2.6 Debugging with CreatiCode XO

Duration: 100-110 minutes | Slide deck aligned: Slides 1-38

Students will be able to:

- Use debugging strategies to identify and explain the cause of a coding error.
- Apply a systematic debugging process to test and fix bugs in a project.
- Use CreatiCode XO to support debugging and improve a program's performance.

Learning Activity Summary

- Warm-up and preparation: introduce debugging clues, log in, and set expectations using Slides 1-3 (5 minutes).
- Part 1 - Learn to Debug: define debugging, build a debugging mindset, practice top-down search, logical deduction, simplification, logging, step-by-step mode, breakpoints, and explain-code debugging using Slides 4-23 (55 minutes).
- Part 2 - Debugging with CreatiCode XO: practice Expected vs. Actual descriptions, good XO questions, XO-supported debugging, and Socratic mode using Slides 24-36 (45 minutes).
- Wrap-up and Q&A: reflect on debugging confidence and review the complete toolkit using Slides 37-38 (5 minutes).

Student Materials

- Computer with Internet access
- CreatiCode account
- Lesson 2.6 slide deck
- Access to sample buggy projects and CreatiCode XO

Teacher Preparation

- Review the “How to Debug” and “CreatiCode XO - Debugging” tutorials.
- Open or bookmark the sample buggy projects for Problems 1-12.
- Verify that XO, logging/print blocks, step-by-step mode, and breakpoint blocks are working.
- Decide which problems students should complete independently, in pairs, or as optional extensions.
- Prepare assessment or exit-ticket directions, including the Wayground check if used.

Teacher Notes / Differentiation

Support: Pair students for collaborative debugging, provide Expected vs. Actual sentence frames, and ask guiding questions before giving fixes.

Extension: Assign advanced students additional problems such as Problems 15-20 or ask them to add conditional logging and breakpoints to a more complex project.

Pacing: Keep early explanations concise. Spend the most time on practice, partner explanation, and debugging with XO.

Debugging Culture: Normalize bugs as part of programming. Praise careful observation, specific questions, and one-change-at-a-time testing.

AI Use: Remind students that XO is an assistant. They should describe behavior clearly, test suggestions, and explain what changed.

Key for this guide

= explicitly explain

\+ = student action

= teacher tip

WARM-UP / PREPARATION (5 MINUTES)

Slide 1

Debugging with CreatiCode XO

Debugging with CreatiCode XO

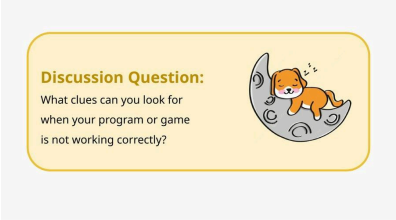
Slide shows: *Title slide with lesson name, part number, and CreatiCode branding.*

Welcome students and frame debugging as a normal part of building projects, not a sign that someone failed.

\+ **Student action:** Students log in, open CreatiCode, and get ready to investigate bugs.

Teacher tip: Keep the launch brief so most of the lesson time can be spent testing and reasoning through examples.

Slide 2
Discussion Question



Discussion Question

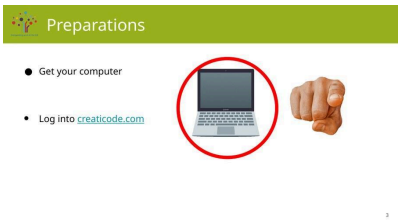
Slide shows: *Opening prompt: What clues can you look for when your program or game is not working correctly?*

Invite students to name clues such as error messages, unexpected movement, missing sprites, values that do not change, or a game rule that is not happening.

+ Student action: Students turn and talk, then share one clue they would check first.

Teacher tip: Use student responses to introduce the idea of comparing expected behavior with actual behavior.

Slide 3
Preparations



Preparations

Slide shows: *Students get computers and log into creaticode.com.*

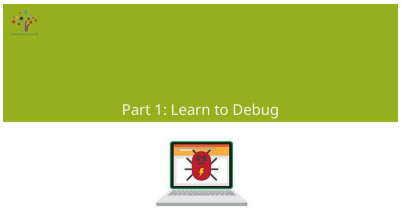
Make sure everyone has the right tools open before debugging begins.

