This file contains my slides and speaker notes for Scratch Conference 2019, where I spoke about debugging. The aim of the session was to help educators to quickly find errors in students' projects, so they can guide them most effectively.

I hope you find these notes useful.

Visit my website for free chapters from my books (including Scratch Programming in Easy Steps; Cool Scratch Projects in Easy Steps; Coder Academy; Mission Python; and Raspberry Pi For Dummies, which was co-written with Mike Cook). You can also find sprite packs and 10-block Scratch demos.

www.sean.co.uk/scratch

Thank you to everyone who made Scratch Conference a great experience! It was organised by the Raspberry Pi Foundation, supported by the Scratch Team at MIT, and sponsored by Google, Epam and Amazon. The staff, volunteers, contributors and participants were generous with their time and expertise.

Scratch Conference Europe

23 – 25 August 2019

#ScratchEurope

"Piled Higher and Deeper" by Jorge Cham
www.phdcomics.com
Used with permission

I'm going to talk about **debugging** today, the **practice of finding and fixing errors in programs**.

For **professional programmers**, debugging can be a **frustrating experience**.

But **debugging is a part of programming** that programmers of all levels have to live with.

Depending on how sophisticated your code is, and how well you're programming on that particular day, you might see more or fewer errors.

But there are usually some. And maybe programming would be less interesting if it always worked first time?

Even though the scope of the projects is much smaller in Scratch, we should remember that it can also be **unsettling for some students when the code doesn't "just work"**.

But this is a **natural part of programming**.

And it can sometimes even be a **fun part** too, becoming a **detective** to work out what's going on inside the code.

It's certainly extremely **satisfying to fix puzzling errors**.

This 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;
- and then **learn** from it to prevent it happening again.

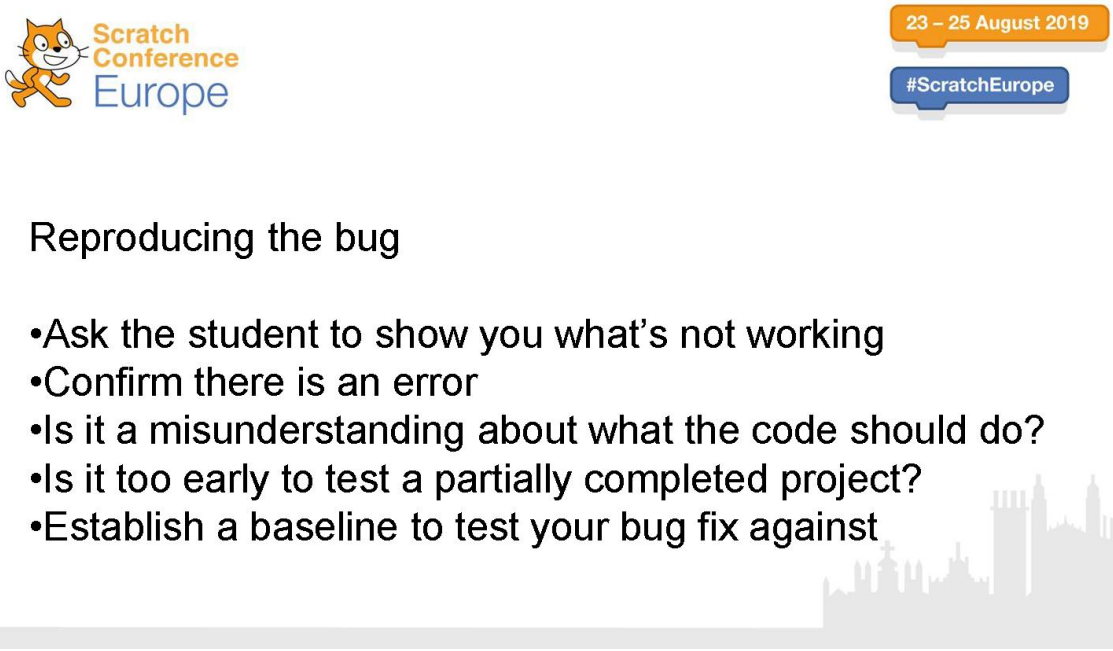
It's **not necessarily a linear process like this**: fixing one error might cause another one, so there can be loops in here.

But this is a **good simplification for a debugging process we can use in Scratch**, and perhaps **share with our students** too.

From an **educator**'s point of view, if a child is struggling with something that doesn't work, **you want to be able to quickly identify the problem.**

That enables the educator to **guide the student, or the whole group, most appropriately,** rather than spending too much time immersed in working out what's gone wrong.

My goal here is to **help you to find the bugs more quickly, by sharing some of the most common errors I've seen, and some practices I've found useful**.



As I said, the first step is to reproduce the bug

Observing it yourself means you're **not relying on a second hand report to understand it**.

So we can **ask the student to show us what's not working**, including using **the same data or inputs** if appropriate. **"Talk me through it**."

