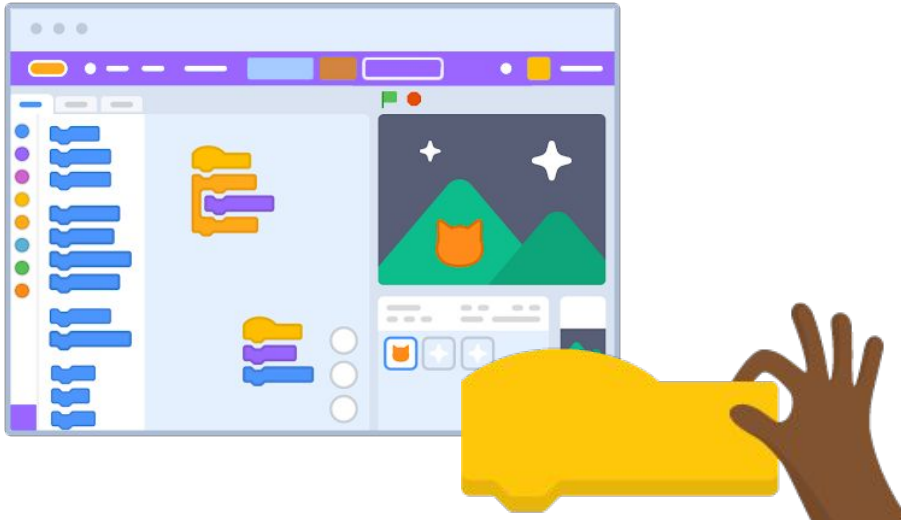
LEVEL UP WITH SCRATCH WORKSHOP SERIES

Squash "Bugs" and Conquer Challenges: Creative Problem-Solving

SCRATCH™

Session Overview

- Computational Concepts
- Creative Learning
- You Don’t Need to Have All the Answers
- Modelling
- Start with Exploration
- Similar But Different
- Coding Reading Challenge
- Comment Your Code
- Many Pathways & Version Control
- Coding Challenge
- Debugging Help
- Debugging Challenge 1 & Why
- Debugging Challenge 2 &Why
- Debugging Challenge 3 & Why
- Prompts to Try & Talk to a Duck!
- Debugging Reflection



Facilitator: Maren Vernon

Scratch Learning Resource Designer
@algorithmar and @scratchlycaterton

SCRATCH™ FOUNDATION

Learning Goals

- Practice debugging and explore strategies to get unstuck with fun challenges
- Embrace playful learning and tinkering mindset values (with the support of facilitation centering student-led inquiry, hands-on building, testing, iterating, and collaboration)
- Reflect on the debugging process

Getting Started

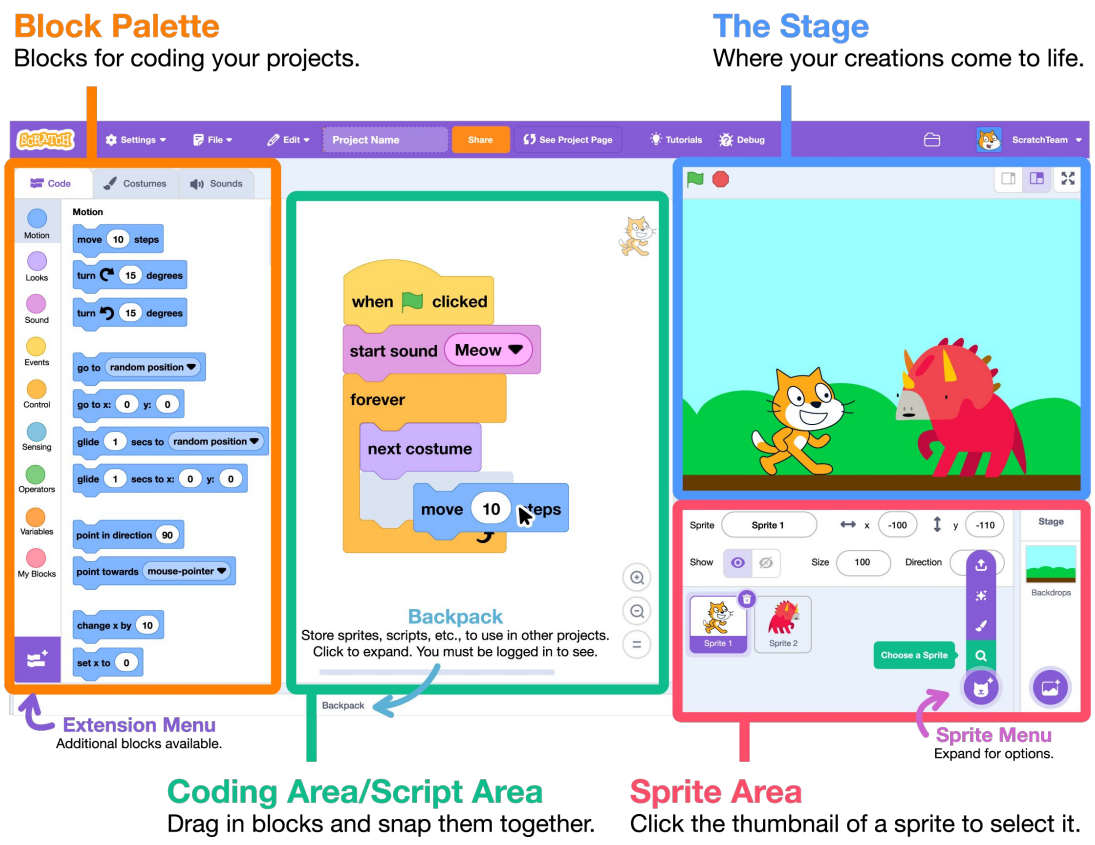
Click “Create” or log in to your free account to save projects.

go to: scratch.mit.edu

Set your language and block color mode.

