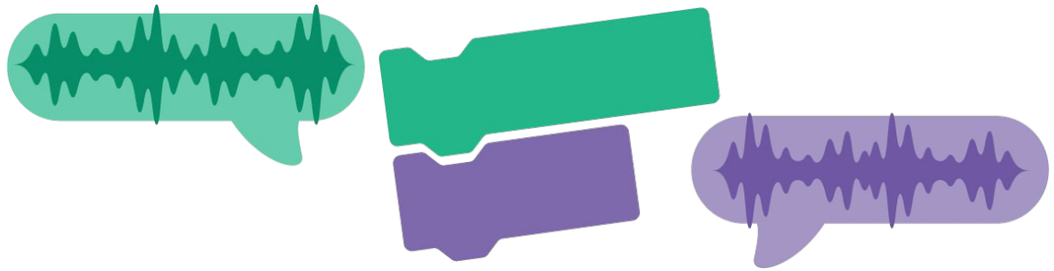
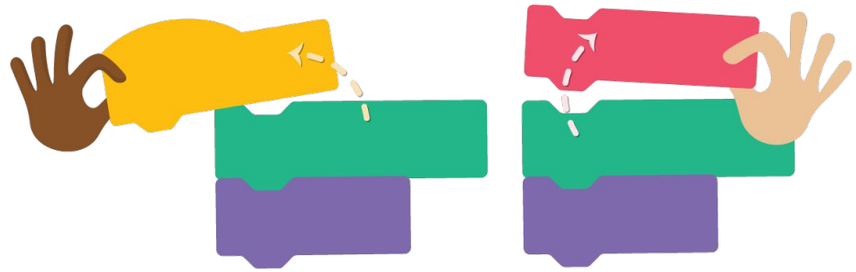


Read Aloud or Explain the Code Step-By-Step, to Yourself or to Someone Else

Have you ever read something you wrote down outloud and discovered grammatical or punctuation errors? Reading through your code sequence aloud step-by-step can produce similar results. As you are reading your code aloud, think from the computer's perspective. Are you including steps that aren't actually present? Are your instructions clear? If something needs to be reset each time the program has run, have you included those instructions in your sequence?

Break Long Sequences Apart into Smaller Pieces

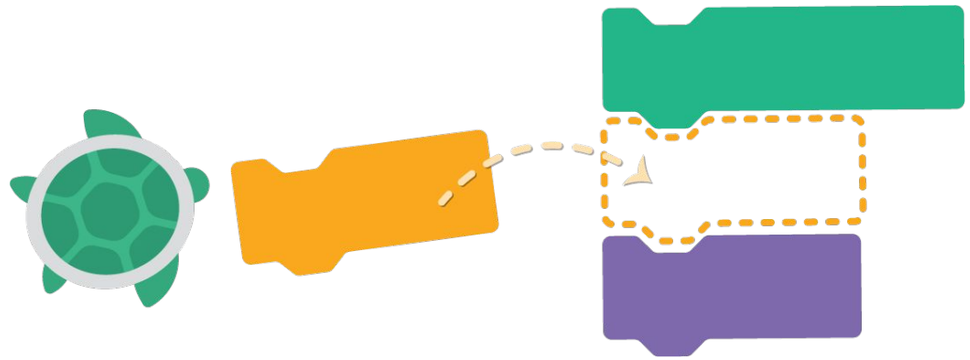


A good practice is to develop your program incrementally: code a bit, test, then code a bit more. Then, if a bug appears, you can more easily identify where it was introduced (likely, in the last piece you added). If your sequence is already written, however, you can still use this idea to debug by breaking long sequences apart.

Separate the blocks and click on each individually to see what it does, or break a long sequence apart into smaller sequences of two to four blocks, in order to narrow down where the bug appears. This process is called decomposition, which is when you break down a complex problem or system into smaller parts. Smaller parts are often more manageable to edit and tinker with, and easier to understand.

Click on a single block or sequence of blocks in the script area to run just that piece of the program. When you know a piece is working as expected, add it back into the main program and move to the next piece to test.

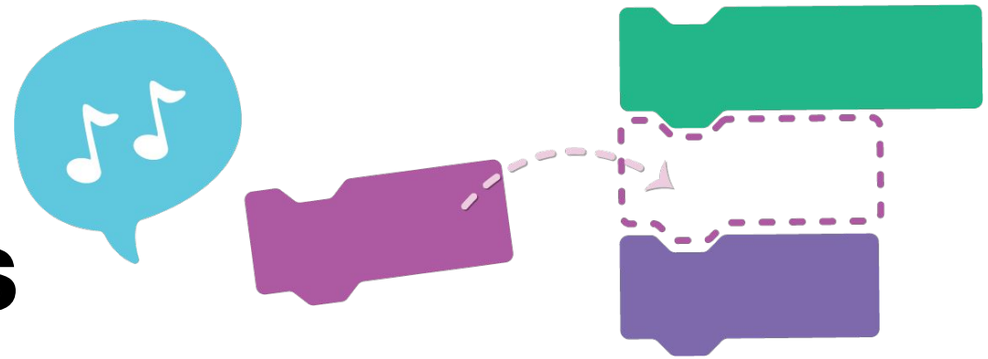
Add Temporary Waits to Slow Down the Action



The computer can run your program so quickly that you may not be able to see all the action with your eyes.