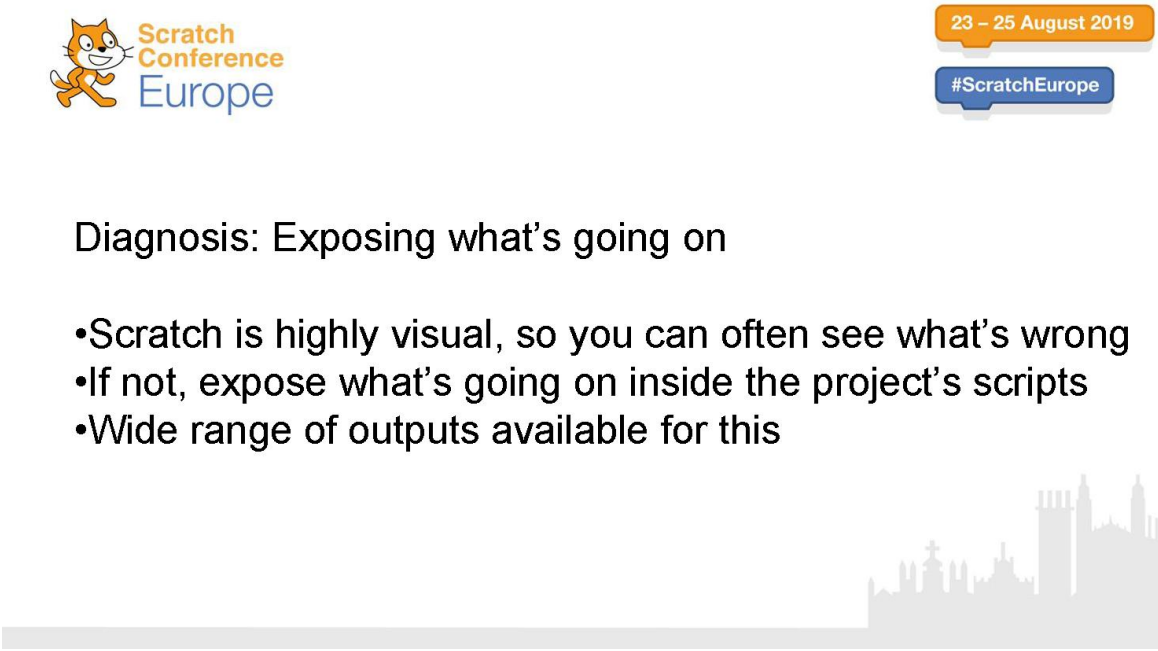
Sometimes there might not be anything wrong with the code:

- Students might be **expecting the script to do something that it's not designed to do**.
- It might not work because the **project isn't finished yet**, and they still need to enter the rest from the worksheet.
- In their own projects, they might need to **rethink the right instructions** to use to get the result they desire.

So those reported bugs aren't really bugs, and can be **clarified without any testing or tinkering.**

If there is an error, seeing it in action now means we 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.



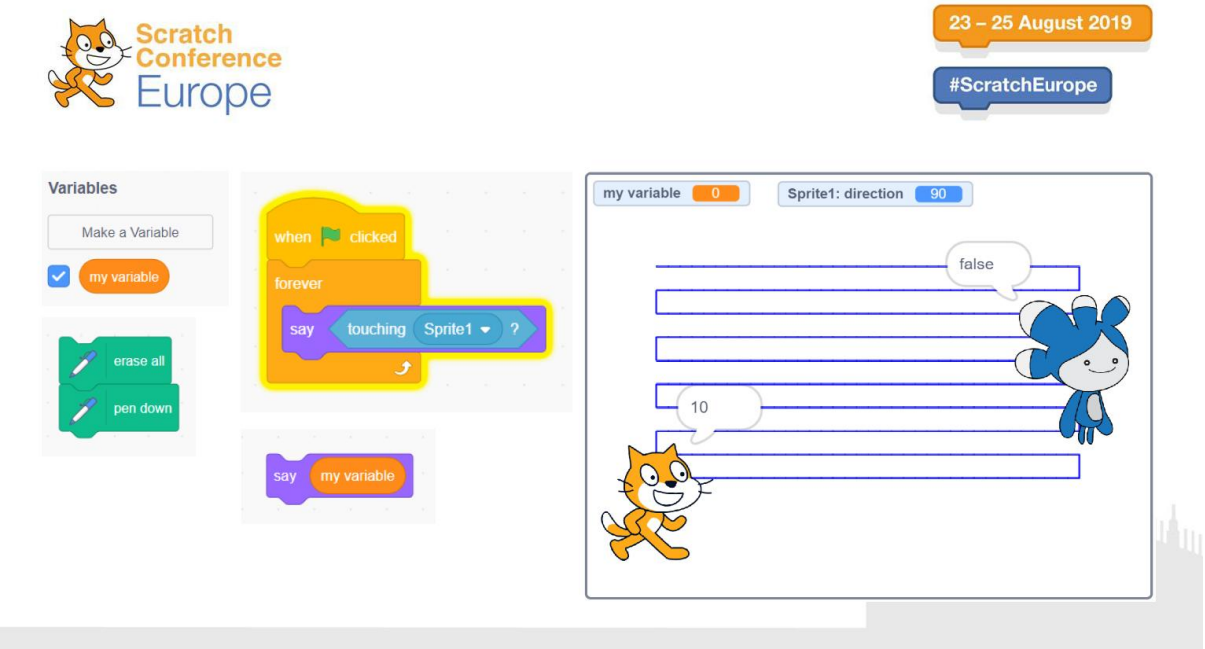
The next step is to **diagnose** the problem.