+ Student action: Students get their computers, log into CreatiCode, and prepare to open the sample buggy projects.

Teacher tip: Resolve login issues now; debugging practice becomes difficult if students are not in the playground.

PART 1: LEARN TO DEBUG (55 MINUTES)

Slide 4
Part 1: Learn to Debug



Part 1: Learn to Debug

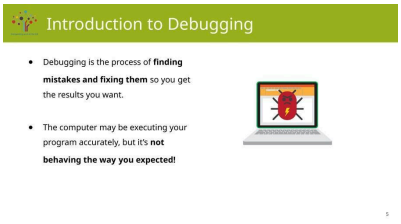
Slide shows: *Section divider introducing the first half of the lesson.*

Explain that Part 1 is about learning debugging strategies before asking XO for help.

+ Student action: Students listen for the strategies they will use during practice.

Teacher tip: Emphasize that XO is more useful when students already know how to describe and investigate a bug.

Slide 5
Introduction to Debugging



Introduction to Debugging

Slide shows: *Definition of debugging and the idea that the computer follows the program accurately even when the result is unexpected.*

Define debugging as finding mistakes and fixing them so the program produces the result we want.

+ Student action: Students restate the difference between what they expected and what actually happened.

Teacher tip: Avoid saying the computer is “wrong.” Say the program is behaving differently than expected.

Slide 6
What is Debugging?

What is Debugging?

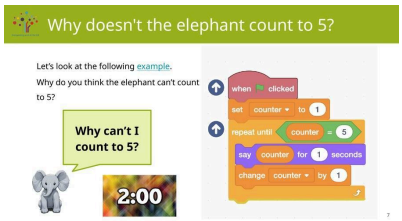
Slide shows: *Everyday examples: spelling a word correctly, fixing LEGO pieces, and fixing a game glitch.*

Connect debugging to familiar problem-solving experiences students already have.

+ Student action: Students add one everyday example of debugging and explain what was fixed.

Teacher tip: This slide helps lower frustration by showing that debugging is a common life skill.

Slide 7
Elephant Count Problem



Elephant Count Problem

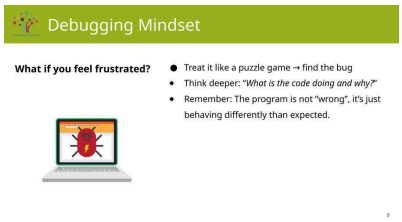
Slide shows: *An elephant counting example: Why does the elephant not count to 5?*

Introduce the first buggy project by asking students to observe before changing code: What should happen? What actually happens?

Student action: Students predict the bug and identify the expected vs. actual behavior.

Teacher tip: Do not reveal the fix too quickly; the goal is to practice reading behavior and code carefully.

Slide 8
Debugging Mindset



Debugging Mindset

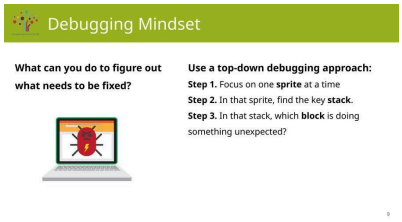
Slide shows: *Mindset reminders for frustration: treat it like a puzzle, think deeper, and remember the program behaves differently than expected.*

Normalize frustration and frame debugging as puzzle-solving.

Student action: Students choose one mindset statement that will help them when they get stuck.

Teacher tip: Reinforce patience before the class begins independent bug hunts.

Slide 9
Top-Down Debugging



Top-Down Debugging

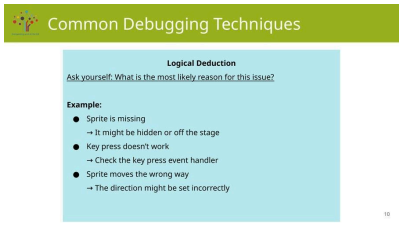
Slide shows: *Top-down approach: focus on one sprite, then one key stack, then the unexpected block.*

Teach the systematic narrowing process: sprite level, stack level, block level.

