DEBUGGING IN SCRATCH

Sean McManus explains how you can help students to find and fix some of the most common errors in Scratch projects

It's a great feeling when code works the first time, but sometimes it's even more satisfying when it doesn't. One of the best learning opportunities comes when there's a bug, or error, in a program, and we take the time to truly understand the code. Debugging may sometimes be frustrating, but it is an inevitable part of programming.

Educators can focus on guiding students, rather than getting bogged down in code, if they can identify the cause of problems quickly. In this article, I'm going to share some tips on debugging Scratch, based on my experience volunteering at a Code Club. You can use these techniques yourself, and perhaps share some of them with your students.

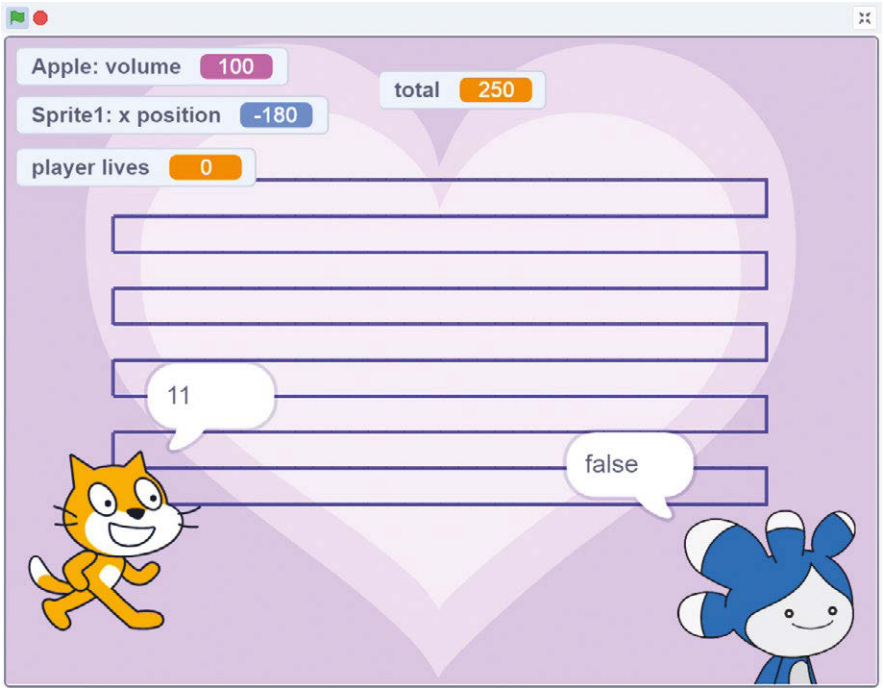
Reproducing the bug

When you're helping a student who has found a bug, the first step is to observe the problem yourself so you're not relying on a second-hand report to understand it. In the classroom, I would usually pull up a chair beside the student and then invite them to show me what they did last time, using the same inputs. I would ask them to talk me through it, including what was surprising about the program's behaviour, so that I could understand what they thought the problem was.

When working remotely, you could ask students to send you a copy of the project, together with instructions on how to reproduce the error, or a video capture showing the bug in action. You should also ask them to explain what they think the code should be doing, and how that's different from what it is doing.

Sometimes there might not be anything wrong with the code. Students might be expecting the script to do something that it's not designed to do, or the code might have missing bits that still need adding from the worksheet. When building their own projects, students might need to rethink the right instructions to use to get the result they want. These reported errors aren't what I would really consider to be bugs, and they can be clarified without any testing or tinkering.

If there is an error, seeing it in action now means you can see the difference when it's fixed later. It's hard to be sure a bug has been fixed if you don't run the code before and after.

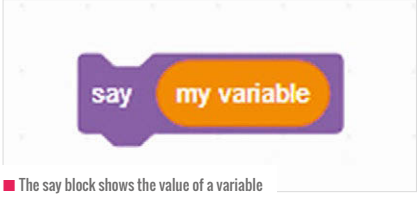


Scratch gives you several ways to see what's going on inside your program, such as using the pen to show where a sprite goes and using say blocks to display the program status

Diagnosing the bug

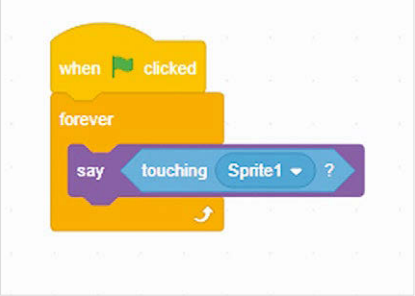
The next step is to diagnose the problem. Scratch is a highly visual language, so you can often see what's happening just by looking at the stage. Sometimes, you might need to get an insight into what's going on inside the project's scripts, or might need to understand what happens at a particular point in the program flow. There are lots of different ways you can do this.