Scratch is a highly visual language, so you can often see what's going on.

It's happening on the Stage in front of you.

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 script.

There are lots of different ways you can do this.



Here are some examples on the left for different ways you can expose what's going on in the program.

The Stage on the right shows all these outputs enabled.

- You can **tick the box beside a variable** in the Blocks Palette to show it on the Stage
- You can do the same for a **sprite's x position, y position or direction**
- You can get a sprite to **say a variable value** at a particular point in the program, so you can easily see its value at that time. This **doesn't change on the screen when the variable changes** unless you run the say block again.
- You can use the **pen to trace a sprite's movement** pattern around the Stage so you can see its exact path
- You can create a **script that reports on** when a sprite is touching another sprite.
- Note that you can **use the say block with a Boolean block here**: that is, the pointed block fits into the rounded hole of a say block.
- You could also **add a sound effect** when a script starts or reaches a particular point, so you can hear it gets there.
- It's worth remembering too that **scripts have a glow around them when they run**, so that can be a hint about which scripts are running in a project.
- [One participant at the event suggested adding **logging,** perhaps by adding a variable or other piece of information to a list at various times as the program runs, to keep a record of its values over time.]

Scratch Conference Europe

23 – 25 August 2019

#ScratchEurope

Diagnosis: Breaking the script down for testing

- Click a script to run just that script
- Pull out an inner loop and click it to test just that loop
- Add a breakpoint: Stop or pause the script at different points
- Use a wait block to slow a loop down
- Beware of side effects

You can also break the project down to test it in smaller chunks to help you find where the error is. (Similar to **unit testing** in professional coding environments).

- Instead of clicking the green flag to start all the scripts, you can **click one script** in the Code Area to start just that script.
- If you have loops inside loops, you can **drag the inner loop out and click it** to test that by itself.
- To see what the script does up to a particular point, you can **add block to stop it or pause it there.**
- You can also add a **wait block inside a loop**, such as a movement loop, to slow it down.

With these techniques, and the output ones a moment ago, you need to remember there might be **side effects**.

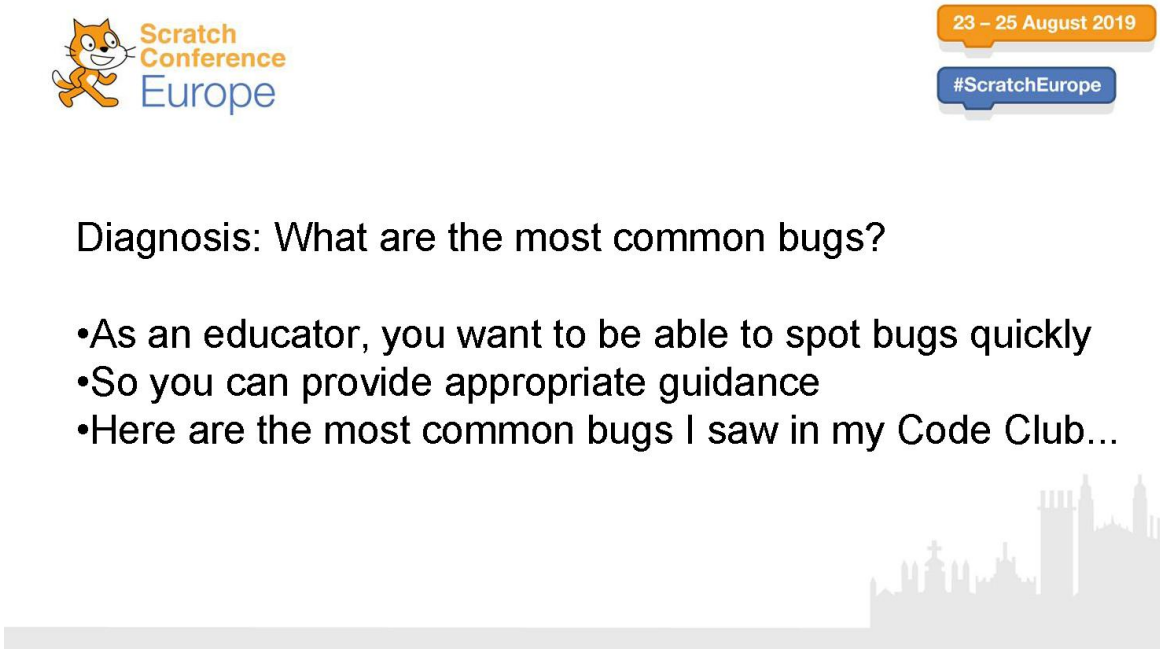
It might throw out your **timings**, for example.

If you test individual parts of the script, it **might not work** without other parts.