Add in temporary “wait” or “wait until (key pressed)” blocks to slow down the sequence and give you time to process if a piece worked or not.

Once you know the code is working, you can remove these waits.

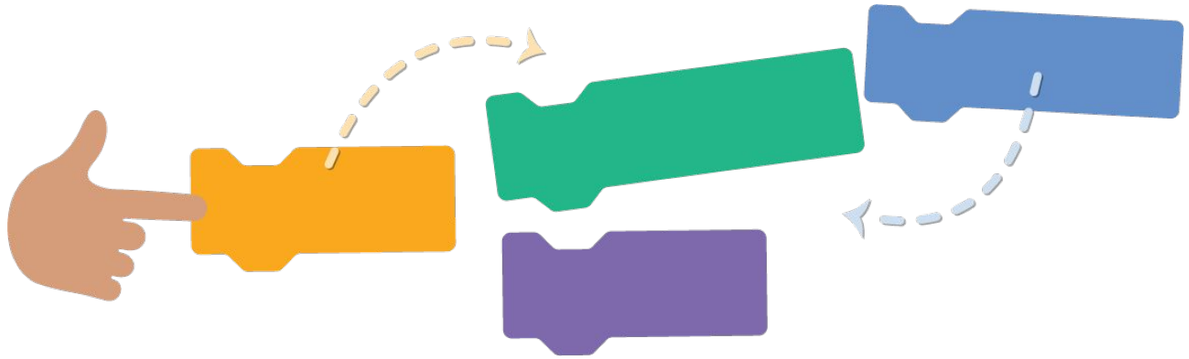


Add Temporary Sounds at Key Checkpoints

Similar to the strategy of adding temporary waits/pauses in your code, identify key points in your sequence and add a different funny sound to play before it runs using the “play until done” block.

If a sound doesn’t play, you know the bug most likely occurs before the sound. Or if a sound plays and then the bug occurs, you know the bug most likely occurs after the sound. Once you know the code is working, you can remove these sounds.

Tinker with the Block Order

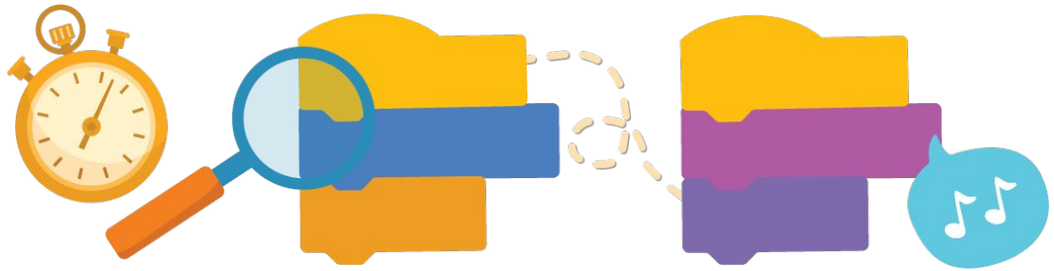


Try adjusting the order/the sequence of the blocks. Ask yourself:

- What needs to happen first?
- What needs to happen second?
- Do values or sprites need to reset before the next piece of the code runs?

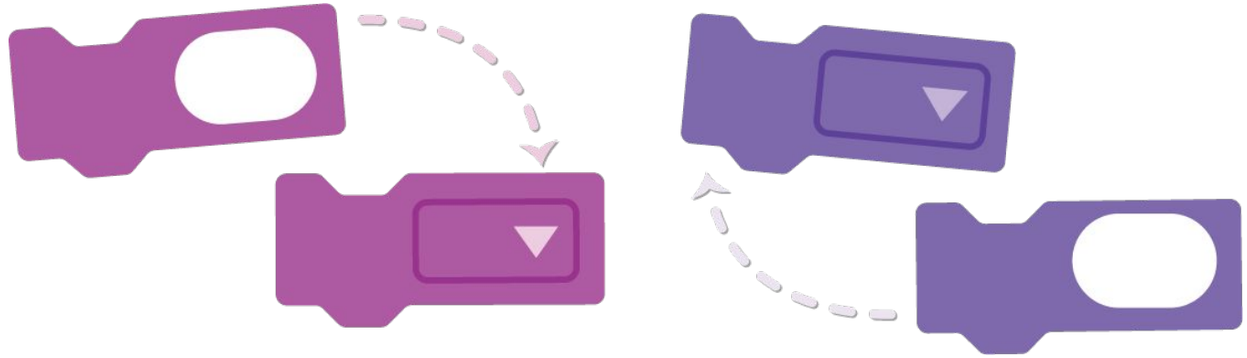
Try using blocks inside a loop or conditional statement, versus outside of a loop or conditional statement. It can make a big difference!