Choose a sprite. Drag and drop code blocks to create a script.

scratchfoundation.org/learn/learning-library/getting-started



Let's Tinker...

Activate Growth Mindset

Computational Concepts

As Scratchers begin exploring computational concepts that are common in many programming languages...

- **sequence**: identifying a series of steps for a task
- **loops**: running the same sequence multiple times
- **parallelism**: making things happen at the same time
- **events**: one thing causing another thing to happen
- **conditionals**: making decisions based on conditions
- **operators**: support for mathematical and logical expressions
- **data:** storing, retrieving, and updating values

...they may run into common errors that will challenge their problem-solving skills and reinforce the importance of iteration and a growth mindset.

Creative Learning

As facilitators, we want to support **playful learning and tinkering mindset values** so that participants can:

- Engage playfully in **projects** that are meaningful to them and elicit joy
- Collaborate with **peers** to experiment, share, and celebrate ideas
- Develop a mindset that is **comfortable with the discomfort** of getting stuck
- Develop a mindset that thinks critically about **strategies for getting unstuck**

scratchfoundation.org/learn/learning-library/scratch-creative-learning-philosophy

You Don’t Need to Have All the Answers

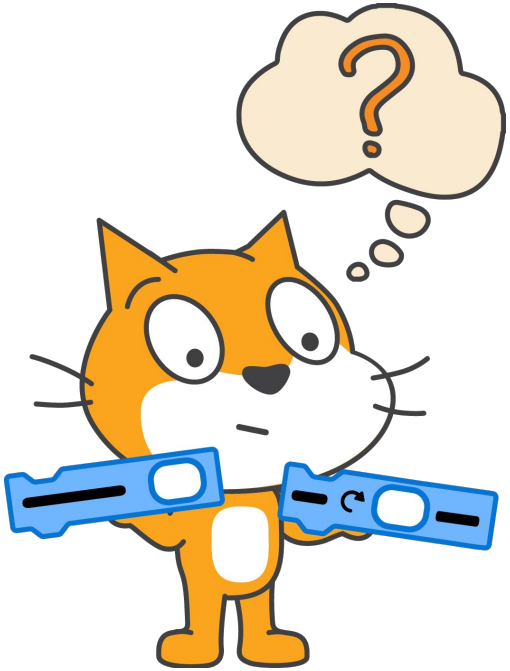
As a facilitator...

- You **don’t need to be a coding expert**.
- You **don’t need to have all the answers**.
- **Collaborative experimentation is the key!**

Even the most experienced programmers need to debug often and practice their growth mindset.

- **Answers can take time.**

Take a break and step away from the screen to clear your mind. After some rest, focusing on something else, or getting some water, you can approach the problem with fresh eyes.

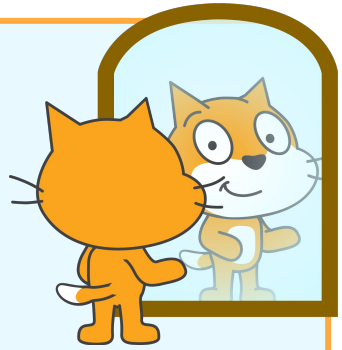


Modelling

It can be powerful to **model getting stuck, debugging, and iterating** alongside your learners.

We are learners just as much as our learners, so give yourself grace and remember, you don't have to have all the answers.

Encourage peer-to-peer conversations: "Ask Three Before Me," ask three peers before asking a facilitator. Bonus: Teaching others can be a powerful way of solidifying information for yourself, so give your learners a chance to try sharing knowledge.



Mistakes and failures are welcome

Get excited when something goes wrong! Rather than avoiding mistakes, encourage learners to be open to them.

As you support them through their work, help them focus on the process.

Start with Exploration

For those new to Scratch, it can be helpful to let users get stuck and **experiment a bit with blocks before sharing tips and tricks**.

In the blog post "Start with Exploration, Not Explanation," Natalie Rusk shared:

Young people who have developed broad creative, computational, and collaboration skills with Scratch usually first learned by "playing" or "messing around" with it, trying things out and seeing what worked. This playful approach helped them build their confidence in their ability to learn and problem solve.

Hold the tools as a last resort

It's tempting to grab the mouse, but try describing the steps rather than doing it for learners.

If you have to navigate the tools, let them try for themselves after you show them and guide them along.

Achievery Unit

Similar But Different

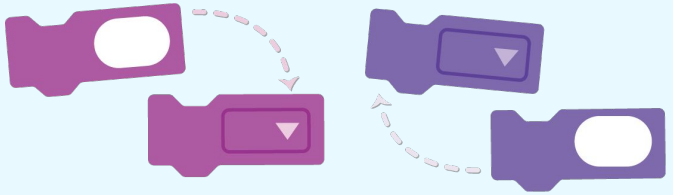
Check out our Achievery Unit "Similar But Different."

Some blocks look similar but can behave differently. This is an opportunity to pause and experiment. Try using a similar block in place of what you have, and see if this affects the result.

Consider pair programming to surface different solutions.

Examine. Write down a hypothesis. Create a code sequence and observe. When might you want to use one method over another?