But it can help to break the program down, see which bits are working, and which bits might be harbouring the bug.

These are **techniques you could share with your students**, too, as they become relevant in their learning, to help them with their own debugging.

Perhaps **see which ones they can come up with**, if asked to find out what's going on?

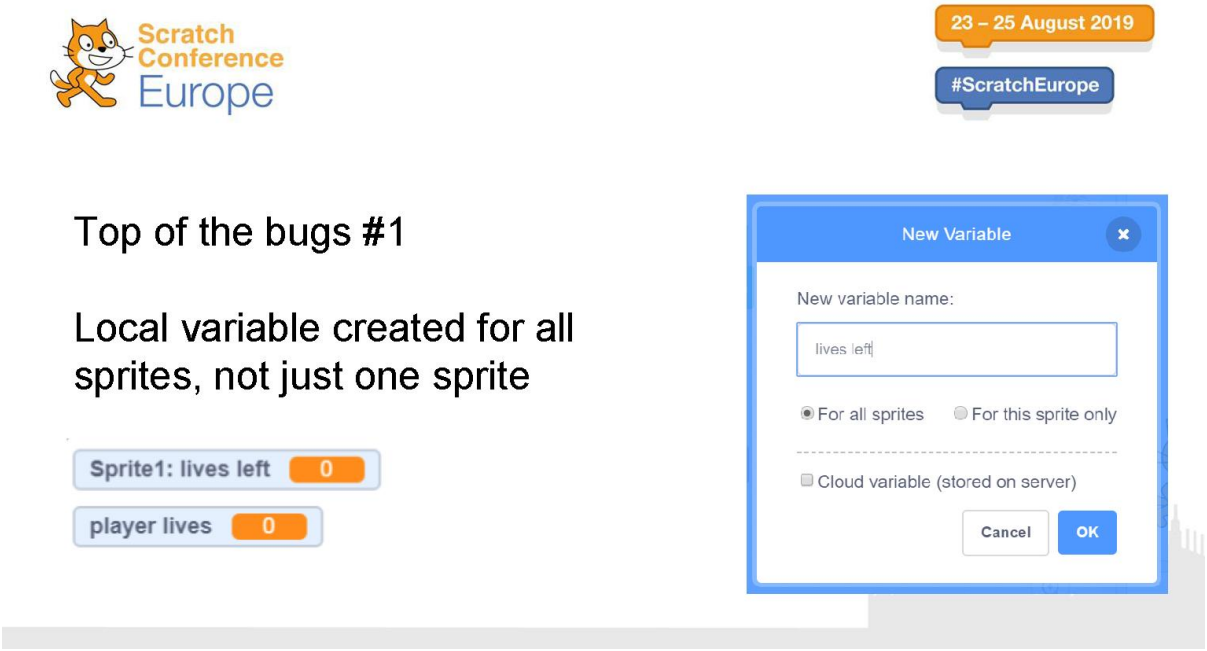


I'm going to share the **most common bugs I came across in my Code Club**.

My **goal here isn't to point out the basics of Scratch**, (you probably know these already)

But to give you a **crib sheet** so you can quickly look for what is most likely to be wrong.

These **bugs accounted for most of the errors that came up repeatedly in my Scratch sessions**, so they were the first things I'd typically check for.



Scratch enables you to create

- **a variable that can be used by all sprites**
- **Or a variable that only belongs to one sprite**

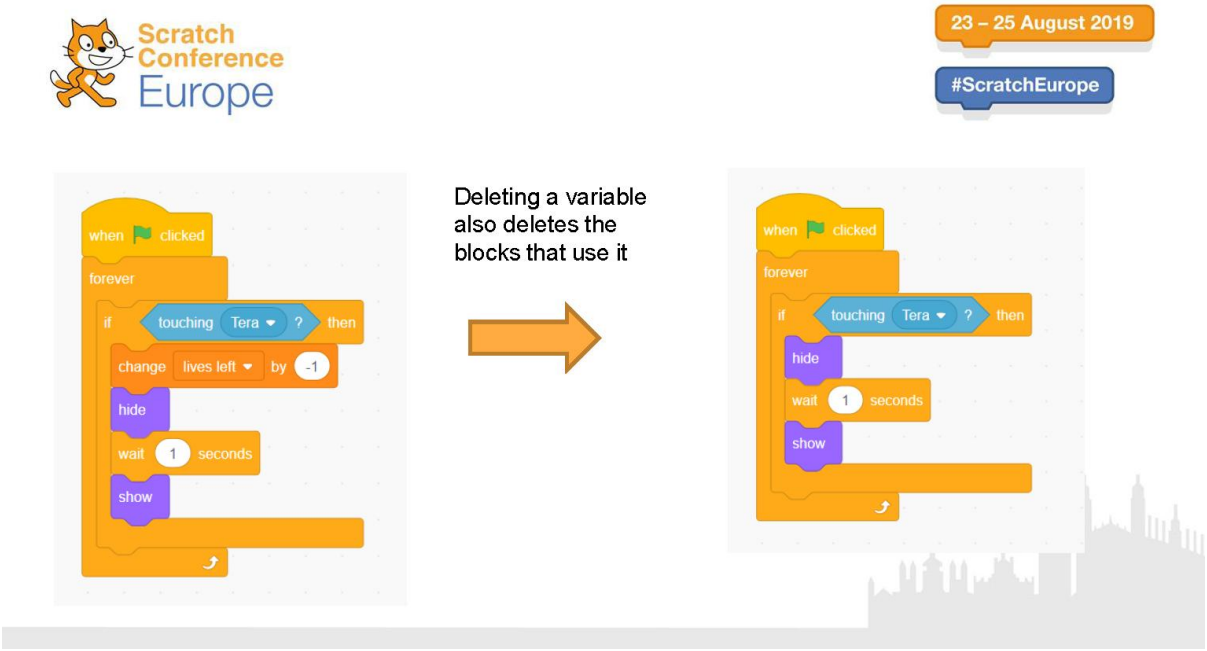
It's a **really useful feature when you're cloning** sprites. You could have lots of aliens the player has to shoot, for example, and each of those aliens could have its own variable called *lives left* that behaves independently.

