Tips for Scratch

CSTeachingTips.org/Tips-for-Teaching-Scratch

1 Emphasize Scratch is REAL coding

to build students' confidence for future learning.

Scratch is similar to other programming languages.

2 Read code aloud

to help students debug code by acting it out.

Read the code aloud and pretend to be the cat!

3 Use implicit then explicit variables

to make creating new variables more intuitive.

Try setting or changing the volume variable.

4 Add sound blocks to code

to help students reason about sequencing.

Add different play note blocks to see how your code works.

5 Contrast set and change blocks

to help students distinguish easily confused blocks.

Set ignores the old value. Change modifies the old value.

6 Use "& wait" blocks

to use blocks that execute sequentially.

Remember to use the broadcast and wait block.

7 Let students write "bad" code

to let them apply abstraction to working code.

Now that your code works, could you use a repeat to make it simpler?

csteachingtips

1 Emphasize Scratch is REAL coding

Students often think that Scratch is a computer game and don't realize that Scratch is a tool to learn computer programming. We often want students to both learn these computer-programming skills and develop their confidence that they could learn more. To achieve this, it is important to emphasize that Scratch is a programming language and not a game!

2 Read code aloud

A common strategy in debugging for kids and adults is to read through and trace on paper what the code would do. To support this it is important to require students to have paper and a pencil out when they are working. When working with sprites in Scratch, you can have one student read the code and another one act out what the sprite would do either by moving around the classroom or drawing on paper.

3 Use implicit then explicit variables

Creating new variables in Scratch (i.e. explicit variables) can be a conceptual leap for students. To help ease this transition, help students see that they're frequently using implicit variables such as coordinates, direction, size, volume, instrument, tempo, pen size, and pen color. Help students see that they are using variables when using these implicit variables!

4 Add sound blocks to code

Students often use an if block when they mean to use a forever-if block. Help students recognize this by saying "if you put a play-note block in the if, how many times would you hear it?" This style of prompt can be a great hint for students and you can use play-note and say blocks to help visualize program execution. Some programming languages use "print-statements" as a similar strategy.

5 Contrast set and change blocks

Students often use a set block when they want a change block and vice versa. I use the phrase "Set ignores the variable's old value. Change modifies the variable's old value." When students make the mistake of using the wrong block, I'll ask "do you want to set the variable or change it?" and/or ask them "what's the difference between set and change?"

6 Use "& wait" blocks

Students can get confused when they use multiple play-sound blocks in a row because if you don't use the play-sound-until-done blocks the sounds start one after another before the previous sound can finish playing. Students can also get confused when broadcasting messages because there are broadcast blocks and broadcast-and-wait blocks. I recommend students use "until done" or "and wait" so that their blocks of code execute sequentially.

7 Let students write "bad" code

Even after students have learned repeat I find that they'll solve problems by copying and pasting code rather than using repeat. I find students are most efficient solving the problem if they get the code working without repeat before I suggest, "could you use repeat to make this code simpler?" My hypothesis is that as students are learning more abstract tools like repeat, it can be helpful to be able to see the working code without the more abstract tool.