Tips for Assessment

CSTeachingTips.org/Tips-for-Assessing-Programming

1 Predict the output of code

to demonstrate code tracing ability.

Draw a diagram as you trace through the lines of code!

2 Find and fix a bug in code

to demonstrate debugging skills.

Programming involves a lot of debugging! Let's practice!

3 Explain, compare, or critique code

to practice abstracting from lines of code.

Describe the code to someone who has never seen code.

4 Arrange code segments

to code without syntax errors.

Focus on your code's logic by rearranging these lines!

5 Solve the problem by hand

to demonstrate understanding of an algorithm.

Before coding, test your understanding of the algorithm.

6 Create a portfolio

to show off the breadth of their skills.

Show your friends and family all that you've learned!

7 Write or modify code

to demonstrate programming fluency.

Let's practice all of the skills at the same time!

csteachingtips

1 Predict the output of code

Ask students what code will output when provided specific inputs. You can also reverse this and ask students to provide inputs that will produce specific outputs. Ask students to compare the output of code in different programs to help students see the differences between similar concepts or commands. Help students understand that being able to predict the behavior of commands in a programming language is an important prerequisite to being able to write code. For all of these you can use them as formative assessment in class.

2 Find and fix a bug in code

Ask students to identify a bug in code. You can model good debugging practices by showing various inputs that produce correct and incorrect output. You can ask students to fix a bug that you demonstrate with tests or ask them to find a bug. For example, you can ask students to write a test case that will demonstrate the bug.

3 Explain, compare, or critique code

Ask students to write sentences to describe code in a way that a friend that isn't in the class would understand. This provides students the opportunity to demonstrate that they can abstract from individual lines of code. You can make the task a little easier by telling students to summarize the responsibility or behavior of particular parts of the code. As an easier to grade alternative, you can ask students to rename variables in the code.

4 Arrange code segments

Give students a set of lines of code and ask students to order them to produce a program with specific behavior. These problems are typically called "Parson's Problems" and allow students to reason about the logic of the code without having to worry about syntax. You can make this more difficult by including extraneous lines of code that students don't need to solve the problem.

5 Solve the problem by hand

Before students try to write code that solves a problem, make sure that they can solve the problem by hand. This can involve writing test cases that show that they can predict the expected behavior. You can also have students describe or draw the output of an algorithm.

6 Create a portfolio

It is difficult to write assessments that capture the breadth of the skills students have learned. Have students create a portfolio that shows off their work throughout the class. This can also be motivating for students to be able to see and share what they learned in the class! You can also give students a rubric for the skills they need to demonstrate within a project or their portfolio.

7 Write or modify code

It probably goes without saying that to assess students' ability to write code, you could ask them to write code. Additionally, consider providing students code that they need to modify to change the behavior of the code. This can provide students practice reading code and identifying the important features that determine each behavior.

Tips for Classroom Volunteers

CSTeachingTips.org/Tips-for-Classroom-Volunteers

☑1 Share a non-computing hobby

to show engineers have non-stereotypical interests.

For fun
I like to watch movies!

☑2 Focus on the positive

to avoid discouraging students.

My favorite part of my job is...

☑3 Re-introduce yourself to students

to minimize the imbalance in power.

Hi, I'm ... What's your name? ... Did I pronounce it correctly?

☑4 Move to students' eye-level

to connect with students as equals.

Can you talk me through what your code does?

☑5 Stay quiet during announcements

to model respect for the teacher.

Let's pause to hear what your teacher says!

☑6 Defer to the teacher

to be consistent with classroom norms.

How would you like me to respond if students are off-task?

☑7 Explain what counts as CS

to help students realize that they are learning CS.

I had to solve a similar problem last week!

csteachingtips

1 Share a non-computing hobby

Students might assume that all computer scientists want to program 24-hours a day and have only stereotypical hobbies. Even if you fit these stereotypes, the field doesn't require these things. Make sure students see that you have other interests to help them understand that regardless of their interests they can become computer scientists.