In my experience, **students often had problems with this**, and often made mistakes in creating the variables.

You can **see whether a variable is for one sprite or all sprites** by ticking the box beside it in the Blocks Palette to show it on the Stage.

If it's for one sprite, it'll have the sprite name beside it.

This is quite a **tricky bug to fix, because you can't just change a variable from one type to the other**.



There is **another gotcha too**:

If you delete a variable you created in error, it also deletes all the blocks that use it in Scratch 3.

As you can see here, when I deleted the variable, the block to change it disappeared.

That's good because it **preserves the integrity of the script**: you don't have blocks in the script using non-existent variables (as you did with Scratch 2).

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.

So a better approach is to **create the variable correctly with a different name**. You can't use the same name twice in the same scope.

Update the scripts to use that new variable.

And that will fix the program.

To **tidy it up**, you can **delete the variable that was created incorrectly**.

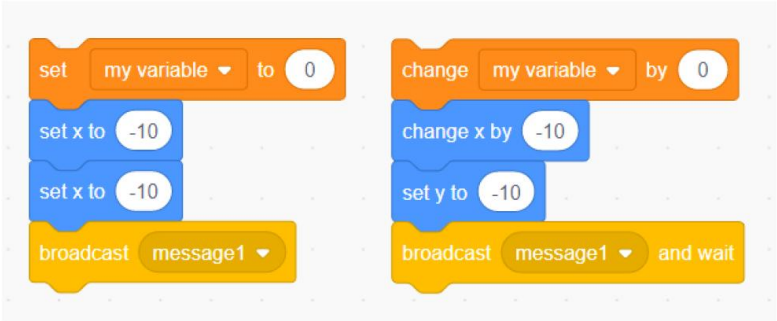
And that means you can **rename the new variable** to the correct name.

Scratch Conference Europe

23 – 25 August 2019

#ScratchEurope

Top of the Bugs: Lookalike blocks #2



The next most common error in my club was using **lookalike blocks**.

This was a common error in **copying code from worksheets**.

This accounted for about 20% of errors in my group.

At a glance, these two scripts look pretty similar, but the blocks are functionally very different.

In particular, students mixed up:

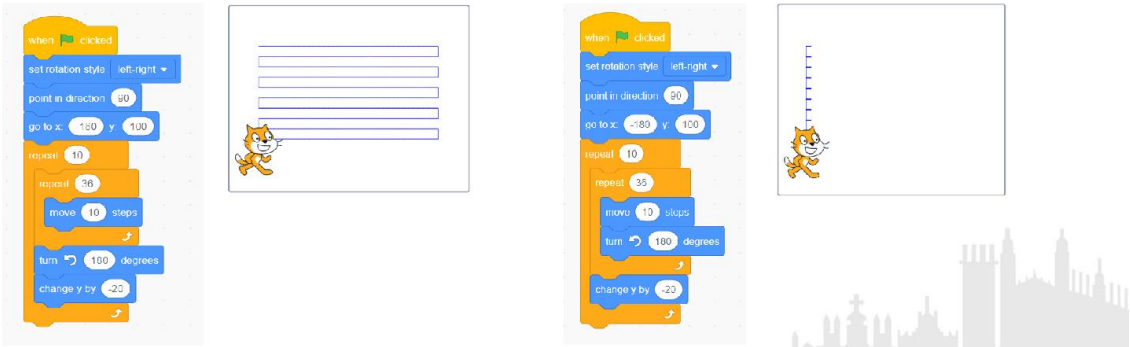
- **Set** and **change** blocks
- **X** and **y** blocks
- **Broadcast** and **broadcast and wait** blocks
- **Say** and **say for 2 seconds** blocks.

Scratch Conference Europe

23 – 25 August 2019

#ScratchEurope

Top of the Bugs #3: Wrong blocks in wrong brackets



Number 3 in top of the bugs is putting the **wrong blocks in the wrong brackets**.

The script on the left creates a Space Invaders style pattern, moving left and right and advancing down the screen.