For example, you can click a variable's block in the block palette to see its current value. This doesn't just work for variables: you can use it for x position, y position, direction, costume number or name, backdrop number



The say block shows the value of a variable

A block that makes it say whether a sprite is touching another sprite



or name, size, volume, loudness, and the pointed sensing blocks (such as touching color). The sprite list also shows a sprite's position, direction, size, and visibility.

Variables and some of the blocks have tick-boxes beside them in the block palette that show their values on the stage. These readouts update as the project runs.

You can also use the say block to show the value of a variable at a particular point in the program's execution. This will not update, and will stay visible until you next run the say block on that sprite, or click the green flag.

You can create a simple script to report when a sprite is touching another sprite, or any other condition you want to check for. The script above makes a sprite say whether it's touching another sprite. When it is, the speech bubble text changes from 'false' to 'true'.

The pen can be used to leave a trail of a sprite's movements, and sound effects can be added to indicate when a script starts or reaches a particular point.

To make it easier to find where the bug is, you can break projects down into smaller parts for testing. Instead of clicking the green flag to start a project running, you can click one script in the code area to start it by itself. If you have loops inside loops, you can drag the inner loop out and click it to test it. To see what the script does up to a particular point, you can add a stop this script block, or use a wait one seconds block to pause it. You can also add a wait block inside a loop, such as a movement loop, to slow it down so that you can more easily see what it's doing.

With these techniques, there could be side effects. The timings in the project might be thrown out, for example. If you test individual parts of the script, they might not work without other parts. Nonetheless, breaking the program down like this often helps to isolate where the error is by quickly confirming which bits are working fine.

Image: Scratch Foundation

> "SOME OF THE BEST LEARNING OPPORTUNITIES ARISE WHEN THERE'S A BUG IN A PROGRAM

LEARN FROM DEBUGGING

Students can sometimes fix the bug without learning anything. They might spot the difference between what they've got on-screen and what's in the worksheet, or just tinker until it works. Discussing bugs with the affected students, or with the whole group, helps to ensure that the cause of the error is understood.

Students can also be encouraged to code mindfully, thinking about what the script does as they build it, rather than just copying it from the sheet. That helps to reduce errors, and helps them to prepare for making their own projects.

Fixing the most common bugs

Over time, I found that the same bugs were coming up repeatedly in my Code Club, and it became easier for me to identify what was wrong. I was able to steer students away from some of the pitfalls by leading a group discussion about the potential problem before they built the scripts. In the rest of this article, I'm going to share the most common bugs I saw.

Variable creation

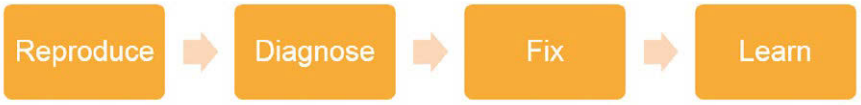
The first concerns variables. A Scratch variable can be created for all sprites (so they can all use its value) or created just for one sprite (so only one sprite can use its value). It's a really useful feature when you're cloning sprites, because if you create a variable for one sprite only, each clone of that sprite has its own separate version of the variable. For example, you can create an alien that stores how many lives it has left in a variable created for that sprite only. Each clone of the alien will behave in the same way, with its own record of its own lives left.

You can see whether a variable is for one sprite or all sprites by ticking the box beside it in the block palette to show it on the stage. If it's for one sprite, it'll have the sprite name beside it.

When a project requires students to create a variable for one sprite only, this often results in errors. This is quite tricky to fix, because you can't just change whether a variable is for one or all sprites. There's an additional complication, too: if you delete a variable you created in error, Scratch also deletes all the blocks that use it, leaving no trace of it in your scripts. If you're using the variable in lots of blocks across lots of scripts, it can be difficult to put those blocks back in again.

Here's a process for changing whether a variable is for one sprite or all sprites:

- Create the variable correctly with a different name to the one originally used. Scratch won't let you use the same name for a variable for all sprites and a variable for this sprite only.
- Update the scripts to use the new variable. The program should now work.
- Delete the variable that was originally created incorrectly, to avoid using it by mistake.
- Now you can rename the new variable to the correct name, if you want it to match a worksheet.



Debugging can be thought of as a four-stage process, from reproducing the fault through to learning from it

Lookalike blocks

The second most common error I encountered was lookalike blocks, responsible for about a fifth of the errors in my group. This was a common error when copying code from worksheets. In particular, students mixed up set and change blocks; x and y blocks, broadcast, and broadcast and wait blocks; and say, and say for two seconds blocks. This can be hard to spot, because at a glance the script looks good, with the right coloured blocks in the right place and most of the text on the block correct, too.

Wrong blocks in wrong brackets

Problems often arise when students put the wrong blocks inside or outside the brackets of the blocks for repeating (repeat, repeat until, forever) or making decisions (if, if... then... else). Wherever there are brackets, it's worth doing a quick check at the top and bottom of them to make sure the right blocks are inside them.