- **Sound: "start sound" vs "play until done"**
- **Looks or Variables: "set" vs "change"**
- **"Say," "say" with time, "think," and "speak"**
- **"Move" vs "change x" (or y) vs "move" plus "point in direction"**
- **"Point in direction" vs "turn"**
- **"If then" vs "if then else"**
- **"repeat until" vs "wait until"**



Achievery Unit

Coding Reading Challenge

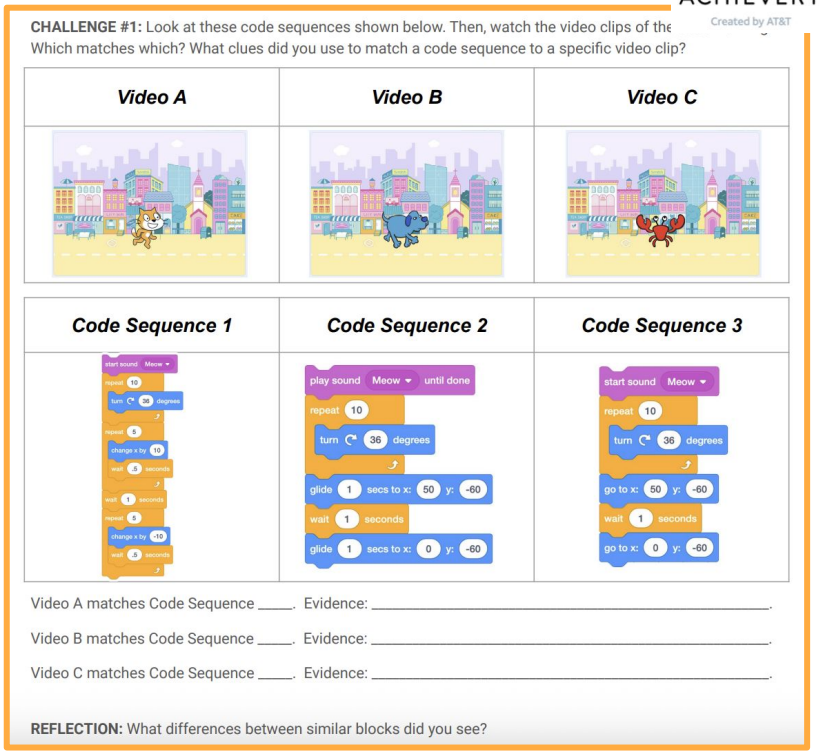
Check out our Achievery Unit "Code Reading Challenge." Try to create your own!

As you are watching similar but different sequences being performed, determine which code sequence matches each video clip.

What clues did you use to match a code sequence to a specific video clip?

Was the end result dramatically different or similar?

When might you want to use one solution over another?



CHALLENGE #1: Look at these code sequences shown below. Then, watch the video clips of the
Which matches which? What clues did you use to match a code sequence to a specific video clip?

Video A	Video B	Video C

Code Sequence 1	Code Sequence 2	Code Sequence 3

Video A matches Code Sequence ____. Evidence: _____.

Video B matches Code Sequence ____. Evidence: _____.

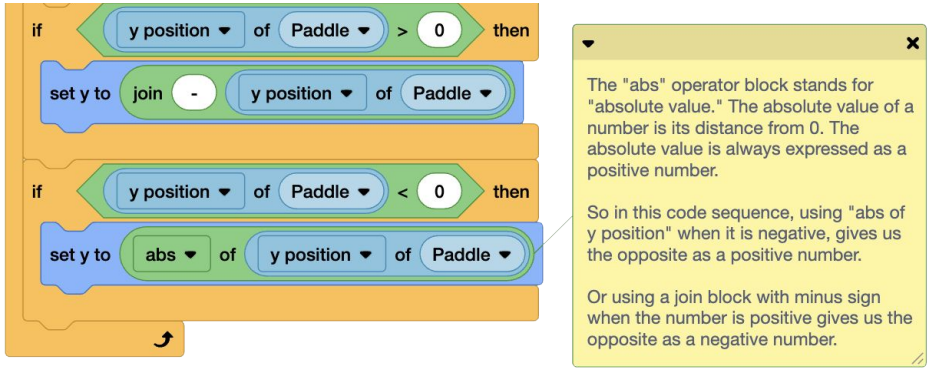
Video C matches Code Sequence ____. Evidence: _____.

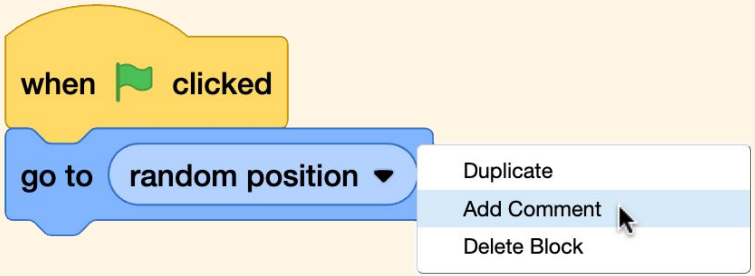
REFLECTION: What differences between similar blocks did you see?

Comment Your Code

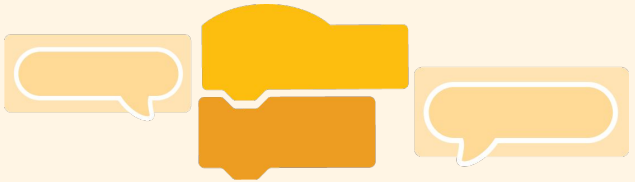
In the script area, adding comments to your code can help others looking at your code to understand it, and also remix it and make changes.

It can also help you remember how your code works when you come back to it later. (See our starter projects for examples).





Right click on script area to “Add Comment.” Use everyday language to explain what a block, or small sequence of blocks, does.