Considering Timing and Parallelism



Do you have multiple events trying to run at the same time? It may be unnoticeable to your eye, but it takes a small amount of time to run each code block. If two sequences are programmed to start at the same time (for instance, you are using several green flag blocks), you can get unpredictable behavior if you don't establish some timing and order through waits, broadcasts, or user interaction (like clicking or key presses).

Say one code sequence resets a variable, like the score, when the green flag is clicked. But a second sequence that also runs as soon as the green flag is clicked and is set to check that score and do something based on the value. Adding a small wait to the second sequence checking the score allows the program to reset the score before the second sequence starts checking the score, which will result in greater accuracy/less chance of unexpected failure.



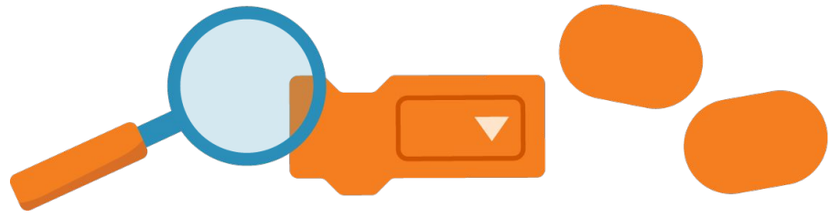
Is There a Similar but Different Block Option?

Some blocks look similar but can behave differently, or be more appropriate to use in different contexts.

“Set” versus “change” or “play until done” versus “start”...

Try using a similar block in place of what you have and see if there is a difference to the outcome.

Check the Values



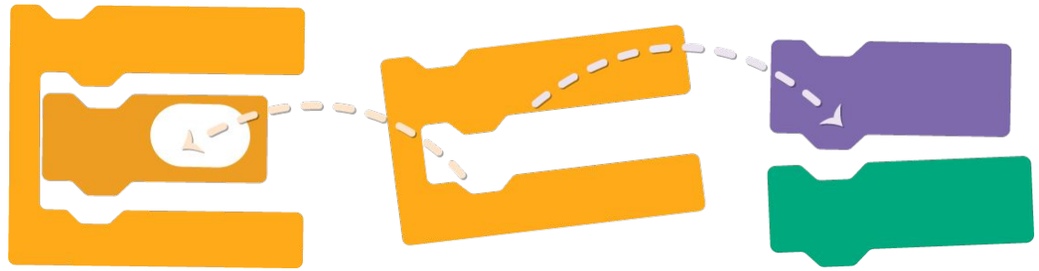
Are you using variables (either custom variables or provided reporter blocks, like “answer,” “x position,” etc.) in a code sequence? Do you know what the value is at the moment the code sequence is being run?

If you need to know the value to determine whether or not a sequence is functioning as expected, there are a few ways to find the value:

- If there is a checkbox next to the variable or reporter block in the block palette, ensure it is checked off to show it on the stage. You can uncheck to hide it again after debugging.
- Place the variable or reporter block inside a “say” block to have your sprite report the value. If you put the “say” block inside a “forever” loop, it should continually report the updated value as it changes.
- Click on the variable or reporter block in the block pallet or script area to get a one-time report of the value.

Also ask yourself: Do/should all the sprites control a variable, or should only one sprite have control? Where is the value reset? Where is it changed?

To Loop or Not to Loop



Does your program use Control blocks like "forever" and "repeat" to loop through steps?

- Check that all the blocks inside a loop should be there, or is there a block missing to reset the action or adjust the timing, like a "wait" block?
- Do you want your loop to run forever or just for a finite number of times?
- Or should something stop the looping?

