS on a graph containing a triangle cycle `A – B – C – A` **without** keeping a `visited` set?
3. After running BFS from `Start = M`, your `parent` table records: `parent[M] = None`, `parent[P] = M`, `parent[Q] = P`, `parent[R] = P`, `parent[T] = R`. What is the shortest path from `M` to `T`, and how many hops long is it?

Worked answers for closing check

1. Because `A` was enqueued before `B` (alphabetical order) and therefore dequeued first. When `A`'s neighbors were scanned, `D` was unvisited, so `D` was marked visited with `parent[D] = A`. When `B` was dequeued next, `D` was already in `visited`.
2. Without a `visited` set, vertices are enqueued repeatedly around the cycle (`A` enqueues `B, C`; `B` enqueues `A, C`; `C` enqueues `A, B`, and so on). If the code stops at a reachable `goal` (such as

`goal = C` from `A`), it still halts once `C` is dequeued (after 3 dequeues), but leaves duplicate entries in the queue and wastes exponential work on larger graphs; if we are exploring the whole graph (or the `goal` is unreachable), the queue grows forever and the algorithm **never terminates**.

3. Tracing backward from `T`: `T ← R ← P ← M`. Reversing gives **`M → P → R → T`**, which is **`3` hops** long.

Differentiation

- **Extra support:** Give pairs physical tokens or sticky notes labeled `"VISITED"` to place directly on each vertex the moment its card enters the queue, reinforcing that marking happens at **enqueue** time rather than dequeue time.
- **Extra challenge:** Have students trace true recursive or late-marked **Depth-First Search (DFS)** (where a vertex is marked visited when popped rather than when pushed) alongside our enqueue-marked stack variant on the 7-vertex campus graph and compare the two visitation orders.

Teacher notes

- **Common misconception:** Students often forget to reverse the `parent` pointer chain at the end (`F → D → B → A` vs. `A → B → D → F`). Remind them that `parent[v]` points **backward** toward the start vertex, just like breadcrumbs leading back home.

Lesson 5: Weighted Routes — Dijkstra's Shortest-Path Algorithm

- **Unit:** Data Structures, Graph Algorithms, and Shortest Paths (Computing, Grade 10)
- **Time:** 50 minutes
- **Grouping:** Whole class and pairs

Learning objective

Students will explain why fewest hops (BFS) can fail on a **weighted graph**, execute **Dijkstra's shortest-path algorithm** step by step using a tentative distance table (`dist[]`, `prev[]`, `settled`), and explain why negative edge weights violate Dijkstra's greedy settlement rule.

Materials

- Printed weighted graph trace sheets (diagrams below) and pencils

Vocabulary

- **Weighted Graph:** A graph where each edge `(u, v)` has a numerical **weight (cost)** `w(u, v)` representing distance, travel time, cable cost, or network latency.
- **Tentative Distance (`dist[v]`):** The shortest known total cost from the start vertex to `v` found so far (initialized to `0` for `start` and `∞` for all other vertices).
- **Edge Relaxation:** Checking whether going from `start` through `u` to neighbor `v` (`dist[u] + w(u, v)`) beats the current best known cost `dist[v]`; if `dist[u] + w(u, v) < dist[v]`, we update `dist[v]` and set `prev[v] = u`.
- **Settled (Finalized) Vertex:** A vertex whose minimum distance from `start` is permanently locked in.

Lesson flow

✎ 1. Opening: When Fewest Hops Is the Slowest Route (7 min)

1. Draw a 3-node triangle on the board representing travel times in minutes between `A`, `B`, and `C`:
2. Direct highway edge `A — C` has weight **`10` minutes** (congested construction zone, `1` hop).
3. Local road `A — B` has weight **`2` minutes**, and `B — C` has weight **`3` minutes** (`2` hops total).
4. Ask:
5. "Which route from *`A`* to *`C`* does **BFS (Lesson 4)** pick?" → `A → C` (`1` hop, **10 minutes**), because BFS ignores edge weights and only counts hops!
6. "Which route is actually faster?" → `A → B → C` (`2 + 3 =` **5 minutes**!).
7. Introduce Edsger W. Dijkstra's 1959 insight: When all edge weights are **non-negative (`w >= 0`)**, if we always pick the **unsettled vertex `u` with the smallest tentative `dist[u]`**, no detour through any other unsettled vertex can ever make `dist[u]` smaller!