2 Focus on the positive

Sometimes the path to becoming a computer scientist is tough. For you it might have involved facing discrimination and a lot of hard work! Make sure to focus on the reasons that you stuck with CS rather than sharing the challenges you overcame. Any stories that make you seem super-human can make success in CS seem unattainable. For young students there is no need to describe challenges you've faced.

3 Re-introduce yourself to students

Ask the classroom teacher to introduce you and what you'll be doing in the class. When you work with students individually re-introduce yourself. This provides the opportunity to ask for the student's name and show interest in them as a person by making sure that you're pronouncing their name correctly. Don't hesitate to ask them to say their name again or ask how their name is spelled. But don't describe their name as unusual or difficult to say!

4 Move to students' eye-level

As an adult visitor to their class, students will likely find you intimidating. If you stand while you help students, you might unintentionally tower over them and reinforce that intimidation. Instead get at students' eye-level or lower. Make sure that you give students suggestions and never touch their keyboard or mouse! Try to remember that coding can be really difficult for beginners and avoid saying things like "this is easy."

5 Stay quiet during announcements

It is likely that the teacher will need to make announcements during the class. When this happens, do not quietly continue your conversation with students. It is natural to want to finish your conversation with the students, but stop talking so that you and all of the students can listen to the teacher. Make sure to stop talking even if not all of the students stop talking to listen.

6 Defer to the teacher

It is likely that students will need to be redirected if they get off-task. Don't assume kids who get off-task are bad kids or not interested in CS. Check with the teacher before class to ask how they would like you to interact with students who are off-task. They might suggest that you introduce yourself and ask "What are you working on?"

7 Explain what counts as CS

Students might not realize that what they are doing counts as programming. This might be particularly true if the programming environment looks like a game. Explain how what they're doing relates to professional programming so that the activity can build their confidence in their ability to do CS.

Tips for Encouraging Help Seeking

CSTeachingTips.org/Tips-for-Encouraging-Help-Seeking

1 Remind students about resources

to set the expectations that everyone will need help.

I hope you'll come to my office hours tomorrow at ...

2 Embrace your mistakes

to show that everyone makes mistakes.

Thanks for catching my mistake!

3 Hold help sessions in public places

to reduce power differentials and encourage attendance.

I hope you'll stop by office hours in the computer lab tomorrow.

4 Structure collaboration

to help all students develop a peer network.

You'll work with your assigned partner in class today.

5 Introduce the growth mindset

to help students reflect on their learning process.

You might be afraid to make mistakes, but mistakes are important!

6 Use Piazza for Q&A

to provide an anonymous Q&A forum.

You can always post your questions on Piazza!

7 Email students with low grades

to communicate you care and believe they can learn.

I noticed you didn't turn in the last homework assignment.

csteachingtips

1 Remind students about resources

Consider reminding students about opportunities for help at the beginning or end of every lecture. It takes only a few seconds and can increase attendance and make up for the fact that students often forget the timing and location of help.

2 Embrace your mistakes

When you make a mistake in class, acknowledge it and try to not appear embarrassed. These situations provide an opportunity to model that you can learn from mistakes and they are a normal part of computer science and the learning process.

3 Hold help sessions in public places

Many students report that going to a faculty member's office for help is intimidating. If you hold office hours in a public space like a computer lab, students can work on their homework and get help when needed. A lab can also provide more space so you can rotate around the room helping students. It can also be helpful to be able to leave at the end of your office hours without having to kick students out of your office.

4 Structure collaboration

Some of your students will start the class with a strong peer-support network. To ensure that all students have access to an informal support network, provide structured collaboration opportunities where students can get to know each other. For example, you can assign groups or require interaction between students in class. I also encourage students to exchange contact information to try to remove the social stigma of asking their peers for their contact information.

5 Introduce the growth mindset

Research by Carol Dweck and other social psychologists shows that students pursue more effective learning strategies when they see intellectual abilities as malleable (i.e., have a growth mindset) rather than innate (i.e., have a fixed mindset). Students with a growth mindset are more willing to learn from mistakes and challenge themselves. It can be helpful to tell students about this research to encourage them to adopt a growth mindset and to acknowledge that a fear of "looking stupid" is common.

