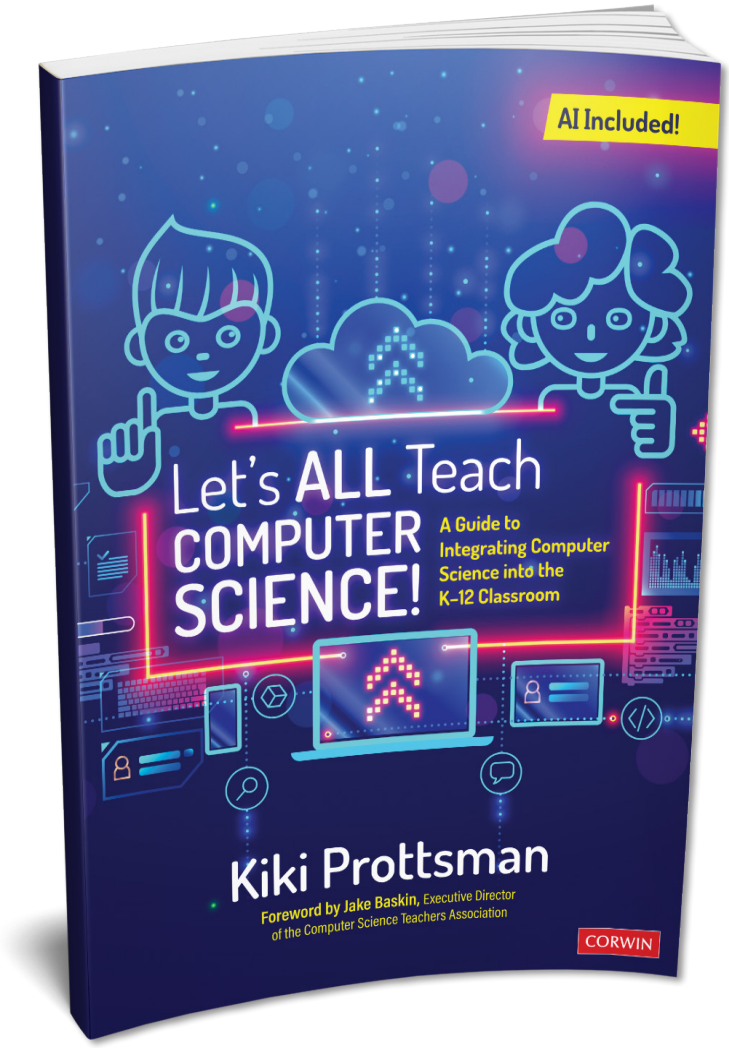


Thank you

FOR YOUR INTEREST IN CORWIN

Please enjoy this complimentary excerpt from Let's All Teach Computer Science!.

LEARN MORE about this title!

CORWIN

Step #3: Truly Integrate Lessons

"Absorb what is useful, reject what is useless, add what is specifically your own."
— Bruce Lee

This is the gold standard for 21st century education—replacing the lessons that you used to offer with enhanced versions that are rich in computer science exposure. It may be a few years (or half a decade) until the large curriculum providers have high-quality solutions, so let's explore ways that we can do this ourselves.

Fortunately, there are several CS standards that naturally coordinate well with math and science, especially when it comes to collecting and representing data. Look for moments in your lessons where you can showcase different ways to present the same information. You can take this one step further by helping students to analyze their data to support a claim. Both of these are common occurrences in subjects like math and science. Collecting and exchanging data via digital means is another great integration point. Find moments in your lessons where students can collect information, process that information using software, then present the information in multiple forms. Those simple lesson modifications hit multiple standards for Grades K–10 and require very little extra in the form of time or equipment for activities that already focus on information gathering or processing.