2. Direct instruction: Complete Step-by-Step Trace of Dijkstra's Algorithm (16 min)

Consider this 5-vertex weighted network (weights in milliseconds of network latency): - `A — B` (weight `4`), `A — C` (weight `1`) - `C — B` (weight `2`), `C — D` (weight `5`) - `B — D` (weight `1`), `B — E` (weight `7`) - `D — E` (weight `3`)

```
       (4)       (7)
   A ------- B ------- E
   |       / |       /
(1)|  (2)/   |(1)   /(3)
   |   /     |     /
   C ------- D ---/
       (5)
```

Dijkstra's Algorithm Rules: 1. Set `dist[A] = 0` and `dist[v] = ∞` for all other vertices. Set `settled = {}`. 2. While unsettled vertices remain: - **Pick `u`:** Choose the **unsettled** vertex with the **minimum `dist[u]`** and mark `u` as **settled**. - **Relax edges out of `u`:** For each unsettled neighbor `v` of `u`, compute `new_cost = dist[u] + w(u, v)`. If `new_cost < dist[v]`, update `dist[v] = new_cost` and `prev[v] = u`.

Trace every iteration on the board:

Round	Settled `u` (`dist[u]`)	Edge Relaxations Tested from `u`	`dist[A]`	`dist[B]` (prev)	`dist[C]` (prev)	`dist[D]` (prev)	`dist[E]` (prev)	Settled Set
Init	—	—	`0`	`∞`	`∞`	`∞`	`∞`	`{}`
1	**`A` (0)**	`A→B: 0+4=4 < ∞` ✓ `A→C: 0+1=1 < ∞` ✓	**`0`**	`4 (A)`	`1 (A)`	`∞`	`∞`	`{A}`
2	**`C` (1)**	`C→B: 1+2=3 < 4` ✓ **(improved!)** `C→D: 1+5=6 < ∞` ✓	`0`	**`3 (C)`**	**`1 (A)`**	`6 (C)`	`∞`	`{A, C}`
3	**`B` (3)**	`B→D: 3+1=4 < 6` ✓ **(improved!)** `B→E: 3+7=10 < ∞` ✓	`0`	`3 (C)`	`1 (A)`	**`4 (B)`**	`10 (B)`	`{A, C, B}`
4	**`D` (4)**	`D→E: 4+3=7 < 10` ✓ **(improved!)**	`0`	`3 (C)`	`1 (A)`	`4 (B)`	**`7 (D)`**	`{A, C, B, D}`
5	**`E` (7)**	(all neighbors settled)	`0`	`3 (C)`	`1 (A)`	`4 (B)`	**`7 (D)`**	`{A, C, B, D, E}`

Reconstruct the minimum-weight path from `A` to `E`: Follow `prev` backward from `E`: `E ← D ← B ← C ← A`. Reverse it: **`A → C → B → D → E`** with total cost `1 + 2 + 1 + 3 = 7 ms`! Notice that `A → B → E` (2 hops) costs `4 + 7 = 11 ms`, and `A → C → D → E` (3 hops) costs `1 + 5 + 3 = 9 ms`, while the **4-hop path `A → C → B → D → E` costs only `7 ms`**!

✎ 3. Main activity: Pair Dijkstra Trace & The Negative-Weight Counterexample (17 min)

1. **Part A — Pair Dijkstra Trace (12 min):** Pairs trace Dijkstra's algorithm from `S` to `T` on this 5-vertex graph:
2. `S — A` (6), `S — B` (2)