Many Pathways

In the blog post "There's More Than One Way to Code a Cat," Natalie Rusk shared:

Before responding, I find it's helpful to ask what they have in mind, so they can think aloud about the process. Talking out their idea is often enough to help them to figure out what to do next. It's interesting how often students will come up with a way to code that is different than one I might have suggested, but ends up working the way they want...

Focusing on a single pathway not only limits the creative potential of coding, it also limits who becomes interested in coding and decides to learn more.

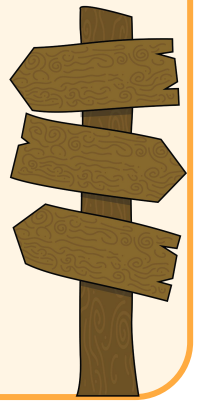
Encourage experimentation

Gently encourage participants to move out of their comfort zones to try new activities and concepts.

Remixing other people's projects is a great way to explore new ideas!

Create a studio or host a Gallery Walk to view and learn from others' solutions.

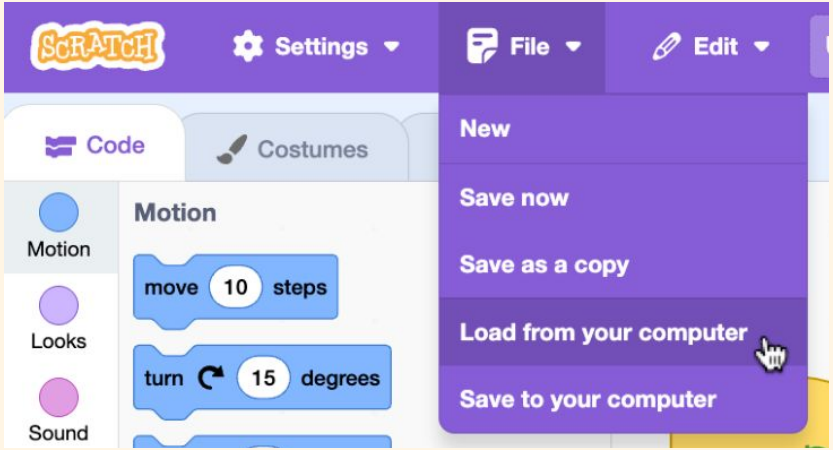
Record reflections in the Scratch Design Journal and/or comment code.



Version Control

Before altering your code, consider keeping a current copy in a safe place, in case you need to refer back to it. This is known as version control.

- Save a copy of the Scratch program file to your computer (File > Save to your computer). Load from your computer to a new project if you'd like to reference it. Name with a date, version number, or helpful wording.
- Or duplicate the code sequence in the script area. Remove any hat block/event block at the top that would make it run. You'll have in your program as a backup you can reference.



Achievery Unit

Coding Challenge

Consider creating a small coding challenge that can have multiple solutions, It can be a fun opportunity to see the different ways learners approach a similar problem and generate classroom or peer-to-peer discussion/reflection.

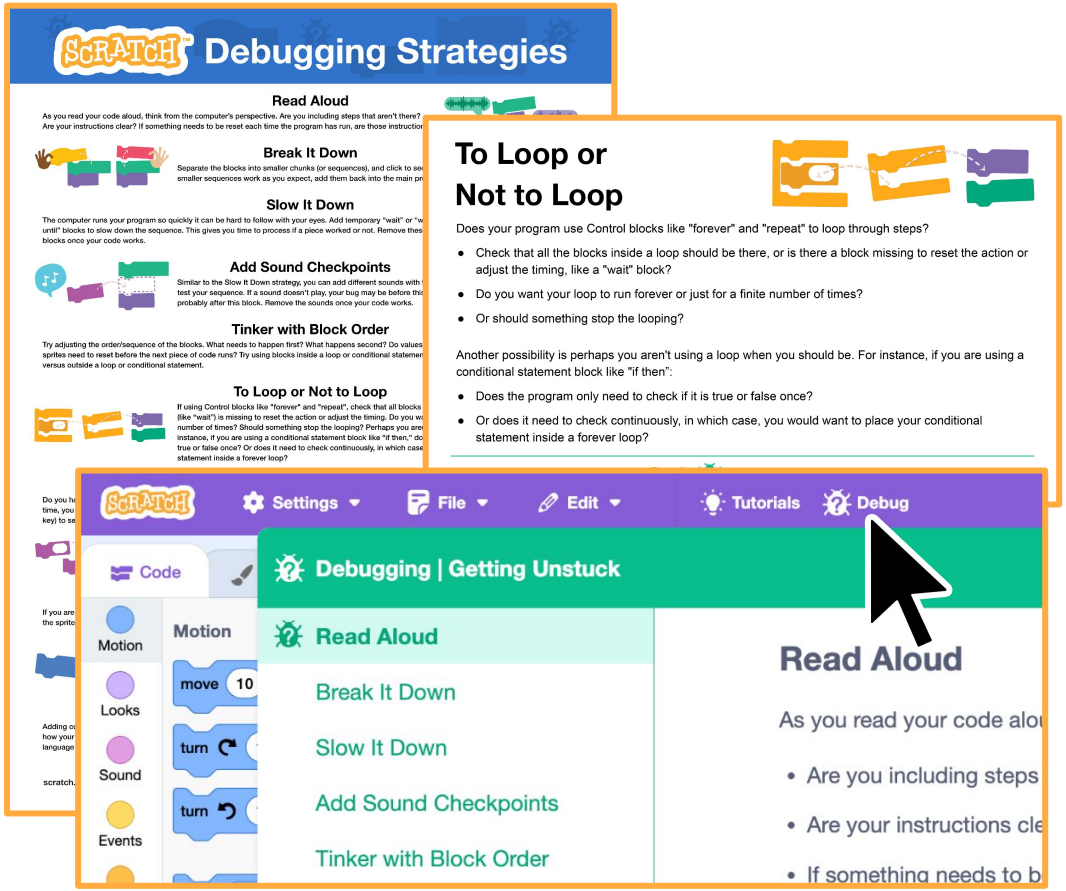
Check out our Achievery Unit "Coding Challenge, Part 1," "Part 2," and "Part 3" as an example. We investigate ways to trigger events at specific times, adding an appearing sprite and visual effects, and making the project interactive by giving the user more control over the action.