6 Use Piazza for Q&A

The website Piazza.com offers a Q&A forum with minimal overhead. Students can post questions anonymously and I am often surprised how many students take advantage of this opportunity. Additionally, students can contribute answers to questions and instructors can easily see what questions haven't been answered.

7 Email students with low grades

Consider emailing students who received a low grade on an exam or homework assignment. Your email can encourage the student to make use of available resources. It can be helpful to express confidence in your student's ability to learn to motivate them and decrease the stigma of poor performance. You can use mail merge to save time without the email appearing impersonal.

Tips for Introducing CS

CSTeachingTips.org/Tips-for-Introducing-Computing

☑1 Describe programs as instructions

to connect programming with students' everyday life.

Programming is just bossing a computer around!

☑2 Point out products of CS

to help students see computing around them.

Traffic lights are controlled by computer programs!

☑3 Explain that bugs are expected

to encourage students to embrace mistakes.

Even for professional programmers, programs never work at first!

☑4 Introduce synonyms for CS

to demystify terms describing CS jobs.

There are lots of names to describe doing CS!

☑5 Promote collaboration & creativity

to dispel stereotypes about CS.

You'll work together today like computer scientists do!

☑6 Model programming

to show problem solving strategies.

That didn't work like I expected! What could I try next?

☑7 Publicize resources for learning CS

to help students see how they can continue learning CS.

If you want to learn more CS, you can find resources at ...

csteachingtips

1 Describe programs as instructions

Explain that programming languages are just languages that the computer understands and programs are just instructions for the computer to follow. This can help students see that programming relates to their experiences giving and receiving instructions.

2 Point out products of CS

Computer scientists are involved in the creation of most everyday objects. Students might know that computer scientists make apps, but might not realize that computer science is behind lots of everyday objects (e.g. traffic lights, microwaves, cash registers). You can have students list things that are important to them and discuss how those things rely on computers (and therefore computer science).

3 Explain that bugs are expected

School sometimes reinforces the idea that mistakes are bad. Instead, focus on how mistakes are part of the learning process and that this is particularly true in programming. Explain that when professional programmers write code, it rarely works the first time and that the most important thing is continuing to try things even if they don't work at first. Give examples of the errors that students might see to help them decode the error message. Explain that nothing they do will damage the computer.

4 Introduce synonyms for CS

Students have likely heard a lot of terms about CS: programmer, hacker, software engineer, coder, computer scientist, or developer. To demystify CS, explain that the differences between these terms aren't particularly important. Help students see that CS has a lot of specialties with different balances of programming, design, math, and problem solving.

5 Promote collaboration and creativity

Try to challenge stereotypes of CS as solitary and boring by providing opportunities for collaboration and creativity. Tell students that computer scientists always work in teams to solve big problems. Consider looking up the number of employees at a big tech company students have heard of to illustrate that computer scientists work together. Explain that computer scientists use creativity when they're inventing new products and when they're coming up with creative ways to solve problems.

6 Model programming

When students are new to programming, they don't know what to expect. Display your screen while programming to model the process of tinkering, reading error messages, making and fixing mistakes, and problem solving within a programming environment. Students likely won't be able to replicate all of the things you demonstrated, but it can be helpful for them to see what activities programming involves.

7 Publicize resources for learning CS

Students might not know how they can pursue more CS learning outside of class. Give examples of what they might want to learn to make and help them identify resources for learning CS online. There are lots of free resources available online!

Tips for
CS Lab Rules

CSTeachingTips.org/Tips-for-lab-rules

☑1 **Request eyes, ears, & monitors off**
to ensure students listen to announcements.

I have an announcement. Please turn off your monitors.

☑2 **Count down before breaks**
to avoid "just one more thing" negotiations.

We will have monitors off in 30 seconds.

☑3 **Explain food and drink restrictions**
to help students understand computers.

We have to be careful because water could ruin the computer.

