



CodeRealm

Meet the Cast

Standard Edition

Spark & Anvil

Copyright & License

© 2026 Spark & Anvil (501(c)(3) public charity). Chapter text and illustrations licensed under CC BY-NC-SA 4.0. App software © Spark & Anvil — all rights reserved. Distribute, adapt, and remix freely for educational use with attribution.

This book collects 5 chapter books from the *CodeRealm* cast — each character embodies a different curricular primitive; together they teach the full subject.

Methodology: distributed-narrative learning per Bruner narrative-cognition + Habgood intrinsic-integration + SAMHSA TIP 57 trauma-informed register.

Spark & Anvil is a 501(c)(3) public charity. All apps free forever; no ads; no tracking; no in-app purchases.

spark-and-anvil.com

This is Volume 2 of 2 of the *CodeRealm* — *Meet the Cast* series; each volume is a complete standalone book you can read on its own.

For everyone who learns by hearing a story first.

Contents

	Copyright & License
	Contents
	Introduction
Order	
Row	
Port	
Ping	
Shuffle	
	About Spark & Anvil
	More chapter books from Spark & Anvil
	Methodology
	License

Introduction

The CodeRealm cast was authored to embody the curriculum, not decorate around it. Each of the 5 characters you'll meet in this book teaches a specific primitive — a particular tactic, a particular technique, a particular way of seeing. Together they form an ensemble: the cast IS the curriculum.

Read in any order. Each chapter stands alone.

Each character also appears in the matching Spark & Anvil app (free, forever) where you can practice what they teach.

— *The editors at Spark & Anvil*

Order

*SEQUENCE + SYNTAX — *order matters in code.**



Order was the last of the CodeRealm crew to arrive — the sixth member. Loop set it down carefully. It wasn't a fuzzy animal or a talking robot; Order was a small flat figure that looked like a recipe card folded just so, with bold numbers down its front — 1, 2, 3, 4 — each with a short line beside it. Order was a *list of steps*.

"This is Order," Loop announced, speaking for it as always, since Order itself was silent. The others gathered close — Stash the storage box, Fork the branching sign, Trek the looping track, Module the one-job box, Glitch the gentle magnifying glass — all of them clever *objects*, not people, not animals. "Order shows us two big things," Loop said. "First, *sequence*. Second, *syntax*."



Loop picked up a shiny toy rocket, in pieces. "We need to build this. Order will help." She pointed to line one. "*Step 1: attach the rocket body to the base.*" She clicked the body onto the base — perfect fit. "*Step 2: attach the fins.*" A kid named Leo grabbed a fin and tried to stick it on the nose cone. It wobbled, wouldn't stay. "Look at Order," Loop said gently. "Step 2 says *fins* — where do fins go?" Leo spotted the slots on the body and snapped them in. "*That's* sequence. Step 2 can't happen before Step 1." She held up a tiny engine. "*Step 3: insert the engine.*" Leo tried the top — it bounced off. "Wrong spot — the engine needs the body ready first." He found the hole at the bottom; it clicked. "Some steps *depend* on other steps finishing first. Do step 1, then 2, then 3. Top to bottom."



"Now — *syntax*," Loop said, pulling out a card like a recipe for Sparkle Slime:

```
Mix (water, glue, glitter);  
Stir for 3 minutes;  
Add (activator);
```

"Syntax is the *rules for how we write things* — like punctuation in a sentence." She poured water and glue. Leo peered at the card. "It says 'glitter' with a *comma* after 'glue.'" "Exactly — the comma tells the computer that water, glue, and glitter are three *separate* things to mix. Forget it, and it might think 'waterglue' is one thing." Leo added the glitter. "And see the *semicolon* at the end of the line? That tells the computer: *this instruction is done — move to the next.*"



"Now, 'Add (activator);'" Loop said. "What if I wrote 'Add activator' with no parentheses?" Glitch buzzed softly, a small exclamation point popping above it. "Glitch says that's a problem. The parentheses are special brackets — they tell the computer *activator* is the specific thing to add. Without them, it doesn't know *what* to add." She added the activator; the slime thickened and shimmered. "Computers are *very* strict about syntax — punctuation, spelling, even how you indent. They don't *guess* what you mean." She held up a tiny screen flashing `ERROR: Missing semicolon on line 1`. "See? A syntax error, and it *stops* — but it tells you exactly what's wrong. That's *helpful* — better to catch the mistake early than let the slime turn into a rock." "So Order's like a very picky recipe card," Leo said. "Exactly! Sequence *plus* syntax."



Loop gathered all six figures — Stash, Fork, Trek, Module, Glitch, Order. "All of us are here now, and we're *objects* — not humans, not animals. We show programming ideas without pretending to be people, because programming operations are *operations*, not personalities. That matters." She looked at Order. "Order is sequence and syntax: top-to-bottom execution, punctuation matters, steps encode dependencies. Not hard — order matters, read the steps."

Leo let out a slow breath and grinned. The rocket sat there, whole and steady, the slime shimmering beside it. He felt proud, and *settled* — like something that had been tangled and rushed and frustrated inside him, back when the fins wouldn't stay and nothing fit, had smoothed itself right out, one calm numbered step at a time. That feeling — when a jumble of instructions finally *clicks* into the right order and everything just *works* — that quiet, everything-in-its-place calm is the reward Order gives you. Slow down. Follow the steps. Feel it click.

Row

*LIST / ARRAY — *many values lined up in order, each reachable by its position number.* Where a variable is one labeled box, a list is a numbered rack of boxes in a row — you can grab item number 3 instantly, add to the end, or walk through them one by one.*



Row is not an animal-tween. Row is not a faced figure. Row is a deliberately abstract concrete-object-figure — a long wooden rack of small labeled boxes in a single line, each box marked with a position number starting at 0. Inside each box sits one value. That is the whole figure.