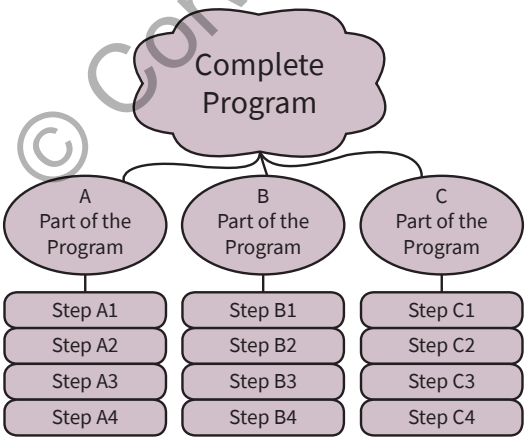
Beyond data collection and modeling, coding is a great way to help students digest the ideas they are learning. The beauty of writing code is that you need to understand the way things work to represent them computationally. This is part of the reason why computational thinking exploded onto the scene in the early 2010s as CS gained popularity.

Computational thinking is generally said to be made up of four pillars:

- Decomposition
- Pattern matching
- Abstraction
- Algorithms

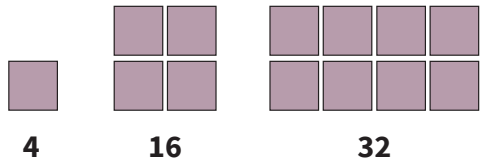
Decomposition

Decomposition is the ability to break a big problem down into smaller pieces to make the issue more manageable. It's like pulling a story problem apart to know which steps need to happen to solve it.

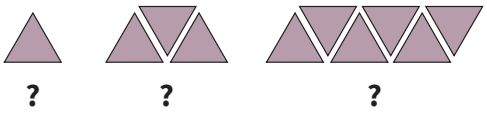


Pattern Matching

Pattern matching is the ability to look at a problem and see how it might be similar to something else that has been tackled before (or how it is similar to something else that needs to be tackled). It's like recognizing that a story problem on a test is just like a story problem assigned in homework, but with oranges instead of apples.



Can you find the pattern?

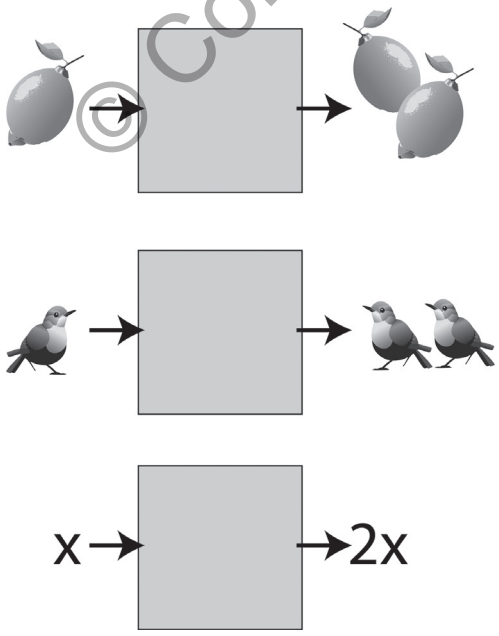


Abstraction

Abstraction is the ability to focus on what is important and abstract away little details. It's like realizing that it doesn't matter whether the girl in a story problem is named Tish, Latoya, or Betty … or whether she's buying apples, oranges, or beach balls; you can create a formula that will solve the problem for any of those cases.

Abstraction also refers to the idea that you don't always need to know every detail to count on a result. For example, in Figure 5.1, you don't need to know *how* the box does what it does. All you need to know is that one thing goes in, and two of them come out.

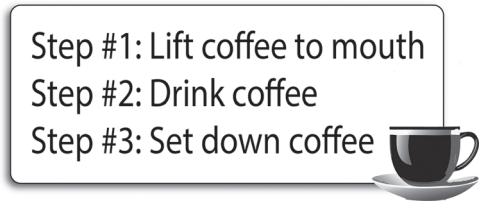
Figure 5.1 Magic Doubling Box



Source: Kiki Prottsman, 2023. Elements designed using Adobe Illustrator with AI generation.