Debugging Help

Debugging is finding and fixing issues or errors in your code that result in it not working as expected or at all. Issues are often called bugs.

Our debugging module in the Project Editor shares strategies that are expanded on in our printable posters. They can be found in our Learning Library under Debugging.

The following challenges demonstrate opportunities to use these tips. Hopefully, they'll inspire you to create some challenges of your own!



Achievery Unit

Debugging Challenge 1

Check out our Achievery Unit "Debugging Challenge: The Batter."

Can you debug this sequence and come up with a solution so the batter approaches the plate each time, swings in the correct order, and the costume changes can be seen better?

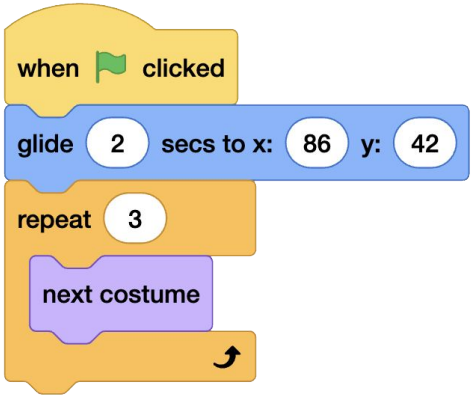
Recreate the batter's code yourself in a Scratch project, or see the code in action here: projects/877707825 (Possible solutions are shared in the unit and inside the project, but there can be many pathways.)

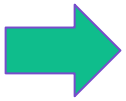
THE ACHIEVERY™
Created by AT&T

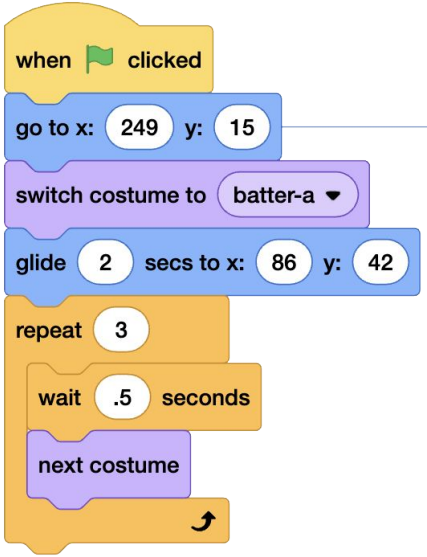
SCRATCH FOUNDATION

Achievery Unit

Debugging Challenge 1 - Possible Solution







Here is one possible solution to debug this code, though other code sequences may accomplish the same goal.

In this example solution, we have a code block to establish the initial starting place and a block to establish the initial costume at the very start of the sequence. Now, the character will be reset each time the green flag is clicked. The program only knows what you tell it, so if it hasn't been told to reset the character, it won't. We've also added a small wait between each costume change to slow down the action.

Why This Challenge?

One common issue Scratchers face is the need to reset the scene when a program runs again.

Scratch scripts don’t automatically reset the position of the sprites, their visibility, their costume, etc., each time the green flag is clicked.

The action of the program can also move quickly, so slowing down the action could make it easier to debug.

Slow It Down and/or Add Sound Checkpoints

The computer runs your program so quickly it can be hard to follow with your eyes. Add temporary “wait” or “wait until” blocks to slow down the sequence. This gives you time to process if a piece worked or not.

And/or you can add different sounds with the “play until done” block at key points to test your sequence. If a sound doesn’t play, your bug may be before this block. If the sound plays, the bug is probably after this block. Remove the sounds once your code works.

Achievery Unit

Debugging Challenge 2

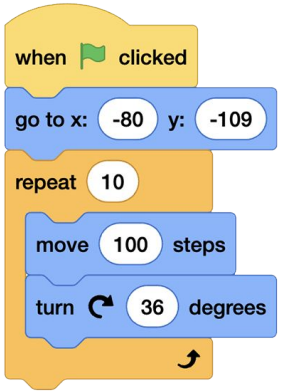
Check out our Achievery Unit “Debugging Challenge: Cat and Bug.”

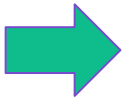
Can you debug this sequence and come up with a solution so the cat spins and then moves to the ladybug, and then the ladybug plays “Hey”?

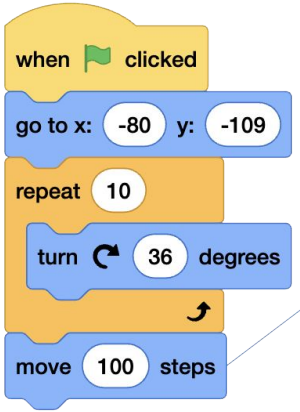
Recreate the code yourself in a Scratch project, or see the code in action here: projects/877745661 (Possible solutions are shared in the unit and inside the project, but there can be many pathways.)