Student action: Students point to the sprite, stack, and block they would inspect first in a sample project.

Teacher tip: This top-down language should be reused throughout every practice problem.

Slide 10
Logical Deduction



Logical Deduction

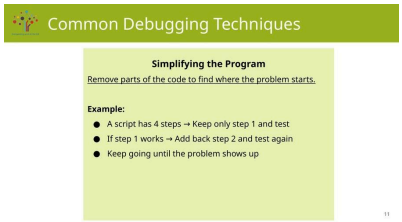
Slide shows: *Common technique: use clues to identify the most likely cause.*

Model logical deduction with examples: hidden or off-stage sprites, broken key press handlers, and incorrect directions.

Student action: Students match one symptom to one likely place to check in the code.

Teacher tip: Encourage students to make a reasoned guess before randomly changing blocks.

Slide 11
Simplifying the Program



Simplifying the Program

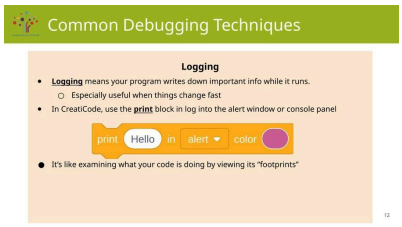
Slide shows: *Common technique: remove parts of the code to find where the problem starts.*

Explain that students can test one piece at a time by keeping step 1, adding step 2, and continuing until the problem appears.

Student action: Students describe which block group they would detach first in a long stack.

Teacher tip: This is especially useful when a long script has several steps and the bug location is unclear.

Slide 12
Logging



Logging

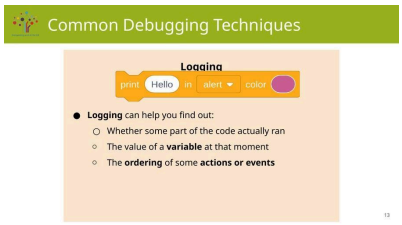
Slide shows: *Logging means the program writes down important information while it runs, often using print blocks.*

Define logging and explain that it leaves "footprints" showing what the code did while the project was running.

+ Student action: Students identify a value or event that would be useful to print in a fast-moving project.

Teacher tip: Use logging when behavior happens too quickly to observe by watching the stage only.

Slide 13
Logging: What to Check



Logging: What to Check

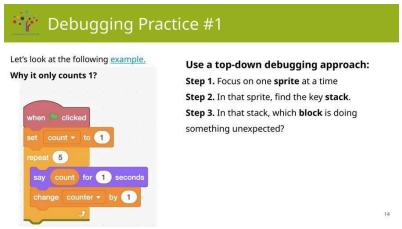
Slide shows: *Logging can show whether code ran, the value of a variable, and the ordering of actions or events.*

Explain three common logging goals: confirm a stack runs, check a value, and check the order of events.

+ Student action: Students write one possible log message and where they would place it.

Teacher tip: Push for specific logs such as "SPACE handler started" instead of vague logs like "test."

Slide 14
Debugging Practice #1



Debugging Practice #1

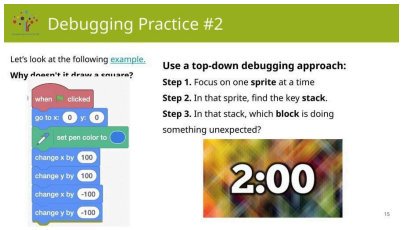
Slide shows: *Practice prompt: Why does the example only count 1? Use the top-down approach.*

Guide students through the first practice problem using sprite, stack, and block narrowing.

+ Student action: Students inspect the example, identify the unexpected block or placement, and explain their reasoning.

Teacher tip: Support students with questions rather than fixes: Which sprite? Which stack? Which block behaves unexpectedly?

Slide 15
Debugging Practice #2



Debugging Practice #2

Slide shows: *Practice prompt: Why does it not draw a square? Includes a timed practice window.*

Set a short investigation time and remind students to use top-down debugging instead of guessing.