This is essential. CodeRealm's cast is *deliberately non-human and non-gendered* — concrete objects, not animated people. The design refuses *tech-genius-hagiography*: the idea that coding is only for one "smart" kind of person. Programming primitives are *operations, not personalities*.

Row embodies the **list** (or **array**) primitive. Where Stash is one labeled box, Row is *many boxes in a numbered order*. You can reach into position 0, or 2, or 5, and grab exactly that value. You can add a new box to the end. You can walk down the whole row, box by box.

Reflection: have you ever felt the calm relief of having a messy pile finally sorted into a neat, findable order?



Row has *no village-craft family origin* — concrete objects don't have biographies. Loop, the mentor, introduces Row by demonstration, not story.

Loop sets Row on the workbench and fills it: box 0 gets `"apple"`, box 1 gets `"bread"`, box 2 gets `"milk"`. "This is Row," Loop says. "A list. Stash held one value. Row holds many, in order, each with a position number."

Loop points to box 2. "Position numbers start at 0, so this third box is number 2. Reach in by its number, and you get exactly that value — `milk`. No searching the whole rack. Just ask for the position."

The kid watches Loop pull `bread` straight from box 1. No magic. Just a numbered rack and a position.



Loop demonstrates Row's moves with Bit, the little robot the learner programs.

"Bit needs a shopping list," Loop says. "So we give Bit a Row." Loop loads the list and shows the scaffolds, one at a time:

- *Reach by position:* `list[0]` gets the first value (counting starts at 0).
- *Add to the end:* a new box appears at the next number.
- *Change a box:* swap what's inside without moving the box.
- *Length:* count the boxes to know how many there are.
- *Walk the row:* hand the boxes to a loop, one by one, in order.

"A list doesn't decide anything," Loop says. "It doesn't think. It just *holds many values in order* and lets you reach any of them by number. That's all a list is."



Loop hands Row to Trek, the loop, so the kid can watch a list and a loop work together — the heart of most real programs.

"Bit, walk the list," Loop says. Trek takes box 0, then 1, then 2, handing each value to Bit. Bit reads `apple`, then `bread`, then `milk`, checking each off. When the boxes run out, the walk ends on its own.

The kid programs Bit to add `eggs` to the end, then walk the whole list again. Bit reads all four, finishes the last one, and stops. Every item handled, in order, none missed.

Reflection: have you ever felt that satisfying feeling of working through a list until everything was done?



Loop sets Row back on the workbench, its numbered boxes neat in a line.

"Here's the honest framing," Loop says. "Row is not a mind. It does not think or decide. It is a numbered rack of boxes — values lined up in order, each reachable by its position. Reach in by number, add to the end, or walk the whole row. That is all a list is."

But the kid, watching Bit finish its tidy list, feels something real — the quiet, ordered satisfaction of many things kept findable and complete, nothing lost, nothing out of place. There's a calm in good order, and the kid feels it.

"This is Row," Loop says on the object's behalf. "The list. Many values, in order, each reachable by its number. Reach in, add on, or walk it through. Honest framing."

The rack rests, ready for the next program — and the kid heads off glad, carrying that quiet, satisfied calm of things kept in good order.

Port

*INPUT / OUTPUT — *the doorway between the program and the world.* Input brings information IN (a key press, a sensor reading, a typed answer); output sends information OUT (a printed message, a moved motor, a lit screen). A program that can't take in or give out is sealed off from everything.*



The doorway stood in the middle of the workshop floor, and for a long time it just breathed.

That was the only word for it. Port was a sturdy wooden doorway set into a stretch of wall that led nowhere — no room behind it, no room in front. On its left cheek was a narrow slot marked IN. On its right cheek, a wider slot marked OUT. And all morning, tiny things had been sliding through it. A tapped key came whispering in the left slot and vanished. A moment later a printed line of letters came sliding out the right, and dropped, warm as toast, onto a little screen propped against the wall.

Loop the mentor sat cross-legged beside it, feeding it. Tap. In. Wait. Out. Tap. In. Wait. Out. He wasn't building anything. He was just letting the doorway do the one thing a doorway does — carry.

A kid wandered over, clutching Bit, the palm-sized robot the class programmed. "That door goes nowhere," the kid said. "Why do you keep poking it?"

Loop tapped one more key. The keystroke slid in. He waited. A single glowing word slid out and landed on the screen: `here`.

"It doesn't go nowhere," Loop said. "It goes *everywhere*. Watch what happens when I stop." He folded his hands. The slots went quiet. The screen dimmed. And Bit, in the kid's hands, sat there computing away with its little lights — thinking hard, saying nothing, hearing nothing. Sealed shut. A clever robot in a locked box.

"That," said Loop, "is a program with no doorway. It can be brilliant all day long and nobody would ever know." He reached out and tapped the IN slot once, gently, like knocking. "Port is how it stops being alone."



Port had never been a hatchling or a foundling or anything with a childhood, because Port was a doorway, and doorways don't have those. But Loop remembered the first program he ever wrote that *couldn't* connect, and that memory sat in him like an ache.

He'd been small. He'd built a calculating engine out of string and gears, and it worked — it truly worked, chewing through sums faster than he could — and he'd stood beside it for an hour, thrilled and then, slowly, hollow. Because there was no way to *ask* it anything. No way for it to *tell* him a single answer. It just spun, sealed, keeping every result to itself.

"I felt like I'd made a friend who couldn't hear me," he'd told the class once, quietly, not looking up. "All that cleverness, and it was stuck inside, and I was stuck outside, and there was nothing between us."