Achievery Unit

Debugging Challenge 2 - Possible Solution

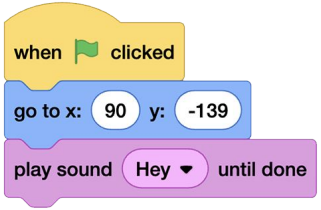


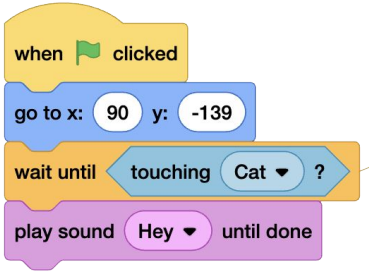




Here is one possible solution to debug this code, though other code sequences may accomplish the same goal.

In this example solution, we placed the “move 100 steps” block outside the loop and after the loop. When the “move” block was inside the loop, that action was being repeated 10 times and the cat was moving before each smaller turn (rather than moving once after all the turns). Order and placement inside/versus outside a loop make a big difference.





Here is one possible solution to debug this code, though other code sequences may accomplish the same goal.

In this example solution, for more exact timing, the conditional statement “wait until touching Cat” is used. Since the cat sprite ends up touching the edge of the ladybug sprite, that event can be recognized and acted upon.

Why This Challenge?

Sequence and block placement are important, especially when using loops.

Timing events between different sprites can be a challenge, and there are multiple ways to approach it. For example, to time the ladybug to play “Hey,” you could:

- add a short “wait” block before the sound plays
- use a conditional statement like “wait until touching Cat”
- use a “broadcast”

Reflect on different approaches. Not one right answer! It depends on goal.

To Loop or Not to Loop

Can a loop speed up your coding process? Check that all blocks inside a loop should be there. Do you want your loop to run forever or a certain number of times? Should something stop the looping?

Tinker with Block Order

Try adjusting the order/sequence of the blocks. What needs to happen first? What happens second? Do values or sprites need to reset before the next piece of code runs?

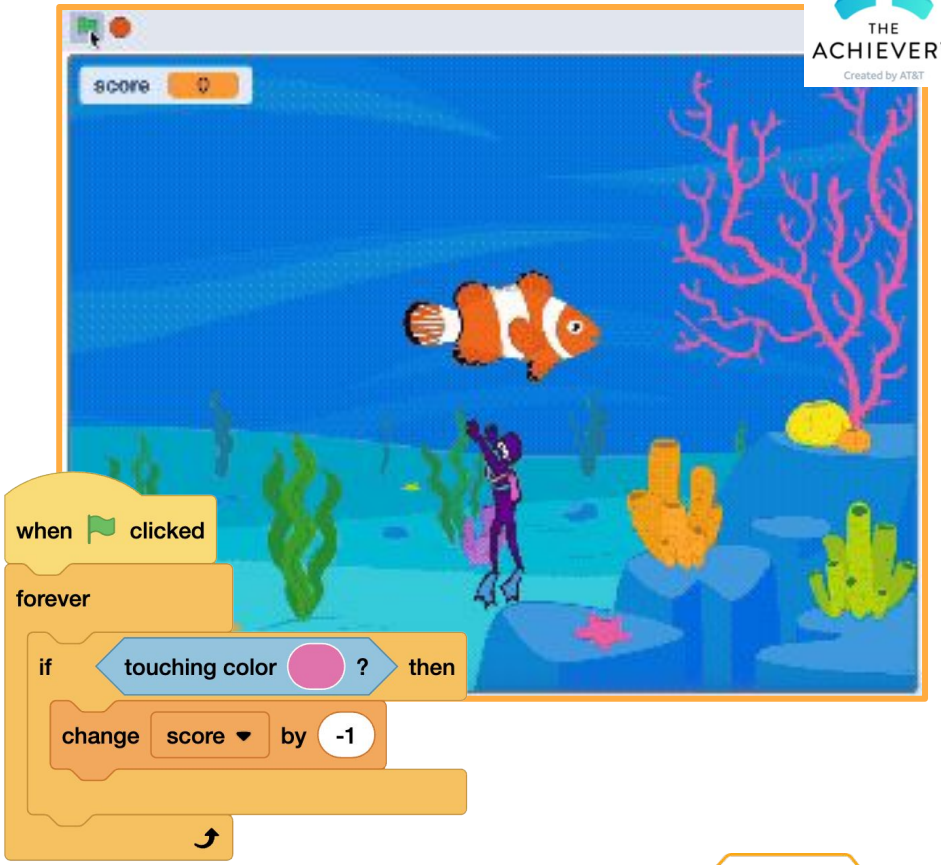
Achievery Unit

Debugging Challenge 3

Check out our Achievery Unit “Debugging Challenge: Color Coral.”

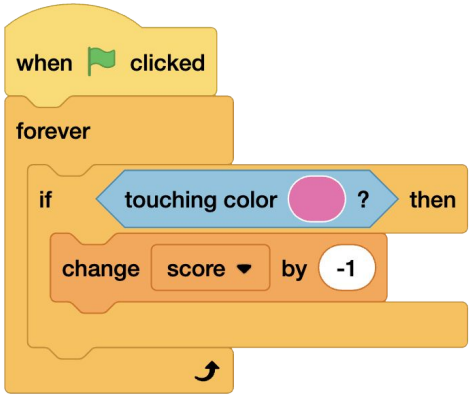
Can you debug this sequence and come up with a solution so the player only loses one point each time the fish reaches the coral, and the player doesn’t lose points if the fish touches the starfish?