+ Student action: Students work independently or with a partner, then share the most likely cause.

Teacher tip: If students are stuck, ask them to simplify the drawing script and test one step at a time.

Slide 16
Step-by-Step Mode

Step-by-Step Mode

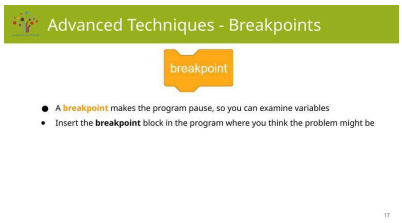
Slide shows: *Advanced technique: step-by-step mode for running code slowly.*

Introduce step-by-step mode as a way to slow down execution and watch what happens one block at a time.

+ Student action: Students observe where the program pauses or exits step-by-step mode.

Teacher tip: Use this for complex behavior; it may be slower than other methods for simple bugs.

Slide 17
Breakpoints



Breakpoints

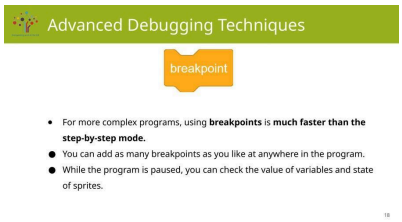
Slide shows: *Breakpoint block explanation: pause the program where the bug might happen.*

Explain that a breakpoint pauses the program so students can inspect variables and sprite states.

+ Student action: Students identify a place where a breakpoint would help them check the program state.

Teacher tip: Place breakpoints near the suspected problem, not everywhere at once.

Slide 18
Using Breakpoints



Using Breakpoints

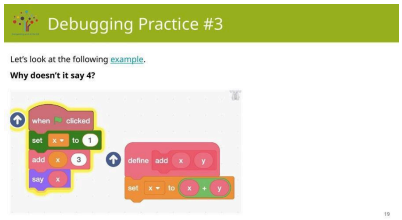
Slide shows: *Breakpoints are faster than step-by-step mode for more complex programs and can be placed in multiple locations.*

Compare breakpoints with step-by-step mode and explain why breakpoints are efficient in bigger projects.

+ Student action: Students decide whether a sample bug needs step-by-step mode or a breakpoint.

Teacher tip: When the program pauses, students should inspect values and sprite state before changing blocks.

Slide 19
Debugging Practice #3



Debugging Practice #3

Slide shows: *Practice prompt: Why does it not say 4?*

Release students to apply one advanced technique or a common technique from earlier slides.

+ Student action: Students debug the example and explain which clue led them to the bug.

Teacher tip: Have early finishers write an expected vs. actual statement for the problem.

Slide 20
Advanced Practice #4

Advanced Practice #4

Slide shows: *Practice prompt: Why does it not turn when SPACE is pressed? Includes a timed practice window.*

Ask students to reason about event handlers, key presses, direction, and any relevant conditions.

+ Student action: Students use the two-minute timer to inspect the SPACE key behavior and propose a fix.

Teacher tip: This is a strong moment to introduce or revisit logging: print when the SPACE handler starts.

Slide 21
Debugging Practice #5

Debugging Practice #5

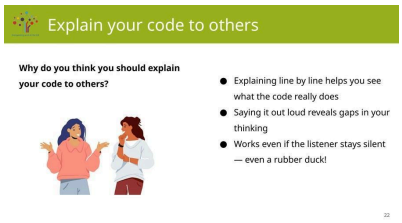
Slide shows: *Practice prompt: Why does it not move when the green flag is pressed?*

Guide students to check the green flag event, motion blocks, values, and sprite state.

+ Student action: Students identify why the sprite does not move and explain the most likely cause.

Teacher tip: Remind students to test from the green flag after each change, not after several changes at once.