Blocks in the wrong order

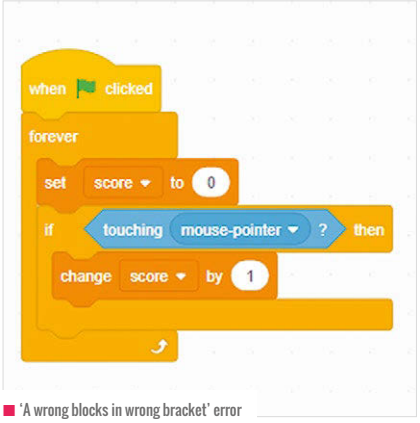
Getting blocks in the wrong order can sometimes stop the program from working as expected. In particular, it's a problem when variables or lists are being initialised after scripts have started changing their values. In the script at the bottom of this page, the score never goes above 1, because it's being reset inside the forever loop, instead of outside of it at the start of the program. This is also a 'wrong blocks in wrong bracket' error. It's easy to see the problem in this tiny script, but it can be harder to spot in longer scripts, or in projects that have multiple scripts.

Rogue spaces

When you're comparing pieces of text, rogue spaces can cause unexpected results. In the script opposite, it looks like these two pieces of text ('hello') are the same. They're not, because the one on the left has an extra space, so the sprite says 'No match!' In a real project, you'd probably have an answer block or a variable in place of one of these hellos, which would make it even harder to spot the problem.

Scripts on the wrong sprite

When using worksheets, some students in my Code Club used to skip straight to the code without reading the instructions fully. That sometimes resulted in them putting scripts on the wrong sprite. Students can copy the script to the correct sprite by dragging it onto that sprite's icon in the sprite list. They will often forget to delete the script on the wrong sprite, though, so you need to watch out for this.



'A wrong blocks in wrong bracket' error

Image: Scratch Foundation

Duplicate scripts

Duplicate scripts arise when students forget to delete a script they originally put on the wrong sprite, or when they are following a worksheet and create a new script instead of adding blocks to an existing script.

If the project runs really fast, that might be because there are two movement scripts running at the same time.

Students might call your attention to the new script they've made, which is perfect, and you might not immediately notice thatthere's an old script that does nearly the same thing on the sprite. You can drag a duplicate script into the block palette to delete it.

Synchronisation and timing issues

Using multiple scripts in Scratch can help to keep code readable, but it can lead to synchronisation and timing issues when different scripts are running at the same time.

Simplification is the key, and reducing the number of green flag scripts often helps. The broadcast and wait block can be used to trigger scripts that should finish before the program proceeds. If the scripts sending and receiving the broadcast are on the same sprite, you can make your own blocks instead.

Some of these errors may seem fairly basic to experienced Scratch users, but they can be frustrating to newcomers, and consume a disproportionate amount of energy. Over time, it gets easier for students to avoid, find, and fix bugs like these, but they can still crop up. I hope that this article provides a handy checklist so educators can help students more effectively. (HW)

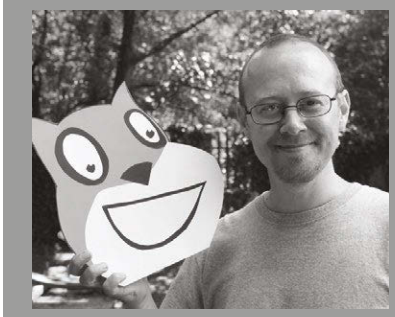
> "STUDENTS SHOULD BE ENCOURAGED TO CODE MINDFULLY, THINKING ABOUT WHAT THE SCRIPT DOES AS THEY BUILD IT

Not changing default numbers

It's easy to overlook that the number in a repeat 10 block needs to be changed to 100 when copying scripts from a worksheet. Similarly, students sometimes forget to change the default values in operator, motion, and other blocks.

'It worked a minute ago' errors

Sometimes a program runs fine the first time, but behaves strangely after that. These errors often result because the project doesn't reset to a known state when it runs. For example, a new game might begin with a main sprite



SEAN MCMANUS

Sean McManus is a copywriter and author specialising in technology. His books include the new 2nd edition of *Scratch Programming in Easy Steps*, *Cool Scratch Projects in Easy Steps*, and *Mission Python*. He posts his Scratch resources at sean.co.uk/scratch (**@musicandwords**).

A DEBUGGING WORKFLOW

This is the kind of workflow a professional programmer might use if they're alerted to an error in a program by a user:

- Reproduce the error so they can see it for themselves
- Diagnose what's causing it, using a mixture of testing and logical thinking
- Fix the problem by updating the code
- Learn from it, to prevent it happening again

There might be loops in the process, because fixing one error might cause or reveal another.

Rogue spaces can cause unexpected results