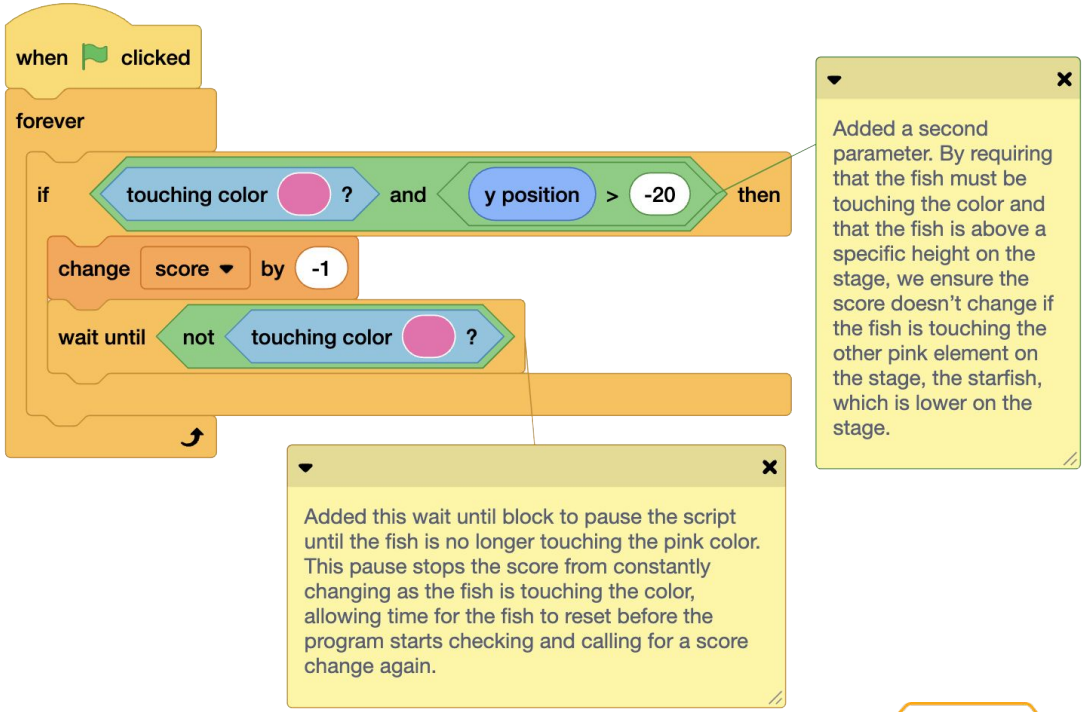
Recreate the code yourself in a Scratch project, or see the code in action here: projects/877715309 (Possible solutions are shared; many pathways.)



Achievery Unit

Debugging Challenge 3 - Possible Solution





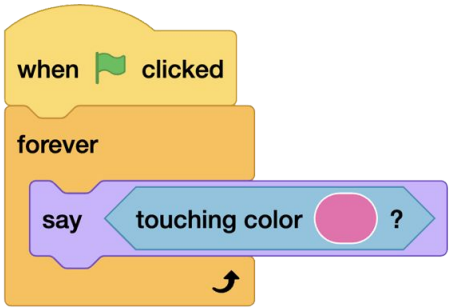
Why This Challenge?

No one solution; supports multiple pathways.

This challenge introduces Boolean blocks, which report values as “true” or “false.”

It is important to understand what the values are, in a given moment in the script, when using variables, reporter blocks, or Booleans.

Tip: While debugging, have the sprite report values using a “say” block:

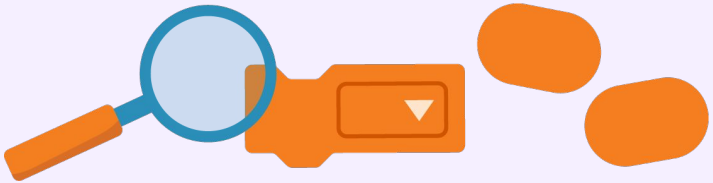


Check the Values

If you are using variables, reporter blocks, or Booleans, check the value at the moment the code sequence is run.

Do/should all the sprites control a variable, or should only one sprite have control?

Where is the value reset? Where is it changed?



Prompts to Try

“I love it! What is it?”

“Can you explain what your program does?”

“Walk me through your code. What does it say?”

“What do you want your program to do?”

“Can you tell me more about that?”

“What new things did you try out?”

“Which category do you think would be helpful?”

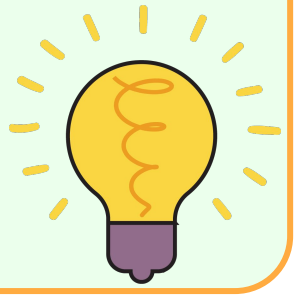
“I don’t know, but let’s find out together/look around the room.”

“What are your next steps for this project?”

Ask questions vs giving answers

It may be tempting to give answers to questions right away. Or you may not have the answers.

Ask questions instead so that learners can arrive at their own answers.



Talk to a Duck!

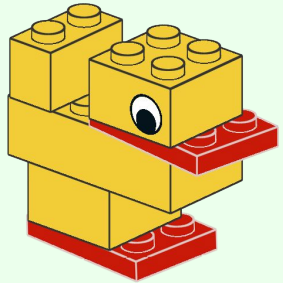
Read your code aloud and think from the computer's perspective:

- Are you including steps that aren't there?
- Are your instructions clear?
- If something needs to be reset each time the program has run, are those instructions included?

Explain what you wanted to accomplish to an inanimate object, like a rubber duck. Often, talking through the code aloud or explaining it to someone else (even an imaginary someone) can be revealing.

Rubber Duck Debugging

A technique in software engineering, wherein a programmer explains each step of their code in natural language which can reveal mistakes and misunderstandings.



Debugging Reflection

What challenges came up for you while creating this project? How did you get unstuck, or what strategies did you use to debug the code?

When you have moments of frustration while debugging, what techniques do you use to manage your emotions or your level of stress?