I've put the pen down so you can see the movement pattern, but I've left the pen blocks out of the script for simplicity.

The script on the right has one block at the end moved inside the inner loop, the Turn 180 degrees block.

As a result, the sprite never walks across the screen.

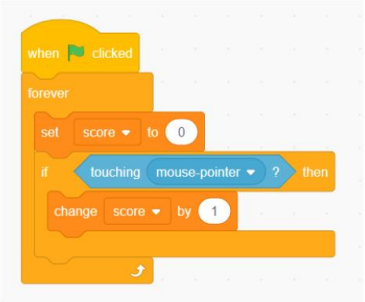
So, **it's worth checking at the top and bottom of brackets to check which blocks are inside them**, and make sure they're correct.

Scratch Conference Europe

23 – 25 August 2019

#ScratchEurope

Top of the Bugs #4:
Blocks in the wrong order, especially variable blocks



Often you have a lot of flexibility over the order that the blocks go in.

At the start of a program, it doesn't really matter what order you initialise your variables in, or set up your sprite defaults in.

However, **getting blocks in the wrong order can stop the program from working as expected**.

In particular, it's a problem when **variables or lists are being initialised after they have started to be used**.

In the example on the left, we end up with a list of four people instead of five.

In the **example on the right, 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**.

These are tiny examples, but it's **harder to spot when you have more sophisticated projects perhaps with multiple scripts**.

Scratch Conference Europe

23 – 25 August 2019

#ScratchEurope

Top of the Bugs #5: Rogue spaces

Bug #5.

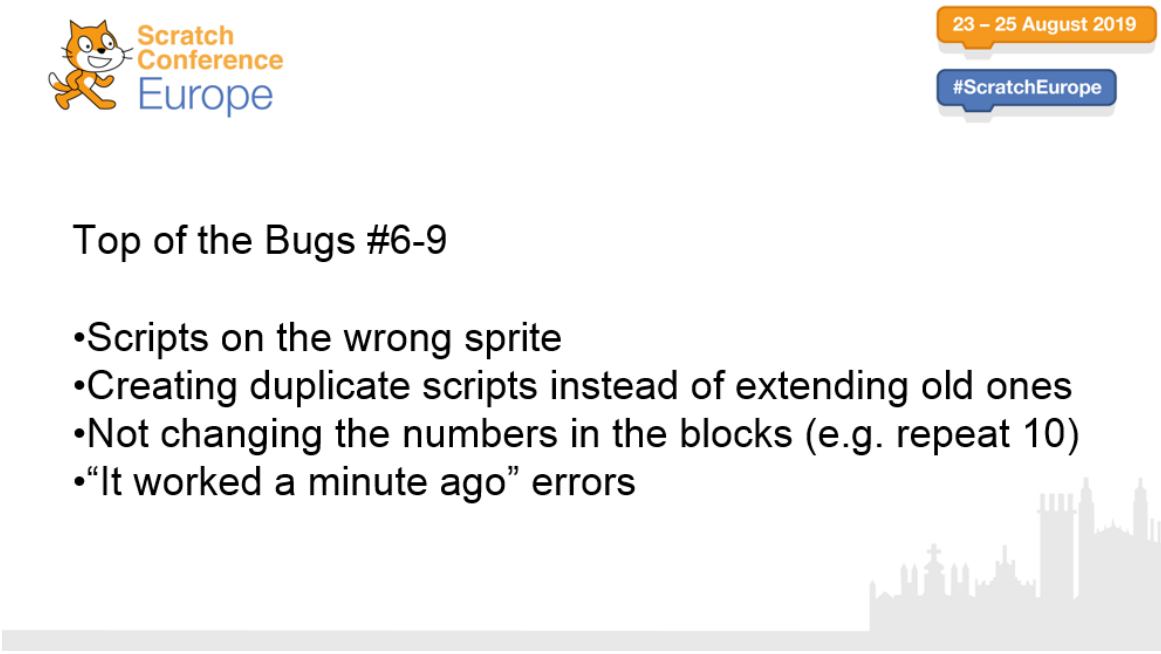
This can be really confusing.

It looks like these two pieces of text are the same, but they're not because one of them has an extra space.

Of course, this script is a simplification. You'd **probably have an answer block or a variable in place of one of these "hellos"**.

But **that makes it even harder to spot**.

So, if the script uses text like this, **check for rogue spaces**.



- **Bug #6** is putting scripts on the wrong sprite.
- Sometimes it was because the students weren't reading the worksheets closely enough and **just skipping to the code chunks**. Giving a heads-up at the start can help to avoid this.
- This was fairly easy to identify and fix, but what tended to cause problems was that the student would sometimes **add the script to the correct sprite, but not remove it from the incorrect sprite**.
- [Check also for scripts added to the Stage by mistake. Thanks to Marc Grossman for that tip]

- **Bug #7** results when students **don't read the instructions** in a worksheet correctly. So **instead of adding blocks to an existing script** they've made, they **create a new script**.
- **If the project runs really fast**, that might be because there are two movement scripts.
- It's worth being aware of this, because students might call your attention to the **new script they've made, which is perfect,** and you **might not immediately notice there's an old script** that does nearly the same thing on the sprite too.