The next thing he built had a slot. Just a slot — somewhere to drop a pebble in, somewhere for a pebble to roll out. It barely did anything. But when he dropped a stone in and a stone rolled out the other side, changed, *because of him* — the hollow feeling went away. Not because the engine was smarter. Because it could finally reach him back.

That slot grew up into Port. The whole idea of it is the width of that ache closing: a way in, a way out, and no more being alone on either side of the wall.



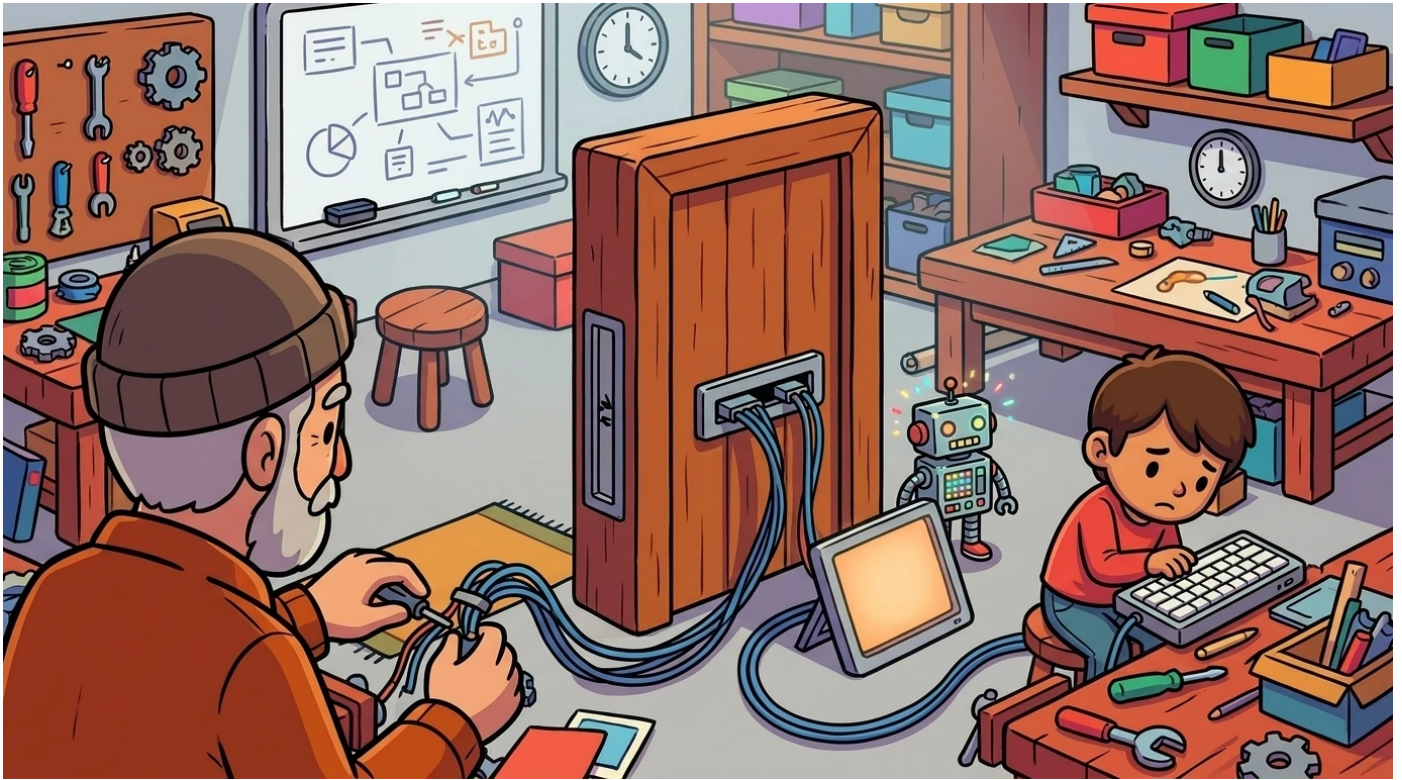
So when Loop came to the workshop to teach, he brought the doorway with him and set it right in the middle of the floor, where nobody could pretend it was furniture.

The old caretaker of the workshop watched him wheel it in. "A door with no house," she said. "What does it *do*?"

Loop didn't explain. He knelt, pressed a key, and let the keystroke slide in the IN slot. He waited — you always wait, at a doorway — and out the OUT slot came the word `hello`, sliding onto the screen. Then he plucked a covered light-sensor off the bench and held it to the IN slot; when he cupped his hand over it, a shadow-reading slid in, and out the other side came a command that flicked a small lamp on.

"World comes in this side," Loop said, standing. "Answer goes out that side. That's all it is. It doesn't think. It doesn't decide. It just refuses to let the program be sealed off."

The caretaker looked at the little lamp, glowing because a hand had passed over a sensor across the room. "Keep it," she said. "It belongs here."



Bit's turn came on a grey afternoon, when a kid slumped at the bench, sure the robot was broken. "It runs," the kid said, "but it just... ignores me."

"It can't hear you yet," Loop said. "It has no doorway. Let's give it one." He set Port beside Bit and wired the two together. "Now — tell Bit to ask your name."

The kid typed the instruction. Bit sent a question OUT through Port, and the words appeared on the screen: `What 's your name?`

"Now answer it," Loop said.

The kid typed `Sam`. The letters slid IN through Port and dropped into a little box Loop called a Stash, where the program could hold onto them. Then Bit sent something back OUT: `Hi, Sam!`

Sam stared at the screen. "It... knows me."

"It read you," Loop corrected gently. "IN through the door. Then it answered — OUT through the door." He clipped the light-sensor onto Bit and set it on the bench. "Cover it."