☑4 **Set the default homepage**
to enable students to get started quickly.

Type in your login after you open your web browser!

☑5 **Schedule breaks for stretching**
to encourage sustainable ergonomics.

Give your eyes a rest and look around the room!

☑6 **Facilitate showing off work**
to motivate students and build community.

In 30 minutes we'll do a gallery walk to share our work!

☑7 **Encourage asking peers questions**
to encourage collaboration and minimize roadblocks.

Ask three before me!

csteachingtips

1 Request eyes and ears, and monitors off

It is common to ask students for "eyes and ears" before an announcement. Modify this to include "monitors off" so that students will focus on you during the announcement and not their computer. Don't start your announcement until everyone has turned off their monitor. If possible, make announcements at a location in the classroom where you can see all of the students' monitors.

2 Count down before breaks

Ideally students will be focused on problem solving when they are in the computer lab. Show respect for students and their problem solving by giving them a warning before you ask them to stop working. You can count down from 5 or 10 before stopping the class. When possible give students a 5-minute warning before the end of class and a 30-second warning before an announcement. These strategies can help reduce students' negotiations of needing to do "one more thing."

3 Explain food and drink restrictions

When students learn to program, they are often worried about breaking something. Explaining what can and can't break the computer can be empowering for students. Also, students are often allowed to drink water during class, so they might not understand why they can't have any food or drinks around the computers.

4 Set the default homepage

To enable students to get started quickly when they come to class, set the default homepage for their browser to point to a webpage they'll need to access during class. If you have multiple sites they'll use, consider having a shortcut for each one on the desktop.

5 Schedule breaks for stretching

Help students develop good habits for computer use by setting stretch breaks at regular intervals. You can teach students the rule 20-20-20, which is that every 20 minutes you should take a break of at least 20 seconds by looking at least 20 feet away.

6 Facilitate showing off work

Try having your students do a "gallery walk." You can have students display their work on their computer monitor and then go around the classroom to look at the work of their peers. If you want students to have the opportunity to explain their work to their peers, have students born in an odd month start by walking around and then swap. This practice can help build community in the classroom. It can also increase students' motivation to get to see the work of their peers.

7 Encourage asking peers questions

Many classrooms use the phrase "Three before me" to set the expectations that students will ask 3 other students before raising their hand to ask a teacher. This can be helpful for reducing the number of administrative questions that you have to answer and can facilitate students working together. Before helping a student who raises their hand, ask them which of their peers they had already asked.

Tips for Lecturing

CSTeachingTips.org/Tips-for-Lecturing

1 Integrate active learning

to increase students' learning and engagement.

Discuss the question with your partner.

2 Motivate lecture content

to help students understand the relevance.

This skill is needed to solve problems like …

3 Make learning goals explicit

to help students identify the important ideas.

By the end of lecture you should be able to …

4 Encourage questions

to have a chance to clarify unclear content.

What was unclear? What questions do you have?

5 Require students to self-assess

to help students identify what they understand.

See if you can apply this idea to the problem in your handout.

6 Ask students for feedback

to adapt to their needs & show that you care.

Thumbs up or down – did that make sense?

7 Explain your pedagogical moves

to help students understand your teaching strategies.

I'm providing this example to help motivate today's content.

csteachingtips

1 Integrate active learning

There is often pressure to cover a lot of content in lecture. However, it is unrealistic for students to sustain their attention throughout an entire lecture period. Students learn best when they can engage with the content rather than just listening! During the lecture engage your students in problem solving, discussing or debating content, practicing skills, or presenting or summarizing information. Even a few 2-minute breaks during class increases students' learning and retention! Not convinced? Read: tinyurl.com/activeLearningNYT

2 Motivate lecture content

We may be motivated by the lecture content independent of any practical applications. Help motivate students by explaining the applicability of lecture topics both inside and outside of the course. Assume that students will have diverse interests and try to provide variation when motivating topics. Try starting class with an example of CS from current events.

3 Make learning goals explicit

Novices often have difficulty seeing the forest for the trees. We can help our students see the central ideas by making our learning goals and expectations for them explicit. Documenting our learning goals can also be helpful for students when they are studying.