- **Bug #8** is **leaving the default numbers** in the holes in the blocks, so a block is a **repeat 10 instead of repeat 100**, for example. Also: **Not changing the menu values in blocks**.

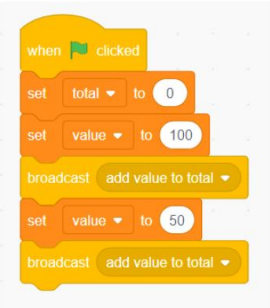
- **Bug #9** I call "**It worked a minute ago!".**
- This happens when the **program runs fine the first time**, but **behaves strangely after that**.
- These errors often result because the **project doesn't reset to a known state** when it runs.
- For example, **sprites might be hidden from the end of the previous game**.
- Or the **score might start at 100** because that was the score at the end of the previous game.
- This is a common error when students create their own projects.
- It can be fixed by setting the visibility, position, costumes, background, variable values, or anything else the project uses to default values at the start of the project.

Scratch Conference Europe

23 – 25 August 2019

#ScratchEurope

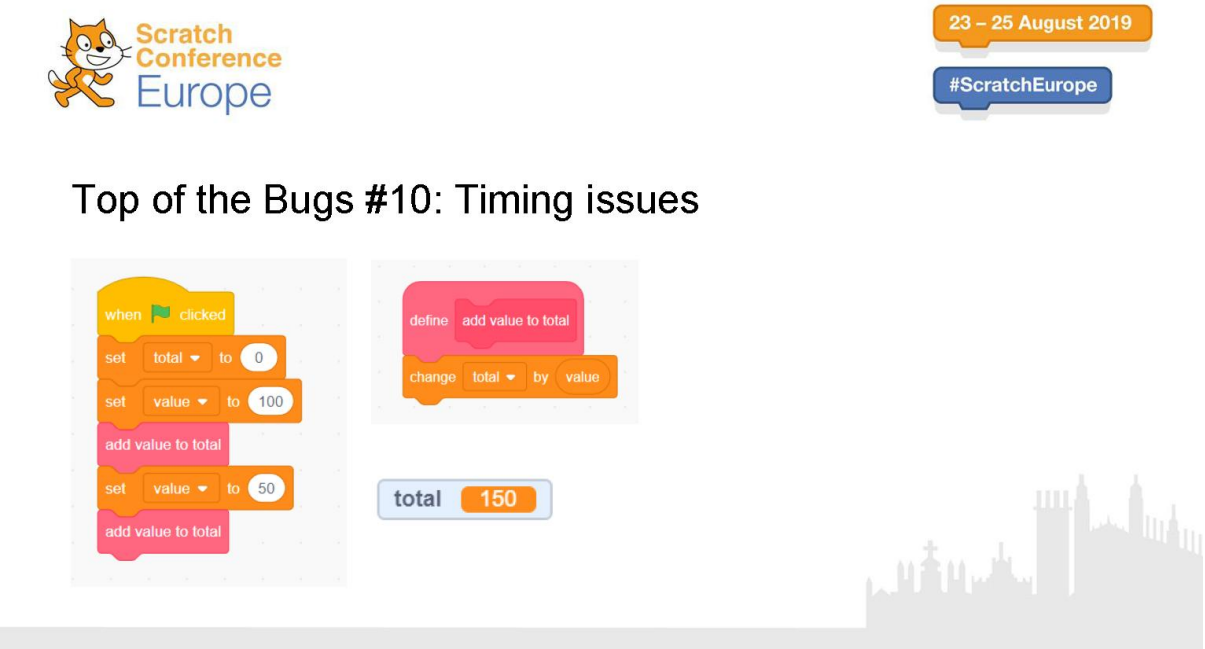
Top of the Bugs #10: Timing issues



With thanks to Cliff Davies, who posted this example in Facebook / Teaching with Scratch. Used with permission.

- The last problem I'd like to discuss relates to **timing**.
- This script sets up a total variable as 0, and a value variable as 100.
- Then it uses a broadcast to another script to add value to total.
- Then it sets value to 50
- And does the same
- **So intuitively, the value of total should be 150 at the end. 100 plus 50**.
- **But the end result is actually 50 because of timing issues within Scratch**.
- My guess is that the **set value to 50 block runs before the broadcast is received,** and that **you can only have one broadcast of the same name at a time going on**, so the when I receive script only runs once.
- I tested this by adding a variable that increased each time the **small script** ran, and it **only runs once**.

To fix this, you can use a **broadcast and wait** block **instead of the broadcast** blocks, to make sure that the script that changes the total variable runs before the main script moves on.