Algorithms

Algorithms are tools to reconstruct and relate the recipe for solving your problem in a consistent and meaningful way. It's like coming out the other side with a mathematical formula that you can use to solve the story problem (and maybe even use it to solve other problems in the future!).



Source: Kiki Prottsman, 2024. Elements designed using Adobe Illustrator with AI generation.

Putting the Pillars Together

When students know how to employ these pillars of computational thinking, they have what they need to break down and digest the ideas at the heart of any subject—whether that be the water cycle, the structure of long division, states of matter, or linear equations. Students will need to decompose the concept into base components, figure out how each piece relates to the application that they want to create, abstract away the parts of the topic that aren't critical to the assignment, and construct a project that meets requirements.

Coincidentally, that's also what teachers need to do to construct their own integrated lessons for these subjects. Imagine that you already have a lesson on two-step equations. Perhaps it consists of three days of material: some lecture time, some worksheets, some partner work—plus a few nights of homework. We can break those lessons down into their most important parts, find patterns between your favorite activities and popular activities that integrate CS, abstract away elements that aren't crucial for learning two-step equations, and weave together a functional and entertaining integrated lesson.

Decomposing Lessons

Let's give this example a try with one of your actual lesson activities. Grab a unit that you would normally teach and pick out the main objectives. Now, pick a computer science standard that you would like to hit.

For my example, I'll continue with two-step equations and my objectives will be:

- Use inverse operations to solve two-step equations
- Describe how operations must occur on both sides of the equals sign for the statement to remain true
- Verify potential solutions by substituting the answer for the variable

My computer science standard will be:

- CSTA 2-AP-16: Incorporate existing code, media, and libraries into original programs, and give attribution

Notice that this standard does not dictate that computer science be the *focus* of the lesson; it only suggests that students need to understand that attribution needs to be paid when using other people's assets. This is perfect for my purposes, because I can prepare a series of puzzle pieces for students to choose from when trying to create an equation solver, and that will cut down our computer time significantly.

Perfect. Now I have my lesson decomposed into the pieces that I need students to learn, and I'm using backward design to boot!

Pattern Matching Lessons

This is a great time to go through lessons for other integrated topics and see how the creators weaved CS into another subject. Did they have students fix broken programs that became utilities for their assignment? Did they encourage students to work together to brainstorm ways to depict story problems using choreographed sprites? Did they suggest that students use their own encoding of secret messages to transport data from point A to point B? What can you find that you can tailor to the lesson that you're building right now?

As I noted earlier, I'm going to follow the model of providing disjointed chunks of code that students need to rearrange via logic and reasoning to create a two-step equation solver. With any luck, the process of figuring out how to arrange prewritten code will

help internalize why we use inverse operations to simplify an equation, as well as the usefulness of keeping an equation balanced by performing operations on both sides of the equal sign. Since this was predicated on the ideas in CSTA Standard 2-AP-16, I am confident that objective will be met, as well.

Abstraction Across Lessons

In our case, this step aligns tightly with pattern matching. To clearly see the patterns between the lessons that others have done and what we want to do, we'll need to ignore the fact that the inspiration lesson may have been about the lifecycle of a butterfly, and instead extract reusable core takeaways.

For my purposes, I'll be ignoring the exact subject of the inspiration lesson, as well as the actual content of the previous code and the language in which the code was written. All those things can be replaced without losing the standards and objectives that I'm trying to hit. And I'm not worried about the fact that I still have one objective left, because this section is about integrating CS lessons, not about completely displacing traditional teaching. I can still use worksheets and videos to pick up additional ideas.

Algorithmifying Your Lesson

Ignoring the fact that it's not a real word, you can finish this process by algorithmifying your lesson. This is just to say that you'll write it up into a lesson plan that you can follow when you're ready to run this with your classroom. If it's successful, you can probably even sell it on Teachers Pay Teachers for $2–$12 a pop and be a hero to educators just like yourself! If it's not as successful as you would like, make notes on your plan and tweak it for next year.