Slide 22
Explain Your Code to Others



Explain Your Code to Others

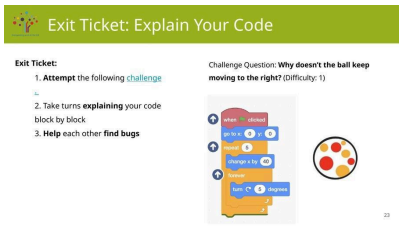
Slide shows: *Rubber duck style debugging: explaining line by line reveals gaps in thinking.*

Explain that saying code out loud helps students notice what the code actually does, not just what they hoped it would do.

Student action: Students practice explaining a small stack line by line to a partner.

Teacher tip: Partners should ask clarifying questions but avoid taking over the keyboard.

Slide 23
Exit Ticket: Explain Code



Exit Ticket: Explain Code

Slide shows: *Exit ticket directions and challenge: explain code block by block and help each other find bugs.*

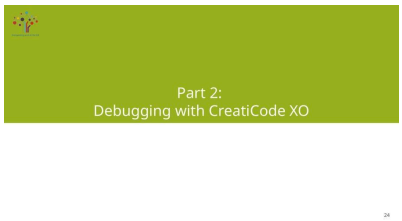
Use the exit ticket to check whether students can explain code and debug collaboratively.

Student action: Students attempt the challenge, take turns explaining code, and help each other find the bug.

Teacher tip: Collect a few responses or quick written explanations before moving to XO.

PART 2: DEBUGGING WITH CREATICODE XO (45 MINUTES)

Slide 24
Part 2: Debugging with XO



Part 2: Debugging with XO

Slide shows: *Section divider introducing CreatiCode XO as a debugging assistant.*

Transition from human debugging strategies to using XO as support.

Student action: Students prepare to describe bugs clearly before asking XO for help.

Teacher tip: Make the expectation clear: students still do the reasoning and testing.

Slide 25
Agenda for Part 2

Agenda for Part 2

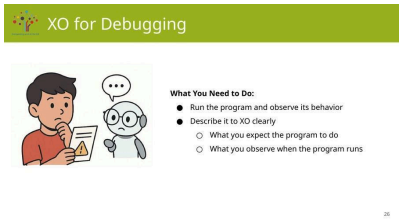
Slide shows: *Part 2 agenda: XO for debugging, how XO can help, Socratic mode, and wrap-up.*

Preview the XO workflow and connect it back to the strategies from Part 1.

Student action: Students listen for how XO supports, but does not replace, their debugging process.

Teacher tip: Keep the pacing brisk; this section works best with hands-on XO practice.

Slide 26
XO for Debugging



XO for Debugging

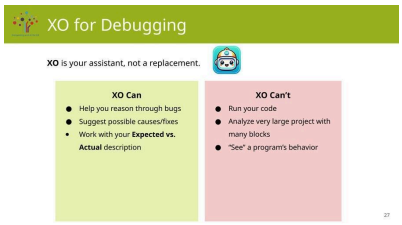
Slide shows: *What students need to do: run the program, observe behavior, describe expected behavior, and describe actual behavior.*

Teach the Expected vs. Actual format for asking XO useful debugging questions.

Student action: Students write or say one expected result and one actual result for a buggy project.

Teacher tip: Do not let students start with “It does not work.” Require observable details.

Slide 27
XO Is an Assistant



XO Is an Assistant

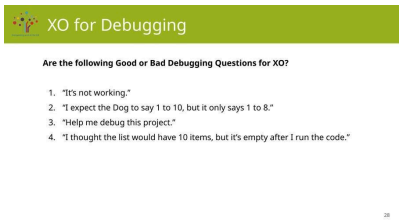
Slide shows: *XO can help reason through bugs, suggest causes/fixes, and use Expected vs. Actual descriptions; XO cannot run code or see behavior.*

Clarify XO's limits: it cannot directly observe the project or replace testing.

+ Student action: Students name one thing XO can do and one thing XO cannot do.

Teacher tip: This helps students treat XO responses as suggestions to test, not final answers.

