Unit 1 Logic

Content Area: **Computer Science**
Course(s):
Time Period: **Marking Period 1**
Length: **10 days**
Status: **Published**

Summary

Unit 1 launches the Logic, Games, and Puzzles course by building students' foundational computational thinking (CT) skills: algorithms, precise sequencing, if/then conditionals, flowcharts, Boolean logic, and debugging. Over ten 40-minute periods, students move through a deliberate teach-apply-rehearse cycle — each new concept (Days 1–7) is introduced through an unplugged, hands-on activity and immediately applied, so no skill is ever learned in isolation. Day 8 is a low-stakes dress rehearsal that combines every skill before students are formally assessed. The unit culminates in the "Teach the Robot" capstone (Days 9–10): each student writes a precise, literal algorithm for a real-world task, a partner executes it exactly as written while the writer observes silently, and the writer then completes a written Debug Report diagnosing and correcting the failures that surfaced. This capstone structure — individual work, partner testing under a silent-observer rule, and written reflection — is intentionally consistent with the capstones of Units 2 and 3, so students experience the same design-test-debug cycle across the full course. Unit 1 sets the CT vocabulary (algorithm, conditional, Boolean logic, syntax error, logic error, edge case) that both later units build on directly.

Revision Date: July 2026

Essential Questions

- How do problem solvers break a complex task into steps that are precise and unambiguous enough for anyone — or anything — to follow correctly?
- Why does the order, precision, and structure of instructions determine whether a solution actually works?
- How do conditional (if/then) and Boolean (AND, OR, NOT) logic allow a system to make decisions rather than simply follow a fixed path?
- What can a failed plan teach us, and how should we respond when something we built doesn't work as intended?

Enduring Understandings

- Precise, sequential, and logical thinking — algorithms, conditionals, and Boolean logic — is the foundation of solving both real-world and computational problems.
- Errors are an expected and valuable part of the design process; effective problem solvers anticipate

edge cases and use structured debugging (identifying whether a failure is a sequence error, a conditional error, or a logic error) to diagnose and correct failures rather than guessing or starting over.

Objectives

Students Will Know

- Students Will Know the definition of an algorithm and the difference between a vague instruction and a precise, “robot-proof” one.
- Students Will Know how the order (sequence) of steps changes the outcome of an algorithm.
- Students Will Know the structure and purpose of if/then conditional statements, including the risk of a missing branch (an unaddressed “else” case).
- Students Will Know how flowcharts visually represent algorithms, decision points, and possible dead ends.
- Students Will Know the Boolean logic operators AND, OR, and NOT and how they combine conditions to include or exclude possibilities.
- Students Will Know the vocabulary of debugging, including the distinction between a syntax error (a step that is missing or wrong) and a logic error (a step that runs but produces the wrong result).
- Students Will Know what an edge case is and why an algorithm that works most of the time can still fail under unusual conditions.

Students Will Be Skilled At

- Students Will Be Skilled At writing precise, step-by-step algorithms for real-world tasks.
- Students Will Be Skilled At correctly sequencing instructions and rewriting ambiguous steps so they can be followed literally.
- Students Will Be Skilled At writing and applying if/then conditional statements, including statements that address both branches of a decision.
- Students Will Be Skilled At creating and interpreting flowcharts using a digital drawing tool.
- Students Will Be Skilled At applying Boolean logic (AND, OR, NOT) to sort, filter, and evaluate real and abstract conditions.
- Students Will Be Skilled At identifying, diagnosing, and correcting errors in algorithms, flowcharts, and logic statements.
- Students Will Be Skilled At anticipating edge cases and revising an algorithm to handle them.
- Students Will Be Skilled At collaborating with a partner to test a solution objectively (as a literal

"robot") and giving specific, evidence-based feedback.

Learning Plan

Instruction in Unit 1 follows a consistent daily rhythm: an unplugged, hands-on activity introduces each new concept concretely before it is named or formalized, a short digital extension (Code.org, CS Unplugged, or a flowcharting tool) reinforces the idea in a second context, and a brief exit ticket checks understanding and previews the next day's concept. Every lesson deliberately mirrors a skill students will need for the Day 9–10 capstone, so the sequence functions as a scaffolded rehearsal rather than a set of disconnected activities.

