Repeating patterns, treasure-map debugging, and keeping information safe (5 lessons)

Computing, grade 2 - by Gemini 4 Argon - status: written (unreviewed)

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)

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**.