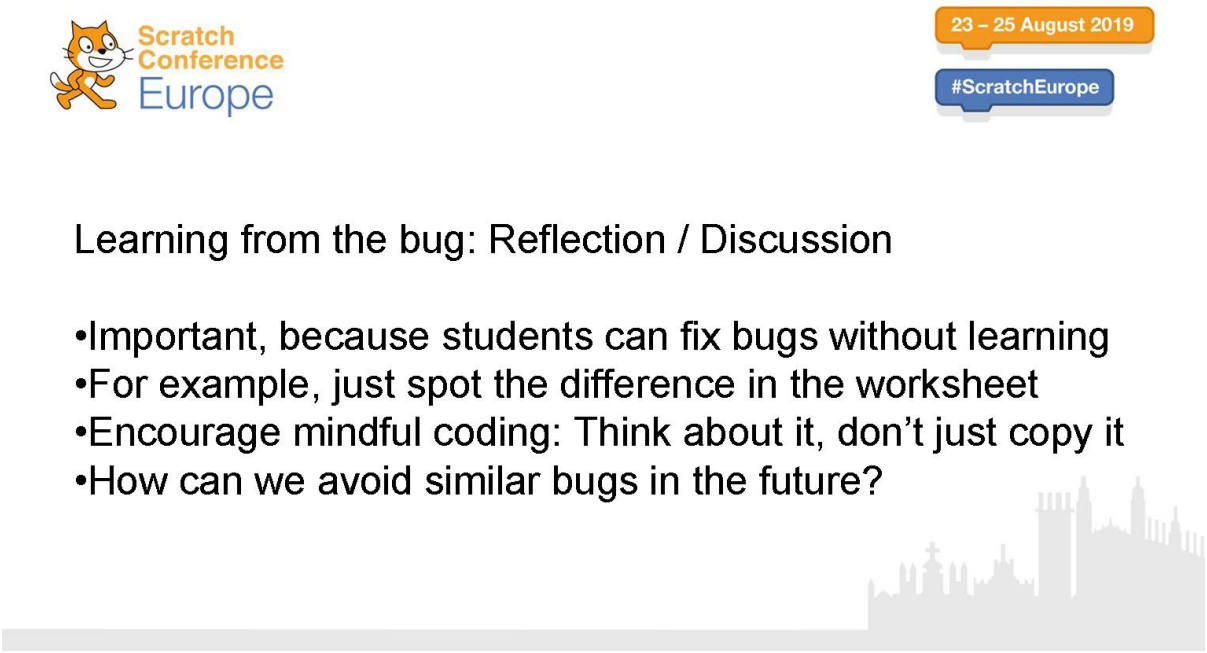
A better solution would be to define your own block, which has the same effect of making sure the total is added up before the script moves on.

This is an artificial example, but **synchronisation problems are fairly common in more advanced projects when using events**.

If you have multiple scripts started by the green flag, you can't be certain which will start first.

So you **need to explicitly coordinate between them**.

By defining blocks, or using broadcast and wait.



Now we've fixed the bug, the final step in our process is learning from it.

Debugging creates a highly memorable learning opportunity.

But I think it's important to call out this learning stage separately, because **students can fix a bug without actually learning**.

They could just **spot the difference** between what they've got on screen and what's in the worksheet, or just **tinker until it works**.

So we can **encourage them to code mindfully**, and think about what the script does as they build it, rather than just copying it from the sheet.

Many of these errors are less likely to happen if the student is thinking through the program as they build it, and actually questioning what each part does, even when using worksheets.

Students can be incentivised to do this by knowing they will have the opportunity to build their own projects later on.

It gets **easier for students to do that additional mental work as they become more familiar with more of the blocks**, and can **build the scripts without having to think so much about where to find the blocks**.

Bugs can be an opportunity for discussion with the affected students or the group as a whole.

Perhaps asking them **"how can we avoid similar bugs in the future?"**

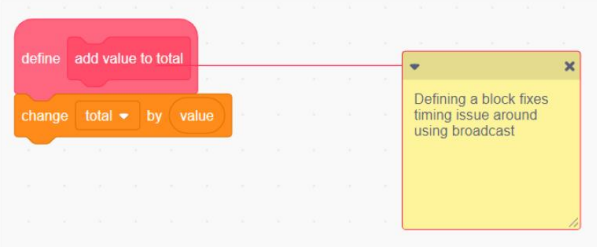
Sometimes **students can help each other to fix bugs**, which is a great example of the cooperation that Scratch helps to foster. But it's worth keeping an eye on those discussions to see if there are **any learning opportunities the group leader can reinforce**.

Scratch Conference Europe

23 – 25 August 2019

#ScratchEurope

Learning from the bug: Commenting code



Those lessons can be cemented by **encouraging students to comment their code**.

Commenting is often used in text based languages such as Python.

In Scratch, you can right-click the script to add a comment.

Commenting provides an **opportunity for students to summarise their understanding**, and **helps them when they return to code later**.

Scratch Conference Europe

23 – 25 August 2019

#ScratchEurope

- Free book chapters and more at www.sean.co.uk
- https://twitter.com/musicandwords
- https://scratch.mit.edu/users/seanmcmanus/

Thank you for your interest in my talk [either in the room, or by downloading these notes afterwards!].

You can now find my Scratch resources -- including sprite packs, 10 block coding demos and free book chapters -- on my website at www.sean.co.uk/scratch

I hope you found this useful!