Slide 28
Good or Bad Questions



Good or Bad Questions

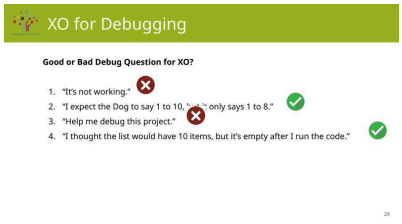
Slide shows: *Four sample debugging questions for XO before classification.*

Have students judge which questions give enough detail and which are too vague.

+ Student action: Students vote good/bad for each prompt and justify their choices.

Teacher tip: Look for students noticing that strong questions include what was expected and what actually happened.

Slide 29
Check Good/Bad Questions



Check Good/Bad Questions

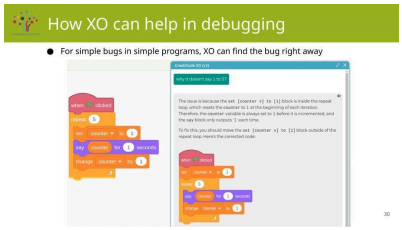
Slide shows: *The same four debugging questions with checks showing stronger examples.*

Reveal or confirm that detailed Expected vs. Actual questions are better than vague requests.

+ Student action: Students revise one bad prompt into a strong XO debugging prompt.

Teacher tip: A good revision should include the sprite, expected behavior, actual behavior, and any relevant trigger.

Slide 30
XO Finds Simple Bugs



XO Finds Simple Bugs

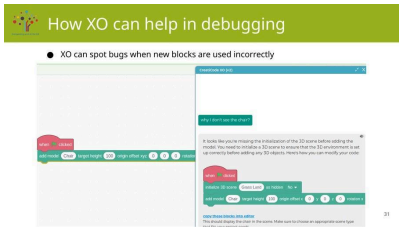
Slide shows: *Example showing XO can find simple bugs in simple programs right away.*

Explain that XO is most effective with small, focused projects and clear descriptions.

+ Student action: Students observe the example and note why the prompt is specific enough for XO.

Teacher tip: If a project is large, students should narrow the problem before asking XO.

Slide 31
XO Spots Incorrect Blocks



XO Spots Incorrect Blocks

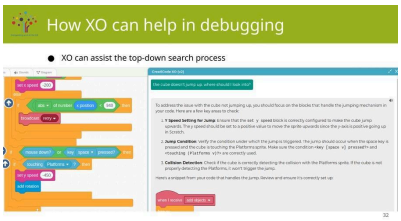
Slide shows: *Example showing XO can spot bugs when new blocks are used incorrectly.*

Explain that XO can help students understand how unfamiliar blocks should be used.

+ Student action: Students identify the new or unfamiliar block in the example and what might be wrong with it.

Teacher tip: Students should still verify XO's suggestion by testing the project.

Slide 32
XO Supports Top-Down Search



XO Supports Top-Down Search

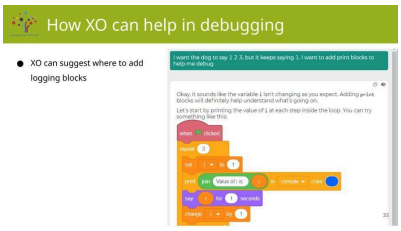
Slide shows: *Example showing XO assisting with a top-down search process.*

Connect XO's help to the sprite-stack-block process from Part 1.

+ Student action: Students ask XO a focused question that narrows the bug to one sprite or stack.

Teacher tip: Encourage follow-up questions that become more specific after each clue.

Slide 33
XO Suggests Logging



XO Suggests Logging

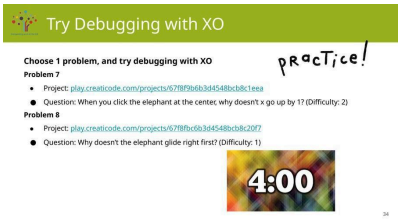
Slide shows: *Example showing XO suggesting where to add logging blocks.*

Explain that XO can recommend useful log messages and placements when behavior is hard to observe.

+ Student action: Students choose a place to add a print/log block based on the bug they are investigating.

Teacher tip: A strong log has a clear message and is placed before, inside, or after the suspected block.