Another possibility is perhaps you aren't using a loop when you should be. For instance, if you are using a conditional statement block like "if then”:

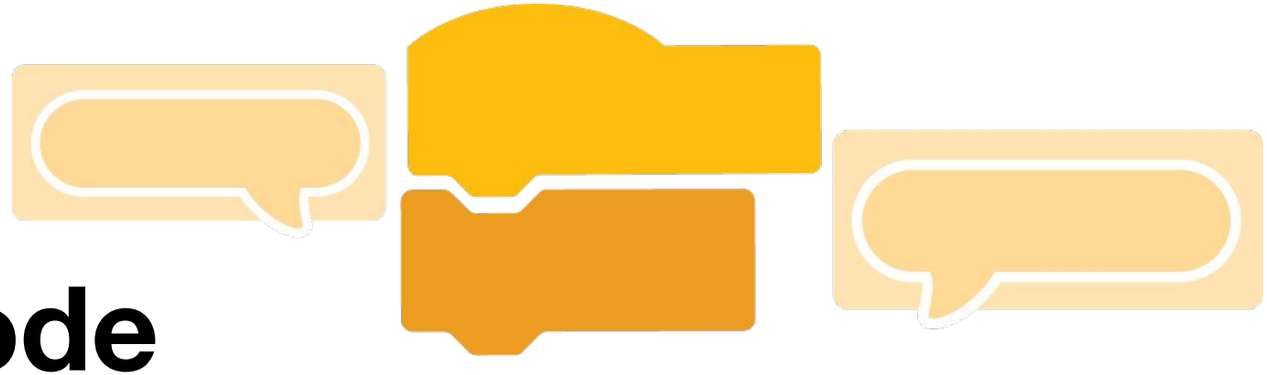
- Does the program only need to check if it is true or false once?
- Or does it need to check continuously, in which case, you would want to place your conditional statement inside a forever loop?

Is Your Code Sequence in the Right Place?

Is your sequence associated with the correct sprite, or the sprite versus the backdrop? Check which sprite is highlighted in the sprite area when looking at your code, or which image is shown in the upper-right corner of the script area. Or is the backdrop highlighted?

If you need to move your code to another sprite or backdrop because you accidentally wrote it in the wrong place, you can drag-and-drop a code sequence onto another by hovering over the correct one in the sprite or backdrop area and releasing when you see it wiggle. Or drag the code sequence to your backpack*, click on the correct sprite or backdrop, and then drag it up to that script area. Just don't forget to delete the code where it was written incorrectly once you've ensured it has been moved.

**Note: you logged in to access the backpack.*



Comment Your Code

Adding comments to your code not only helps others looking at your code understand how it is working, but explaining your code to an audience can strengthen your understanding of what you've created.

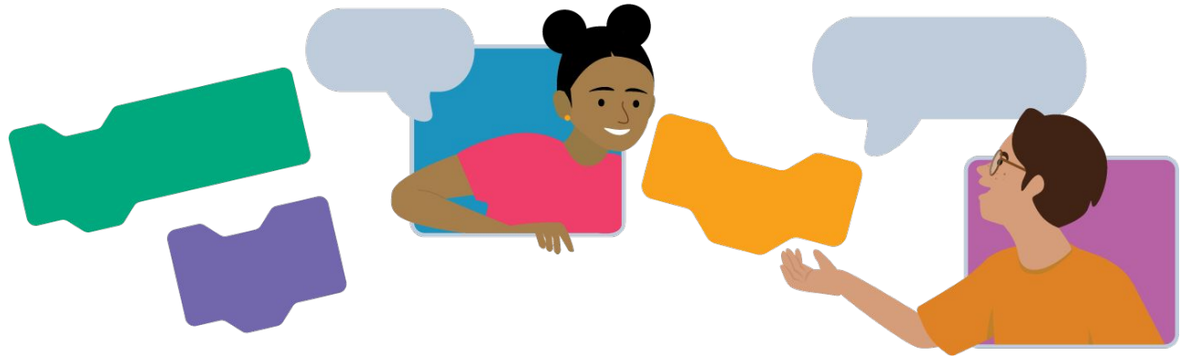
Writing comments can help you spot any errors, or help you identify any unnecessary extra code that can be removed. It can also help you remember how your code works when you come back to it later. Use everyday language to explain what a block, or small sequence of blocks, does.

Take A Break, Step Away

Sometimes, spending too much time focused on an issue can be counterproductive. Frustration can make it hard to think clearly. Taking a break and physically stepping away from the screen could help you clear your mind.

After some rest, focusing on something else, or getting some water, you can approach the debugging with fresh eyes. Start from the beginning, try one of the other strategies, and try thinking about the problem from a different perspective.

Ask for Help



If you've tried the other strategies and are still stuck, you can ask for help from a peer or an activity facilitator. If you enjoy participating in the Scratch online community, you could also share your project, ask for help debugging in a comment or project notes, and then other Scratchers can see inside your project to examine your code and help you debug.

Ask one to three people to try your code, as different people may have different perspectives or solutions. There is often more than one way to solve a problem, so you can determine which is the most efficient way for your program, which gets the closest to accomplishing your goal, and which solution you understand best and can reproduce.