The Case for Computer Science: Equipping Students and Teachers with Future-Ready Skills

Q&A with **Jennifer S. Kay, Ph.D.**, Computer Science and Educational Robotics Expert, LEGO® Education Trainer, and Professor of Computer Science, Rowan University

"Once a student understands the fundamental building blocks of computer science, that's a skill they can apply across any subject."

What is computer science (CS) and why is understanding it and computational thinking (CT) important for future readiness?

Computer science focuses on what's going on inside computers, understanding how computers work, as well as how to efficiently design and code the apps that run on computers, including smartphones.

As far as future readiness is concerned, we know that not everyone will be a computer scientist, but everybody is going to be interacting with computers. We need to develop students' ability to speak to computer scientists in a common language and employ computational thinking skills that allow them to break down problems so computer scientists can efficiently solve them. One of the great things about hands-on learning is it provides a great reminder that you don't need a computer to learn computational thinking.

What are some skills and concepts students learn through CS and CT?

A lot of people hear "computer science" and think "Oh, that's just coding," but there's so much more to it. Things like critical thinking that allows students to carefully define the problems they want to solve, break it into smaller subproblems, figure out what information each subproblem needs, as well as how it may interact with other subproblems. Then we get right into iterative thinking and how to test your code; these are all part of computer science and computational thinking.

Also, because a lot of CS and CT is done in teams, students naturally develop those 21st century skills like cooperation, collaboration, and communication at the same time.

Why is computational thinking an important skill? How does it support future learning and career development?

Computational thinking skills are as foundational as math and language arts. Each area teaches students specific content while simultaneously teaching critical skills such as problem solving, teamwork, critical thinking and communication.

The computational thinking approach to breaking a problem into smaller pieces comes from the perspective of making it possible for a computer to do the work. Computational thinking also teaches students how to make connections by taking a specific solution and generalizing it to work on new problems. This approach helps with attacking other projects, whether that's planning a road, solving a math problem, writing a paper, or following a recipe, these are all tasks that require thinking from the top down. What's the big thing I want to convey? How do I split it into smaller, key concepts? What tools have I already developed, and which can I reuse?

Suppose you're managing multiple building projects. What are the stages? Do you need to plan each one from scratch or can you develop general procedures? In other words, what has been solved already?

Likewise, if your recipe calls for sautéing carrots, and you already know how to sauté onions, you can expand your "how to sauté onions" algorithm and now your carrot problem has been solved!

How easy is it for teachers to illustrate real-world application of these skills in a way that is relevant for students' daily lives?

It's about the lens you teach through and making concepts relevant to the students' daily lives. So, for this generation of students, that could be anything from how apps are developed for your phones to big picture issues like sustainability and the environment. What kinds of solutions can we come up with to eliminate or clean up waste? Passions can be another entry point; think about music, we can explore what music looks like on a computer by introducing students to the relationship between the frequency of a sound and its pitch.

What does hands-on computer science look like? How can it play a role in getting students excited about CS and STEAM learning?

To me, hands-on computer science is simply about showing students how they can do interesting, meaningful things using computers.

One of the firs t things students learn about coding is "if/else" situations. If this, do one thing. Otherwise, do something else. That concept is fundamental to computer science. When I started teaching this concept, we'd write a program about who can vote. If you're over 18, you can vote. Otherwise, you'll be able to vote in five years, or whenever. But that isn't exactly a topic that gets students excited.

However, you can teach the if/else concept by having students take a goofy selfie in front of a green screen, then find another picture to merge with it. Maybe someone chooses a shark with its jaws open. Maybe it's a photo from a soccer game. To merge the two pictures, the students write code that checks every pixel in their goofy image. If the pixel is green, they replace it with the corresponding picture's pixel; otherwise, they leave their photo alone. In the end, a student might wind up being eaten by a shark or scoring the winning goal in the World Cup. Students are learning the same concepts, but they're so much more motivated when it's a fun, interesting activity.

You can teach that same if/else concept using engineering design. Let's say I want my model to follow me in a straight line, always 25 inches behind. My model has a sensor to tell how far it's trailing something. So, I write that if there's nothing in front, it should drive slowly. But if there's something less than 25 inches ahead, it should stop.

So if I'm walking along, and the model is driving and measuring, driving and measuring, but then stops because it notices something 25 inches in front, that's because I've stopped. While stopped, it keeps measuring and starts moving when it no longer detects something 25 inches in front. That's because I've resumed walking. It looks brilliant, but it's just following the steps or the directives it has been given.

That is another activity exploring the same if/else concept, but again, much more interesting than seeing if you're old enough to vote. That's what I call hands-on computing—doing something you can see, something concrete and meaningful to you.

How do skills developed in CS contribute to students becoming well-rounded learners and problem-solvers?

Once a student understands the fundamental building blocks of computer science, it helps them to be able to break up a problem to make it more manageable. And that's a skill they can apply across any subject.

Consider writing code; for each piece, you need to figure out how to tell the computer exactly how to do it step by step, using language that the computer understands. It's all about problem solving and writing clear instructions. I like to think of a computer as a child who takes everything you say very literally. You ask a child to wait a minute, and they start counting "one-Mississippi, two-Mississippi..." all the way up to 60. It's not what you meant, but it is what you said.

For any K12 educators looking to incorporate CS into their curriculum, how would you recommend getting started?

They should join CSTA, the Computer Science Teachers Association, which has free basic membership and excellent resources. I strongly recommend using curricula that align with CSTA standards.

I often call educational robotics my Trojan Horse approach to teaching computer science. When you walk into a classroom with a robot, you instantly have the attention of students. I recommend looking for solutions that offer projects and curricula that align with the CSTA standards, and that are robust enough for classroom use.

No matter what approach you use, there hasn't been a better time to jump into the world of computer science and computational thinking!

To learn more, go to **www.legoeducation.com**

LEGO education

Produced by *District Administration*