3. `B — A (3)`, `B — C (8)`
4. `A — C (1)`, `A — T (9)`
5. `C — T (4)` Students fill out the 5-round table and verify:
6. Round 1 settles `S (0)` → `dist[A]=6 (S)`, `dist[B]=2 (S)`
7. Round 2 settles `B (2)` → `B→A` relaxes `dist[A]` from `6` down to `2+3 = 5 (B)`; `B→C` sets `dist[C] = 2+8 = 10 (B)`
8. Round 3 settles `A (5)` → `A→C` relaxes `dist[C]` from `10` down to `5+1 = 6 (A)`; `A→T` sets `dist[T] = 5+9 = 14 (A)`
9. Round 4 settles `C (6)` → `C→T` relaxes `dist[T]` from `14` down to `6+4 = 10 (C)`
10. Round 5 settles `T (10)` → Shortest path is **`S → B → A → C → T`** with cost **`10`**.
11. **Part B — Why Negative Edges Break Dijkstra (5 min):** Draw a 3-vertex directed graph:
12. `X → Y` has weight `5`
13. `X → Z` has weight `2`
14. `Y → Z` has weight **`-6`** (a negative edge!)
15. Trace what Dijkstra does from `X`:
    - Round 1 settles `X (0)` → `dist[Z] = 2`, `dist[Y] = 5`.
    - Round 2 greedily settles **`Z (2)`** because `2 < 5`!
    - But going `X → Y → Z` actually costs `5 + (-6) = -1 < 2`! Because `w(Y, Z) = -6` was negative, Dijkstra's assumption that "adding more edges can never reduce cost" failed.

✓ 4. Discussion questions (5 min)

- **Question 1:** "What happens when you run Dijkstra's algorithm on a graph where every single edge has weight *`w = 1`*?"
- Example good answer: It behaves identically to **Breadth-First Search (BFS)**! BFS is simply the special case of Dijkstra's algorithm when all edge weights are equal to `1`.
- **Question 2:** "In our Lesson 5 network (*`A → C → B → D → E`*), *`dist[B]`* was first set to *`4`* in Round 1, and then lowered to *`3`* in Round 2. Why was it legal to change *`dist[B]`* in Round 2?"
- Example good answer: Because `B` was still **unsettled** in Round 2! A vertex's distance is only tentative until that vertex itself is chosen as the minimum unsettled vertex `u`.

✓ 5. End-of-unit synthesis & Closing check (5 min)

1. In the Part A trace (`S, A, B, C, T`), why did Dijkstra settle vertex `B` (`dist = 2`) before vertex `A` (`dist = 6`) in Round 2?
2. Give a concrete 3-vertex example (`P, Q, R` with positive edge weights) where the **1-hop path `P → R`** is longer than the **2-hop path `P → Q → R`**.
3. Which data structure or algorithm from this 5-lesson unit (`Stack`, `Queue`, `Hash Table`, `BFS`, or `Dijkstra`) should an engineer choose for each task below?
4. **(a)** Looking up a user's profile by their unique username `"alex_42"` in `O(1)` average time.
5. **(b)** Finding the fastest driving route across a highway network where each road segment is labeled with its travel time in minutes.
6. **(c)** Implementing the `Undo` button in a drawing program.

7. **(d)** Finding the minimum number of friend-of-a-friend introductions connecting two people in an unweighted social graph.

Worked answers for closing check

1. Because at the start of Round 2, the unsettled vertices had tentative distances `dist[B] = 2`, `dist[A] = 6`, `dist[C] = ∞`, and `dist[T] = ∞`. Dijkstra always selects the unsettled vertex with the **smallest tentative distance** (`2 < 6`).
2. Any triangle where `w(P, Q) + w(Q, R) < w(P, R)`, such as `w(P, R) = 15`, `w(P, Q) = 4`, and `w(Q, R) = 5` (`4 + 5 = 9 < 15`).
3. **(a) Hash Table**; **(b) Dijkstra's Algorithm**; **(c) Stack (LIFO)**; **(d) Breadth-First Search (BFS)** (using a FIFO **Queue**).

Differentiation

- **Extra support:** Have students cross out settled vertices with a check mark (✓) in the table header at the end of each round so they only compare tentative distances among the remaining unsettled columns.
- **Extra challenge:** Ask students how A (A-Star) Search* speeds up Dijkstra's algorithm on a physical map by adding an admissible heuristic estimate `h(v)` (such as straight-line Euclidean distance to the goal) to the priority key `dist[v] + h(v)`.

Teacher notes

- **Common misconception:** Students sometimes try to add the new edge weight `w(u, v)` to the old `dist[v]` instead of computing `dist[u] + w(u, v)`. Write the relaxation question on the board in words: "Is going through *u* (*dist[u] + edge*) shorter than the best route to *v* we knew so far (*dist[v]*)?"