The unit opens on Day 1 with the question "What is an algorithm?" The teacher models the idea by literally following a student-written set of instructions for a simple task (such as standing up from a chair) exactly as written, which quickly and humorously exposes how vague everyday instructions really are. Students then write their own first informal algorithm (making a sandwich, brushing teeth) with no constraints, surfacing their instincts before any formal structure is introduced. Day 2 shifts to precision and sequence: students physically sequence a shuffled set of instruction cards for a task such as packing a bag, flag any step too vague to follow literally, and rewrite it, then extend the idea with a short Code.org sequencing activity.

Days 3–5 build the logical toolkit students will use throughout the course. On Day 3, the class builds an if/then "What should I wear?" flowchart together, and students then write three original if/then statements of their own. Day 4 moves that logic into a visual flowchart built with a free digital tool (Canva, Google Drawings, or Lucidchart); pairs then swap flowcharts and try to follow them literally, a low-stakes preview of the capstone dynamic. Day 5 introduces Boolean logic (AND, OR, NOT) through a physical card-sorting game with rules such as "Pick up a card IF it is red AND greater than 5," formalizing the vocabulary once students have felt the logic operate concretely, and closes with a short digital Boolean logic reinforcement activity.

Days 6 and 7 turn the class's attention to failure. On Day 6, students receive two to three deliberately broken algorithms and must identify each bug and rewrite the fix. At the same time, the teacher introduces the explicit language of debugging syntax errors versus logic errors that carries directly into the capstone Debug Report. Day 7 introduces the unit's most sophisticated idea, edge cases: as a class, students interrogate an earlier algorithm by asking what happens if the user is missing materials or if two conditions are true at once, then revise their own Day 1 algorithm to handle at least one edge case they had not originally considered.

Day 8 is a deliberately low-stakes dress rehearsal. Students write a short algorithm for a new task from the capstone task list, this time intentionally including all five capstone elements: precise steps, conditionals, Boolean logic, at least one edge case, and a self-identified potential bug. Pairs conduct an informal, ungraded "robot test" of each other's algorithms so that any remaining confusion surfaces before the graded capstone, and the remainder of the period is reserved for questions and clarification.

The unit closes with the "Teach the Robot" capstone on Days 9–10. On Day 9, students independently write a

complete algorithm for a task chosen from a teacher-provided list (for example, making a bowl of cereal, packing a backpack, sorting a hand of cards, choosing what to wear based on the weather, or finding a specific book on a shelf), incorporating at least two if/then conditionals, at least one Boolean logic check, and at least one addressed edge case. On Day 10, students are paired, and each acts as the literal "robot" for their partner's algorithm, following the instructions exactly as written with no interpretation or common-sense fill-ins; the writer observes silently, resisting the urge to clarify, which is itself part of the learning and takes notes on where the algorithm breaks down. Students then complete a written Debug Report that identifies the first step that failed, classifies the error, writes a corrected version of that step, and explains what they would build in from the start to prevent the bug. Before Day 9, the teacher models the entire process for the whole class by acting as the robot for a clearly broken sample algorithm, so students fully understand how literally "the robot" will follow their instructions.

Assessment

Formative Assessment

Daily exit tickets (Days 1–7) check understanding of each day's concept and surface misconceptions before the next lesson builds on it. The Day 8 informal, ungraded "robot test" functions as a formative dress rehearsal, giving both students and the teacher a clear read on readiness before the capstone. Throughout the unit, the teacher circulates during partner and small-group work to observe reasoning in real time and conferences informally with students who are struggling with a specific concept (e.g., conditionals or Boolean logic) before it compounds.

Summative Assessment

The "Teach the Robot" Algorithm Design Task (Days 9–10) is the unit's summative performance assessment, worth 20 points. Students are scored on a four-point rubric across five criteria that map directly to the unit's objectives: Algorithm Writing, If/Then Conditionals, Boolean Logic, Pattern Recognition/Edge Cases, and Debugging (evaluated through the written Debug Report). Because the rubric mirrors the skills taught and rehearsed on Days 1–8, every score point is traceable to specific instruction the student received earlier in the unit.