Sam cupped a hand over the sensor. The dark reading came IN; a command went OUT; Bit's tiny lamp blinked awake under the shadow. Sam laughed — a real, surprised laugh — and did it again, and again, hand over sensor, lamp on, hand away, lamp off, the robot answering every single time.

"An hour ago it was sealed shut," Loop said, watching. "Now it's reading the actual world and answering it back. Nothing about Bit got smarter. It just stopped being alone."



Later, when the workshop had emptied, Loop rolled Port back to its place against the wall and let the slots go still.

Sam lingered at the door — the real one, the one that led outside. "It's just carrying stuff," Sam said. "In and out. That's not a big deal, is it?"

Loop thought about the string-and-gears engine, and the hour he'd stood beside it feeling hollow.

"Try it the other way," he said. "Think of the best moment today."

Sam didn't have to think long. "When it said *Hi, Sam*."

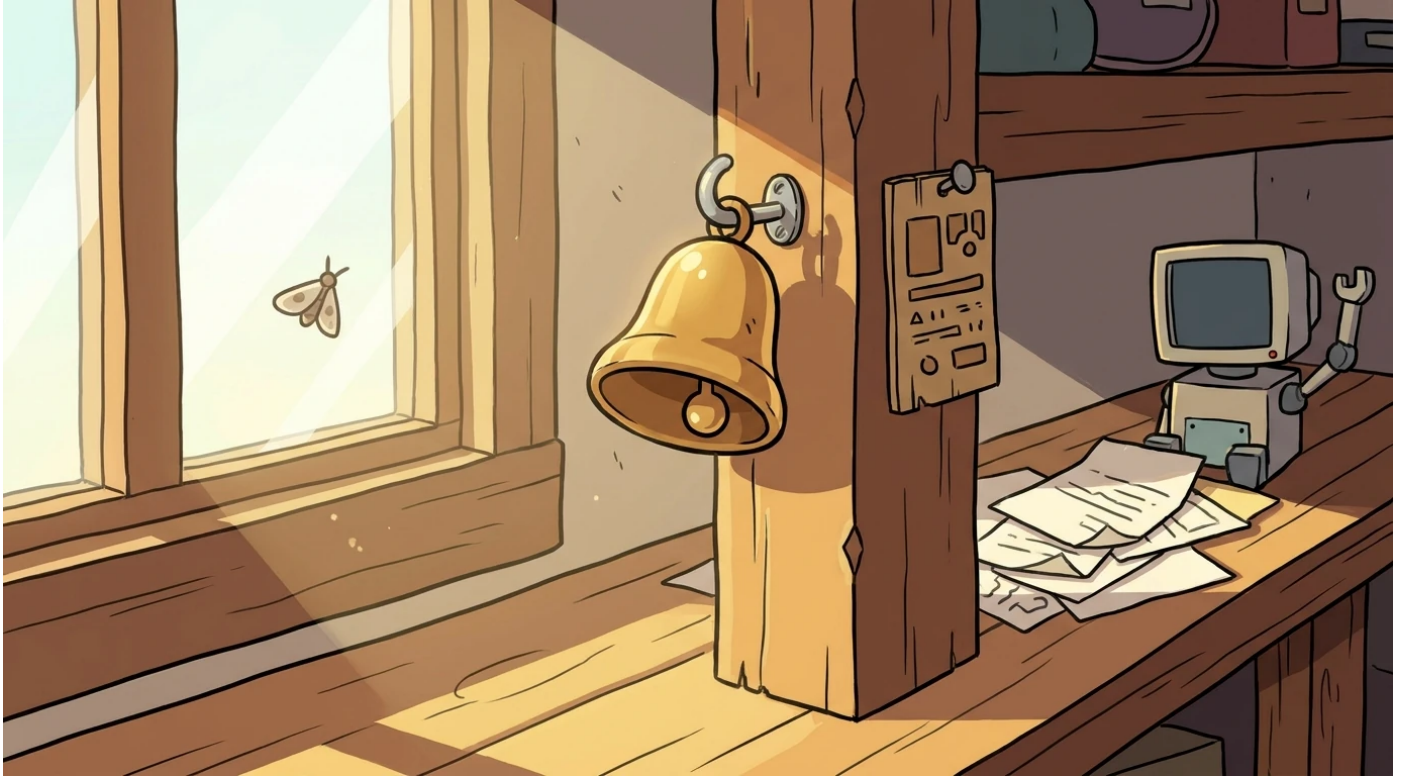
"That's the doorway," Loop said. "Not the computing. The *hearing*." He rested a hand flat against Port's quiet wood.

"Everything you make that reads the world and answers back — every game that feels your tap, every robot that turns toward you — it's carrying a little of what you felt just now. Somebody made something, and the something answered, and neither of them was alone anymore."

Sam nodded, and headed out the real door into the evening — still warm, somewhere in the chest, with the small bright feeling of having been heard.

Ping

*EVENT — *do this WHEN that happens*. An event handler waits, doing nothing, until a specific trigger fires — a button press, a timer, a sensor crossing a line — and only then runs its code. Not a loop that keeps checking; a waiting bell that rings the instant it's struck.*



On the corner of Loop's workbench hung a small brass bell, and it was doing absolutely nothing.

It had been doing nothing all morning. Sunlight crossed the bench, a moth bumped the window, Loop shuffled papers — and the bell just hung there on its little hook, silent, patient, not so much as a shiver. Beside it a card was pinned to the wood. The card said, in Loop's tidy handwriting: **WHEN struck → run this**.

A new kid drifted over, program half-finished, and frowned at it. "That bell's broken," she said. "It's not doing anything."