Slide 34
Try Debugging with XO



Try Debugging with XO

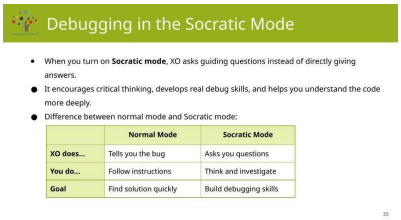
Slide shows: *Hands-on practice: choose Problem 7 or Problem 8 and debug with XO; includes project links and questions.*

Set the task: choose one problem, run it, observe the behavior, and ask XO using Expected vs. Actual language.

+ Student action: Students choose Problem 7 or 8, ask XO for help, test suggestions, and record what changed.

Teacher tip: Circulate to check that students are not asking XO vague questions or applying fixes without testing.

Slide 35
Socratic Mode



Socratic Mode

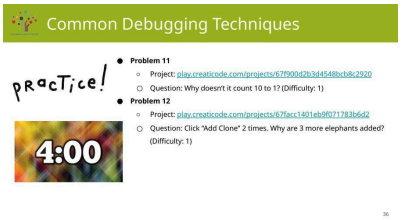
Slide shows: *Socratic mode explanation and comparison with normal mode.*

Introduce Socratic mode as a way for XO to guide with questions rather than direct answers.

+ Student action: Students compare normal mode and Socratic mode and decide when each is useful.

Teacher tip: Use Socratic mode when the goal is deeper understanding, not just a quick fix.

Slide 36
Socratic Practice



Socratic Practice

Slide shows: *Socratic mode practice with Problem 11 and Problem 12; includes a four-minute timer.*

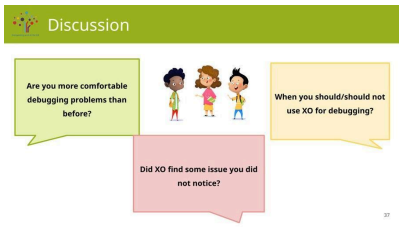
Assign one Socratic mode problem and remind students to answer XO's guiding questions thoughtfully.

+ Student action: Students work through Problem 11 or 12 using Socratic mode and explain what question helped them most.

Teacher tip: Step in if frustration becomes unproductive; the goal is learning the reasoning process.

WRAP-UP / Q&A (5 MINUTES)

Slide 37
Discussion



Discussion

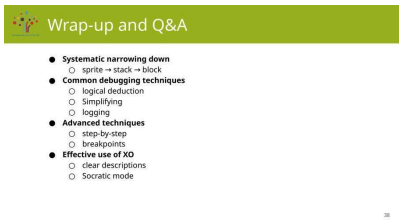
Slide shows: *Reflection questions: comfort with debugging, when to use XO, and whether XO found something new.*

Lead a short reflection on how students' debugging confidence changed during the lesson.

+ Student action: Students answer at least one discussion prompt verbally or in writing.

Teacher tip: Highlight thoughtful use of XO and independent reasoning, not just getting the right answer.

Slide 38
Wrap-up and Q&A



Wrap-up and Q&A

Slide shows: *Summary of key methods: sprite-stack-block narrowing, logical deduction, simplifying, logging, step-by-step, breakpoints, clear descriptions, and Socratic mode.*

Recap the complete debugging toolkit and connect each method to a time when it is useful.

+ Student action: Students ask final questions and name one strategy they will use in the next project.

Teacher tip: End by normalizing bugs as expected and reminding students to test one change at a time.

ASSESSMENT / OPTIONAL WAYGROUND CHECK

Assessment Notes

Use the exit ticket and/or Wayground check to assess whether students can:
- explain debugging as finding where actual behavior diverges from expected behavior;
- choose an appropriate technique, such as simplifying a long stack or adding logs;
- use Socratic Mode when they want XO to guide with questions;
- convert a vague request like "It does not work" into a specific Expected vs. Actual statement;
- describe a concrete logging plan with what to print and where to place the print blocks.

Wayground assessment link: https://wayground.com/admin/assessment/68c0ded8afb0f27281e2acd2?source=lesson_share