If you cannot debug a problem on your own, where can you turn to for help or advice? What are trusted sources of information?

When preparing to debug, what are your strategies to keep track of the changes you are making and what works/what doesn't work?

How did you attempt to fix the code? What do you think is going on after your changes?

Many pathways, many solutions: Compare your code with other solutions. Was your solution similar or different? Why did you choose the blocks you did? How might you iterate/what would you add or change if you had two more days?

Debugging Reflection

Debugging is finding and fixing issues or errors in your code that result in it not working as expected or at all. Issues are often called bugs.

Your name: _____ Date: _____

What challenges came up for you while creating this project? How did you get unstuck, or what strategies did you use to debug the code?

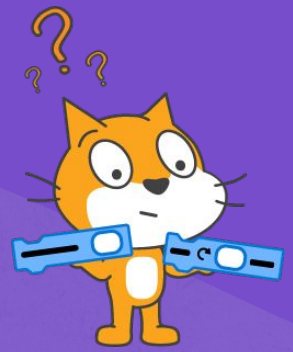
When you have moments of frustration while debugging, what techniques do you use to manage your emotions or your level of stress?

If you cannot debug a problem on your own, where can you turn to for help or advice? What are trusted sources of information?

How did you attempt to fix the code? What do you think is going on after your changes? Explain your solution.

Wrapping Up

Reflecting on Our Session, Resources, Next Steps



Get a copy of our Creative Learning Materials!

In addition to the resources shared throughout these slides:

- See our Learning Library at scratchfoundation.org/learn/learning-library to find lesson plans, coding cards, tutorial videos, and more! See Debugging for tips and posters.
- Scratch Creative Learning Philosophy
- For this session, we shared facilitation tips that have been collected from partners at the Lifelong Kindergarten Group at MIT, Family Creative Learning, and Harvard's Creative Coding Lab (*Getting Unstuck*), in addition to our own facilitation experience.

Find help, inspiration, and information:

- Visit scratch.mit.edu/ideas and scratch.mit.edu/starter-projects
- Click "Tutorials" to see in-editor guides
- Watch tutorial videos on our channel youtube.com/c/ScratchTeam

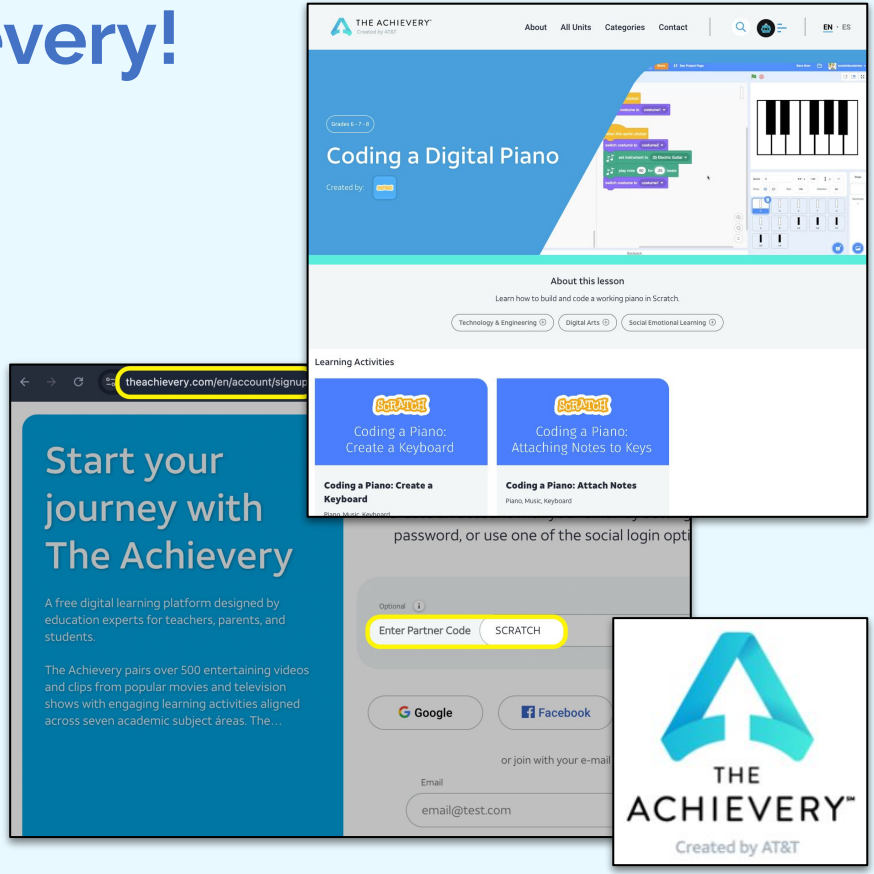
Find Scratch on The Achievery!

The Achievery platform connects K-12 students to a new world of digital learning.

Scratch Foundation has teamed up with The Achievery to provide free beginner and intermediate creative coding lesson plans on a variety of topics for educators, caregivers, and learners.

Sign up (for free!) by using our custom code “SCRATCH” when you register to support our work!

theachievery.com/account/signup



SCRATCH™ FOUNDATION

Thank you!

Be sure to subscribe to our Scratch Foundation YouTube channel for Educators (@scratchfoundation).

Keep an eye on our Event page for additional opportunities: scratchfoundation.org/get-involved/events

Helpful Links:

- Scratch Application: scratch.mit.edu
- Learning Library: scratchfoundation.org/learn/learning-library
- Email Signup: scratch.mit.edu/connect
- Follow us on Instagram and Facebook @ScratchTeam
- Also see our YouTube channel @scratchteam for tutorials