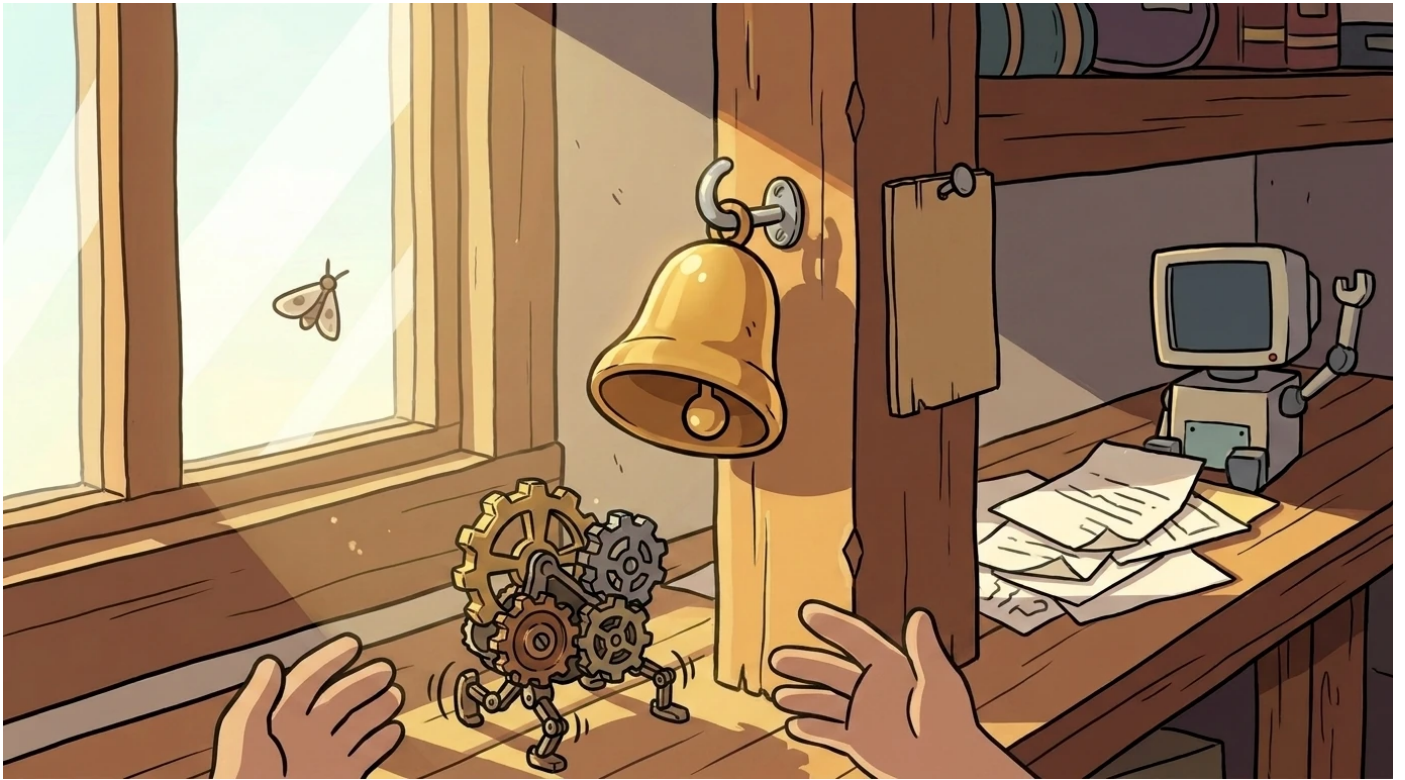
"It's the busiest thing in the room," Loop said, not looking up.

The kid squinted. The bell was extremely quiet for something busy.

"Watch." Loop reached over and flicked the bell with one finger. *Ping!* — a single bright note — and the instant it rang, Bit, the little robot at the end of the bench, snapped its arm up and beeped. The kid jumped. The bell went still again, silent, back to doing nothing.

"See that?" Loop said. "It waited. It didn't check the room a hundred times a second asking *is it time yet, is it time yet*. It just hung there, resting, costing nothing — until the exact moment it got struck. Then, and only then, it ran." He tapped the card. "*When this happens, do that*. That's Ping. That's an event."

The kid reached out and flicked it herself. *Ping* — Bit beeped again, instantly. She flicked it twice more, delighted, and Bit answered every single time, right on the note.



The kid wanted to know why it only ran when struck — why it didn't just run on its own like everything else she'd built.

So Loop told her about Trek, the loop that lived one shelf over. Trek was a little walking gear that went *around and around and around*, doing its work over and over, never resting, checking and rechecking. "Trek is wonderful," Loop said, "but Trek never sits still. If you asked Trek to watch for a button, Trek would ask *is it pressed? is it pressed? is it pressed?* forever, until it wore itself out. Thousands of questions a second, and almost every answer is *no*."

He set Ping and Trek side by side. Trek whirred. Ping hung silent.

"Ping does it the lazy-clever way," Loop said. "Ping doesn't ask. Ping waits to be *told*. You hook it to a trigger, you write down what to run, and then Ping does the hardest, cheapest thing there is — it does nothing, perfectly, until the one moment that matters." He smiled. "Nothing is a lot of work when you have to do it *exactly right*."

The kid looked at the silent bell with new respect. It hadn't been broken at all. It had been ready.