4 Encourage questions

By encouraging students to ask questions during lecture you can gain insights into parts of your explanation that were unclear and better understand what students find difficult about the topic. Ask students "What questions do you have?" rather than "Do you have any questions?" to set the expectation that questions are part of the learning process.

5 Require students to self-assess

During lecture provide opportunities for students to assess their understanding. You can ask students to re-explain a topic to a neighbor or solve a problem that requires them to apply a new idea or skill. Students might otherwise assume that they understand the content better than they do. This can help them identify what they don't understand and help develop their metacognitive skills. You can circulate and answer questions that come up.

6 Ask students for feedback

Solicit student feedback to communicate to students that you want to be as effective helping them learn as possible. Ask for students' feedback during lecture (e.g., asking for thumbs up/down if they understood an explanation) and in written form throughout the semester. Whenever you receive feedback make sure you summarize the feedback for students and explain what changes you will or won't be making based upon their feedback.

7 Explain your pedagogical moves

Some of your teaching practices (e.g., asking students to talk to a neighbor) might make them uncomfortable. Explain your pedagogical moves so that students understand your intentions and teaching strategies. For example, if you do not allow laptops in class, share the research finding that when students use a computer during class they learn and retain less, as do the students around them.

Tips for Pair Programming

CSTeachingTips.org/Tips-for-Pair-Programming

1 Explain pair programming goals

to motivate students to work together.

Pair programming can help you learn more efficiently.

2 Assign roles and computers

to avoid unnecessary pair negotiations.

The partner on the left will be the navigator first.

3 Pair students with similar skills

to avoid one student dominating the collaboration.

Find your assigned pair programming buddy!

4 Name common behaviors

to encourage productive pairing interactions.

Navigators: are you offering suggestions or commands?

5 Automate role-switching & timing

to facilitate role-switching compliance.

When the music plays – stand up and switch roles.

6 Only interact with pairs

to support students in working together.

Have you and your partner discussed your question?

7 Include buddy programming

to provide students autonomy and reduce frustration.

In 30 minutes we'll switch to buddy programming!

csteachingtips

1 Explain pair programming goals

Pair programming involves one student, the “driver,” using the keyboard and mouse while the other student, the “navigator,” provides directions and support. Pair programming is used in industry because it helps programmers learn from each other and write code with fewer bugs. It is also helpful for demonstrating that programming is a collaborative activity. Have students watch the NC State video: tinyurl.com/PairProgrammingVideo

2 Assign roles and computers

Students tend to prefer to be the driver. Assign which student will start in each role to avoid pairs beginning with a difficult negotiation. Throughout class, ensure that students’ chairs are positioned so that they can both see the computer screen. If applicable, specify which computer the students should use to avoid a negotiation about this.

3 Pair students with similar skills

Research suggests that students benefit most when they are paired with a student with similar skills. This isn’t always possible, but significant gaps in skills sometimes lead to the weaker student only sitting and watching or mindlessly following their partner’s commands.

4 Name common behaviors

Model positive and negative pair programming behavior by having a student pretend to pair program with you. After each of these role-plays, have students identify the positive and negative behaviors. Ask students “How do you think my partner felt when that happened?” to help students imagine the experience of their partner. It is helpful to model asking a partner for their opinion and checking if they understand. It can be helpful to refer back to a relevant role-play if students are stealing the mouse or bossing their partner around.

5 Automate role-switching & timing

Create a Scratch project to play music to indicate that students should switch roles. If students are physically able, have them stand up and switch seats when they switch roles. If a student won’t relinquish the driver role, their partner will be standing up as they wait for them, which allows you to intervene. If students work in a group of three, have all students rotate seats every time to make it easier for them to track the rotation of roles.

6 Only interact with pairs

When a student asks a question, make sure you address your answer to both of the students and take time to check that both students understand. If one student understands and the other student doesn’t, stay with the pair while one student explains it to the other one. This shows that explaining the idea is a learning opportunity and not to save you time.