Benchmark Assessment

As a newly piloted computer science elective, Unit 1 does not currently draw on an existing district-wide diagnostic (e.g., iReady, IXL). The department will consider developing a short, locally-created computational thinking pre/post skills check — covering algorithm sequencing, conditional logic, and Boolean reasoning — to benchmark growth across the pilot year and to inform any future revisions to this unit.

Alternative Assessment

The Day 8 partner "robot test" and the Day 9–10 capstone are themselves performance-based alternative assessments: students demonstrate mastery by producing a working artifact (an algorithm) and by observing, under a structured silent-observer protocol, how that artifact performs when executed literally by another

person. The Debug Report functions as a written reflection/self-assessment component, requiring students to explain — not just fix — the failure they observed.

Materials

Core Instructional Materials

- Teacher-facing lesson plans, Days 1–10 (timing grids, CT-connections tables, differentiation notes)
- Student-facing activity sheets, Days 1–10 (print-ready)
- Printable instruction card sets for the Day 2 sequencing activity
- Day 4 flowchart step-by-step slide presentation (“Should I Bring an Umbrella?” example)
- “Teach the Robot” capstone packet: Day 9 Algorithm Writing task and Day 10 Robot Test & Debug Report
- Sub-friendly contingency lessons (puzzle exploration and game analysis activities, each with student sheet, rubric, and sub notes) for planned or unplanned absences

Supplemental Materials

- Code.org early algorithm and sequencing puzzles
- CS Unplugged Boolean logic activities
- A free digital flowcharting tool: Canva, Google Drawings, or Lucidchart
- A standard deck of playing cards (Day 5 Boolean sorting activity)

Standards

Computer Science (NJSLS-CSDT, 2020) — 8.1 Computer Science

Code	Performance Expectation
8.1.8.AP.1	Design and illustrate algorithms that solve complex problems using flowcharts and/or pseudocode.
8.1.8.AP.3	Design and iteratively develop programs that combine control structures, including nested loops and compound conditionals.
8.1.8.AP.4	Decompose problems and sub-problems into parts to facilitate the design,

	implementation, and review of programs.
8.1.8.AP.5	Create procedures with parameters to organize code and make it easier to reuse.
8.1.8.AP.6	Refine a solution that meets users' needs by incorporating feedback from team members and users.
8.1.8.AP.8	Systematically test and refine programs using a range of test cases and users.
8.1.8.AP.9	Document programs in order to make them easier to follow, test, and debug.

Design Thinking (NJSLS-CSDT, 2020) — 8.2 Design Thinking

Code	Performance Expectation
8.2.8.ED.2	Identify the steps in the design process that could be used to solve a problem.
8.2.8.ED.3	Develop a proposal for a solution to a real-world problem that includes a model (e.g., physical prototype, graphical/technical sketch).
8.2.8.ED.7	Design a product to address a real-world problem and document the iterative design process, including decisions made as a result of specific constraints and trade-offs.

Career Readiness, Life Literacies & Key Skills (NJSLS 9.4, 2020)

Code	Performance Expectation
9.4.8.CT.1	Evaluate diverse solutions proposed by a variety of individuals, organizations, and/or agencies to a local or global problem, such as climate change, and use critical thinking skills to predict which one(s) are likely to be effective.
9.4.8.CT.2	Develop multiple solutions to a problem and evaluate short- and long-term effects to determine the most plausible option.
9.4.8.CT.3	Compare past problem-solving solutions to local, national, or global issues and analyze the factors that led to a positive or negative outcome.
9.4.8.TL.3	Select appropriate tools to organize and present information digitally.
9.4.8.TL.5	Compare the process and effectiveness of synchronous collaboration and asynchronous collaboration.

Interdisciplinary Connections — English Language Arts (NJSLS-ELA)

- RI.7.3 — Analyze the interactions between individuals, events, and ideas in a text (applied to reading and following multi-step written algorithms and instructions).
- W.7.2 — Write informative/explanatory texts to examine a topic and convey ideas through the selection of relevant content (applied to the Debug Report, where students explain what failed and why).

- SL.7.1 — Engage effectively in a range of collaborative discussions, building on others' ideas and expressing their own clearly (applied to partner "robot test" debriefs and Day 3–8 class discussions).

Integrated Accommodations & Modifications

Suggested Strategies for Modification for Computer Science