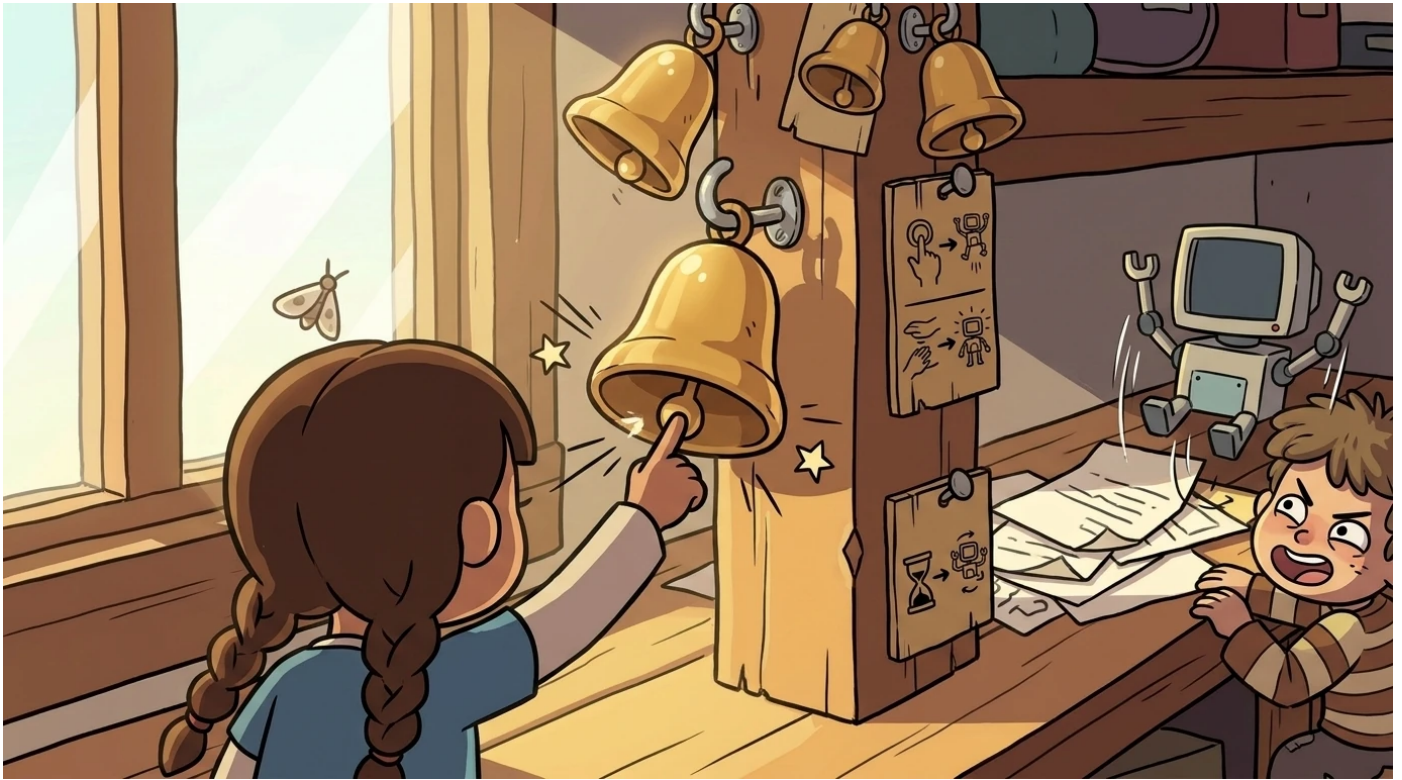
The kid decided a bell that quiet belonged at CodeRealm, where they studied the parts a program is built from — so she carried Ping down to Loop's classroom and hung it by the door.

Loop met her there. He didn't ask her to prove Ping was clever. He asked one thing. "What is an event?"

The kid didn't answer with words. She hooked Ping to a big red button, pinned a card beside it that read **WHEN button pressed → Bit spins**, and then stepped back and folded her arms and waited. Nothing happened. The bell hung quiet. Loop raised an eyebrow, testing her — "It's not doing anything."

"It's *ready*," she said, and pressed the button. *Ping!* Bit spun in a fast, happy circle and stopped. "It didn't run until the trigger. It waited for one specific thing, and *when* that thing happened, it ran its code. That's all."

Loop watched Bit come to rest. "You belong here," he said.



Ping's corner of the classroom soon filled up with quiet, waiting bells.

A boy came in one afternoon, stuck and grumpy. He'd built a game where a robot was supposed to react, but it just sat there. "I told it to jump," he said, "and it does nothing. I did all the work and it does *nothing*."

"Show me," the kid said.

He showed her. His program had Bit asking, over and over, *did they press it yet? did they press it yet?* — so busy asking that it never had a moment to actually jump. "It's exhausting itself," the kid said. "Let's give it a bell instead."