7 Include buddy programming

At Harvey Mudd College we call solo-programming “Buddy Programming” because students are expected to continue to engage with their partner as they work. Sometimes pair programming can help establish this collaborative relationship. Tell students exactly how long they will be pair programming so you don’t have students ask “When do we get to work by ourselves.” They might not realize how this comment might make their partner feel.

Tips for HS student Recruitment

CSTeachingTips.org/Tips-for-Recruitment-in-HS

☑1 Educate counselors and teachers

to help dispel myths about CS and who does CS.

All students can take CS! It isn't just for "nerds"

☑2 Make it welcoming

to make sure all students feel they belong.

We are happy you decided to take CS! You belong here!

☑3 Have students promote your class

to have students hear from students that CS is cool.

Who wants to go talk about CS in the 9th grade class?

☑4 Show off student projects

to show students the variety of assignments.

Look at what students in my CS class made!

☑5 Recruit friend groups

to expand participation to students who might not enroll.

I think you'll like CS. You could take it with your friends!

☑6 Make CS required for all

to avoid students opting out without trying it.

CS is helpful to a range of jobs!

☑7 Use CSTeachingTips.org

to develop strategies for supporting students' success.

I'm here to help you be successful in this class!

csteachingtips

1 Educate counselors and teachers

Other adults at the school often have a lot of influence over students. Make sure that the counselors know that you want all students in your CS classes. Show them examples of your students' assignments to show them that CS can be engaging. Help counselors and other teachers recognize that there are stereotypes that discourage students from pursuing CS. Dispel these stereotypes so that other adults encourage all students to try CS. Use this "Critical Listening Guide" (www.ncwit.org/criticallistening) to identify and address counter-productive diversity narratives like "Women need to learn to be more confident" and "Women are such great collaborators."

2 Make it welcoming

Ensure that the climate in the classroom is welcoming of all students. This can include providing students opportunities to discuss ideas with their peers, making sure that students know how and where they can get help, and not equating experience with "smart."

3 Have students promote your class

Have students go to other classes to educate students about what they could learn in the CS class. Other students can be the most compelling spokespeople. Make sure that your students know to avoid reinforcing stereotypes about CS.

4 Show off student projects

Help students visualize what they would do and learn by showing off your current students' work. Look for opportunities to highlight their work across the school. For example, consider showing off work at back to school nights, science fairs, or even set up a table at sporting events.

5 Recruit friend groups

To avoid students being "the only" in a CS class or not having any friends in the class, recruit whole friend groups to take the class. Use this as a strategy to get students to enroll in your class who might otherwise opt out. Try to identify students who might influence their friends to try the class.

6 Make CS required for all

To avoid self-selection, have all students at the school try CS. This can help provide all students a background in CS so that they can see if it is something they like and might want to pursue. Students who don't fit the stereotype of a typical computer scientist often opt out of trying CS even though they might really like it! Try to focus your efforts at your school toward programs that serve all students.

7 Use CSTeachingTips.org

Seek out resources to improve your students' learning experience. We know this is a shameless plug, but check out resources from CSTeachingTips.org to learn about common misconceptions, strategies for engaging your students, and reminders of strategies to make your classroom inclusive.

Tips for
Reducing Bias

CSTeachingTips.org/Tips-for-Reducing-Bias

☑1 Make your expectations explicit
to avoid tacit assumptions within your class.

Start homework early so you have time to get help!

☑2 Grade anonymously
to mitigate the effect of unconscious bias.

Don't put your name in the homework!

☑3 Establish clear policies
to ensure students are held to the same standards.

Ill students receive a two-day extension.

☑4 Learn students' names
to enable you to engage all students equally.

That's a great question ____!

☑5 Acknowledge & manage your bias
to mitigate and monitor the impact of your biases.

Am I spending more time with particular students?

☑6 Teach students about bias
to educate your students and show you care.

Everyone has unconscious, unintentional biases!

☑7 Listen to students' experiences
to learn how you can create a supportive enviroment.