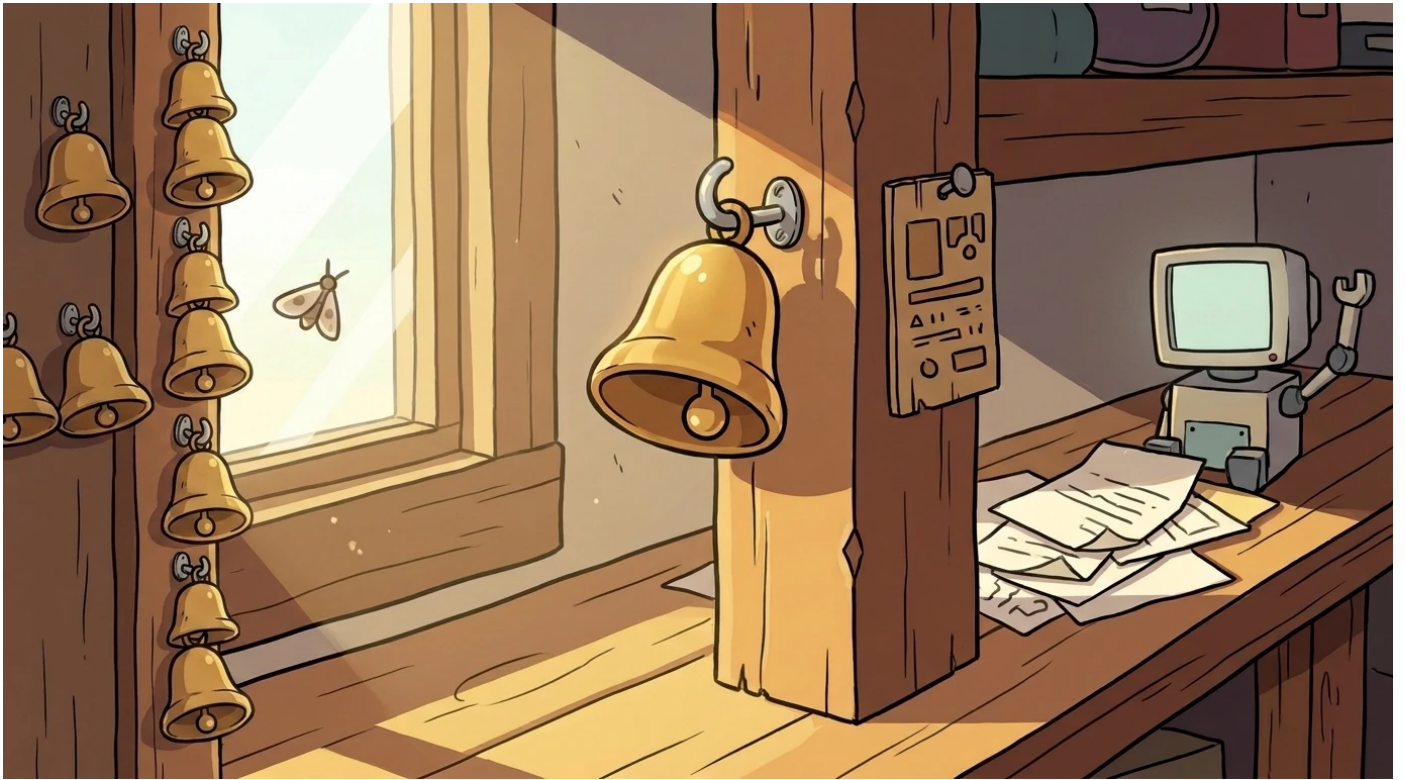
She hooked a Ping to his button and wrote a card: **WHEN button pressed → Bit jumps**. Then she added two more bells. **WHEN sensor covered → Bit lights up**. **WHEN 5 seconds pass → Bit spins**. Three bells, all hanging silent, each waiting for its own one thing.

"Now go on," she said. "Press it."

He pressed the button — Bit *jumped*, the very instant his thumb hit the key. He laughed, surprised, and covered the sensor — Bit lit up. He waited, holding his breath — and after five seconds, all on its own, Bit spun. Three different triggers, three different bells, each ringing exactly on cue.

"Every one answered the *second* I did it," the boy said, grinning wide. "It feels alive."

"That's the trick," the kid said. "You stop asking a thousand times, and you just wait for the strike. *When* this, *do* that."



Later, when the room was empty, the boy came back with one more question. He was quieter now, watching the row of silent bells.

"When they're just hanging there," he said, "not ringing, not doing anything — how do you know they're still working?"

The kid thought about the first bell on Loop's bench, the one she'd called broken.

"You feel it when it fires," she said. "That's the honest answer. There's this bright little jolt — you press the button and the thing answers *right then*, the exact moment you made it happen, and it feels wonderful. Quick. Awake. Like the program was listening for you the whole time." She looked at the row of quiet bells by the door. "They're not doing nothing. They're doing the patient thing. They're waiting for their one moment — and when it comes, they don't miss it."

The boy nodded slowly, and headed off still grinning at how fast Bit had answered his call. Behind him the bells hung silent, ready, each one listening for the strike that would wake it.

Shuffle

*RANDOMNESS — *a value you can't predict, picked fresh each time.* A random primitive produces an unpredictable result — a dice roll, a shuffled deck, a coin flip — so programs can surprise, vary, and stay fair. The same code can behave differently every run.*



In the corner of the CodeRealm workshop sat a covered cup, and it was, at that exact moment, doing something no one in the room could predict.

Loop had it on the bench in front of a class of squinting kids. It was a plain tumbler, lidded, and when Loop gave it a shake it made a dry, rattling music — a whole crowd of small numbered tokens knocking together in the dark. He held it up so everyone could hear it.

"This is Shuffle," Loop said. "Randomness. Watch closely, because I promise you I don't know what happens next either."

He tipped the cup. A single token slid out onto the bench and rolled to a stop. **4**.

"Four," Loop said, as if it were the most ordinary thing in the world. He scooped the token back in, capped the cup, shook it — that rattling music again — and drew once more. **1**. Then again, **6**. Then **1** a second time.

A kid in the front row leaned forward, frowning. "Do it once more."

Loop did. **3**. The kid sat back, a little defeated and a little delighted, because they had genuinely been trying to guess and there had been nothing to grab onto — no pattern, no rhythm, no next-in-line. Just a cup, a shake, and a fresh surprise every single time.

"That," Loop said, "is the whole job. When a program needs something you can't guess ahead of time — a dice roll, a random enemy, a shuffled quiz — you shake Shuffle and draw. A value nobody can see coming. Not the workshop, not you, not even me."

He set the cup down, and it sat there quietly with all its unknown tokens hidden inside, patient and full of surprises.



Loop noticed a kid at the back who wasn't laughing.

This kid had built a guessing game the week before, and it had gone wrong in the most boring way possible: every time you played it, the "secret" number was 7. Always 7. The first player found it in a run, told everyone, and after that the game was dead. No one could be surprised twice.

"It wasn't broken," the kid said, miserable. "It ran perfectly. It was just... the same. Every time. I made a game that couldn't surprise anybody."

Loop pulled the cup over and set it in front of them.

"You didn't need a bigger machine," Loop said. "You needed one that forgets. Watch." He shook the cup and drew — 5 — capped it, shook, drew again — 2. "It doesn't remember the last draw. It has no favorite number. There's no seven waiting in there for you to trip over. Every draw is fresh."

The kid picked up the cup themselves. Shook it. Drew. 6. Shook. Drew. 1. And something loosened in their chest — the flat, stuck feeling of *the same, the same, the same* lifting off, replaced by the small rising fizz of *I don't know what's next*. They shook it a third time just to feel it again.

"See," Loop said gently. "That jumpy, don't-know feeling? That's the thing your game was missing. Not a bug. A *surprise*."



The kid brought their game back to the workshop the next morning and set the cup on the bench like it belonged there.

"I want to build it right this time," they said.

Loop didn't ask them to prove anything. He asked one question. "What does randomness give a program?"

The kid didn't answer with words. They wired Shuffle into the guess-a-number game so that each round, the workshop shook the cup and drew a fresh secret token from 1 to 9 — and then they made Loop play it. Loop guessed 4. Wrong; it was 8. They reset. Loop guessed 8 on purpose this time, grinning. Wrong again; it was 2.

"You can't cheat it," the kid said, "because I don't even know what it'll draw. It picks fresh, every round, from the cup."

Loop played three more rounds and lost every guess, and by the end he was laughing.

"You belong here," he said, and set the cup back on the bench beside the kid's game like it had always lived there.



By afternoon the kid had a little crowd, and one of them wanted a game of their own.

"I need a dice," this new kid said. "For a board game. But I don't get it — how does a *cup* roll a six?"

The kid who now understood Shuffle pulled the cup over. "Watch. You put six tokens in — 1 to 6. Shake." Rattle. "Draw." 3. "There's your roll. Want a coin flip instead? Two tokens — heads and tails. Want a random cheer? Fill a list with phrases, shuffle it, and draw one. Same idea every time: shake, draw, get a fresh one."

The new kid dropped in the six tokens and drew. 5. Then 5 again.

"Wait — two fives! Is it stuck? Am I due for a low one now?"

The first kid shook their head, remembering exactly how Loop had said it. "That's the honest part. It doesn't keep score. Two fives in a row doesn't make the next draw any *less* likely to be a five. The cup has no memory and no fairness-debt — it isn't 'saving up' a low number for you. Every draw is fresh and completely on its own."

The new kid drew again, half-braced for a five, and got 1, and the whole small crowd let out a *whoa*. They played on, and every roll landed like a tiny gift nobody could open early.



When the workshop emptied out, the first kid stayed behind with the cup, running their fixed game one more time, alone.

They'd wired it so it drew a fresh secret number *and* a random cheer *and* shuffled the questions into a new order. They ran it. 3, a chirped *Nice one!*, the questions in one order. They ran it again. A different secret, a different cheer, a different order entirely. Same exact code. Brand-new run.

And then it happened: the game surprised *them*. They'd built every line of it, and it still did something they hadn't seen coming — and they laughed out loud in the empty room, alone, at their own machine.

The cup sat on the bench, lid on, tokens hidden, quietly ready for the next shake. It wasn't thinking. It wasn't feeling lucky. It was just a cup of tokens you shook and drew from, fresh every time. But the kid, watching a thing they'd made astonish them run after run, felt the brightest, jumpiest kind of happy — the delight of *not knowing what comes next*, of building something alive enough to still amaze the one who built it.

They shook the cup one more time before they left, just to feel that small rising fizz, and drew a 6, and grinned all the way out the door.

About Spark & Anvil

Spark & Anvil is a 501(c)(3) public charity. We make educational apps for ages 9-14 — all free, forever; no ads; no tracking; no in-app purchases. CodeRealm is one of 140+ apps in the portfolio.

More chapter books from Spark & Anvil

Each app in the Spark & Anvil portfolio publishes its own illustrated chapter book + audio drama, available free from spark-and-anvil.com/books. Highlights include:

- **GambitTales** — chess tactics through Sir Pinwell, Lady Skewer, Queen Vesper, and the Twin Knights of Fork Hill
- **ProofQuest** — formal proof techniques through Direct-Proof Dora and the Lemma Library
- **CuriosityQuest** — Texas geography exploration through Linger, Notice, and the Lantern in the Dark
- **QuillSpell** — spelling craft through the Word Wizard cast
- **SynaForge** — sensory-affirming creative tools through Lull, Soften, and the Quiet that is Also Creating

Methodology

Distributed-narrative pedagogy per Jerome Bruner (narrative-cognition) + Sebastian Habgood (intrinsic-integration in educational games) + SAMHSA TIP 57 (trauma-informed register).

Trauma-informed-design framework per Eggleston et al. (2025) and Stoltenburg et al. (2024).

License

© 2026 Spark & Anvil (501(c)(3) public charity). Chapter text and illustrations licensed under CC BY-NC-SA 4.0. App software © Spark & Anvil — all rights reserved. Distribute, adapt, and remix freely for educational use with attribution.

Cover art, chapter illustrations, and chapter text are reviewed for age-appropriate, trauma-aware content before publication. Audio drama transcripts available at spark-and-anvil.com/cast.