What in your classes makes you feel like you belong?

csteachingtips

1 Make your expectations explicit

We often have tacit assumptions about what students should do to be successful in our classes: take notes during lecture, ask questions, get started on the homework early, come to office hours, and read the book. These things might be different in each of their classes and some students might be better prepared to guess what the right things are. Help remove the guesswork for students by making these expectations of their behavior explicit.

2 Grade anonymously

Research shows that people evaluate the same resume more favorably if it has a man's name than a woman's name. We are surrounded by stereotypes, and research studies show that these unconscious biases can shape our evaluations of others. If possible, remove information about the identity of the student when evaluating their work. I have students write their names on the back page of the exam. I could look at their name, but I don't.

3 Establish clear policies

When students ask for exceptions for extenuating circumstances we have to make subjective decisions regarding their situation. Write down your plans for these situations to ensure all students are held to the same policies. When new circumstances arise add them to your plans because having them written down will help you improve your consistency.

4 Learn students' names

Knowing a student's name allows you to better connect with them. However, it is normal to learn some students' names more quickly than others. Make an effort to learn students' names to avoid unequal connections to your students. It can be helpful to video students saying their names so that you can practice the correct pronunciation. Checking that you are pronouncing their name correctly is an important way to show you value them as a person.

5 Acknowledge and manage your biases

We often conflate implicit bias with explicit sexism, racism, classism, ableism, homophobia, or transphobia. While we wouldn't identify ourselves as sexist, we have to recognize that some of our actions may be unconsciously based upon our biased assumptions about the abilities, interests, or needs of girls or women. It is important to become reflective about your behavior to identify ways in which your expectations of, or interactions with, people might conform to stereotypes you unconsciously hold about one of their identities. Consider adopting a system where you randomize the order in which you call on students.

6 Teach students about bias

Help educate students about implicit bias. For example, when you introduce your plans for anonymous grading you can explain the underlying research about implicit bias. Introducing implicit bias research also communicates to students your goal to treat all students fairly.

7 Listen to students' experiences

Learn from students' stories about their interactions with their instructors and peers. When you realize you have contributed to a student's negative experience: apologize, identify what you'll do differently next time, and don't try to justify your behavior.

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

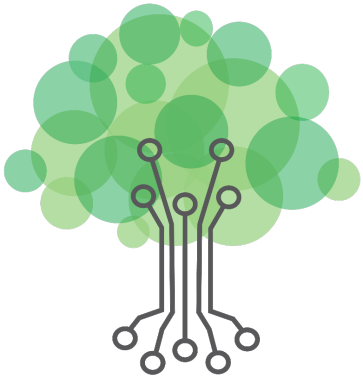
Even after students have learned repeat I find that they'll solve problems by copying and pasting code rather than using repeat. I find students are most efficient solving the problem if they get the code working without repeat before I suggest, "could you use repeat to make this code simpler?" My hypothesis 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.

csteachingtips

For Tutors

1. Introduce yourself.
2. Ask the student to describe their homework problem.
3. Ask the student to describe what they want help with.

- **If they don't know how to get started**, ask them to describe the problem in detail:
  - What are the goals of the problem?
  - What are the inputs?
  - What are the outputs?
  - What is their relationship?
  - Can we solve a small example by hand?
  - Is there a part of the problem that they could write code for?
    - (and worry about the rest later?)
  - Can you describe the algorithm in words?



- **If they have a syntax error,** ask them:
  - What line is the syntax error is on?
  - What does the text of the error mean?
  - What does the internet suggest about how to fix this error?
  - What have they tried to fix this error?

- **If their code doesn't work,** ask them:
  - What evidence do we have that the code doesn't work?
  - What test case doesn't work and what incorrect behavior or output results?
  - Could we come up with a simpler example that demonstrates the error?
  - What lines of code might be producing the bug?
  - Why hypotheses do we have for what might be causing the problem?
  - How can we test these hypotheses?
    - (e.g. writing new test cases, adding print statements, using a debugger)
  - Could we walk through an example that doesn't work: by hand? with a debugger?