Teaching Computational Thinking

An Integrative Approach for Middle and High School Learning

By: Maureen D. Neumann, Lisa Dion

Citation:

Teaching Computational Thinking: An Integrative Approach for Middle and High School Learning
By: Maureen D. Neumann, Lisa Dion
DOI: 10.7551/mitpress/11209.001.0001
ISBN (electronic): 9780262366144
Publisher: The MIT Press
Published: 2021

OA Funding Provided By:

The open access edition of this book was made possible by generous funding from Arcadia—a charitable fund of Lisbet Rausing and Peter Baldwin.

7 Incorporating Computational Thinking into the Classroom

In this chapter, we discuss how you can do this on your own with the content that you teach in order to integrate and embed computational thinking into your classroom content.

We know teachers worry, "How do I do all this on top of everything else I need to do?!" It is true that you have a lot to teach and students have a lot to learn. We advocate that you take small steps in the beginning, starting off small and taking on more as you are able. Change is a process that takes time and persistence. As you progress through a process to include computational thinking and/or programming in your instructional practices, you will need support in that process by gathering all the resources you can, collaborating with different people, persevering through roadblocks and obstacles, and learning from your mistakes.

In this chapter, we hear from teachers who used computational thinking in their classrooms. The first two stories come from teachers who incorporated computational thinking as part of a culminating unit project. The third story comes from a technology teacher whose goal was to teach loops and how to use random number generators, and who incorporated ideas from sections 2.1 (Kandinsky art) and 2.3 (Albers art) of chapter 2 to do it. In the fourth story, the teacher used embodied learning experiences to help her students overcome a misconception when drawing squares and equilateral triangles. The last episode is from an art teacher whose students designed and built an art installation over three months that included several aspects of computational thinking.

The practice of teaching is a complex enterprise. We understand the varied context every individual teacher is in, so we wrote the book with the belief that teachers are professionals who know their students' needs best. We believe that teachers are smart, industrious, and continuous learners who can take the information in this book and adapt it to fit their situations. We intended to give teachers flexibility and choice in how they present the material to their students rather than being too prescriptive regarding timing and organization. Different teachers we worked with confirmed this belief. One teacher commented, "I like how the book took it. It wasn't a lesson; it was outlining how to do this one problem and all the different parts of it. [It] made it easy to figure out a cool lesson plan. If a textbook has a lesson and I don't like parts of it, I discard it easily. I like that I could look at this and choose different pieces that I wanted to use." Another teacher commented that she could take the topology graph from chapter 3 and "use it for a unit on food chains . . . to examine what happens when a rabbit population tanks . . . to look at the different influencing factors. I won't use it for Harry Potter, it wouldn't make sense for my kids." For us, these comments verified that teachers are professionals who choose activities that work for them in their context.

7.1 Start Small: Have a Final Project That Contains Computational Thinking

Wiggins and McTighe (2005) argue that we should plan our lessons or units with a specific desired learning result in mind and then design the learning tasks such that students are able to demonstrate what they know and are able to do as a result of that learning at the end. Students can demonstrate what they know and are able to do through a performance task rather than a paper-and-pencil test. Wiggins and McTighe defined a performance task as an authentic task that uses a student's knowledge to effectively act or bring to fruition a complex product that "assesses the student's ability to efficiently and effectively use a repertoire of knowledge and skill to negotiate a complex and multistage task" (Wiggins and McTighe 2005, 154). These performance tasks incorporate computational thinking in that they require the breakdown of a complex and multistage task into its core components given the students' knowledge and understanding from the unit's big ideas, the transfer of learning to a new situation, and the creative displaying of that knowledge.

As the teacher, you decide how students demonstrate their knowledge to you, to their peers, and to the world. You could empower students to choose the means by which they demonstrate their mastery of the learning goals (e.g., do they want to present a slideshow or PowerPoint presentation, write a paper, create a video, write a song, or make a computer program).

For example, in a sixth-grade classroom, students had to discuss a significant event in a story and how it changed the trajectory of the characters' story arc from a book they were reading, drawing on specific details in the text. They were given the option, among many others, to create an animated program with the Scratch programming environment. A couple of boys took up their teacher's offer to use Scratch. They used the `think` and `say` commands to highlight important dialogue passages from the book. They created a different `backdrop` that reflected where some of the important scenes in the book took place. Students used the `move`, `glide`, and `turn` commands to highlight some key actions in the book. Creating an animated presentation of their knowledge about a book enabled the students to demonstrate their knowledge of the CCSS-ELA standards being assessed and incorporated computational thinking. The teacher created a rubric that outlined the key requirements for the assessment to ensure that the content was not lost in the creativity that the Scratch animation provided.

In another example, a sixth-grade social studies teacher gave the option of using *Minecraft* to create a digital diorama of Plimoth Plantation. The students had to satisfy the same NCSS standards as those who created a three-dimensional diorama. Just as the students had to gather information, plan their environment, solve problems, and persevere in making their three-dimensional models, students in the *Minecraft* environment had to apply those same computational thinking skills in the technology environment.

7.2 Scale Up: Create Lessons or Units of Learning That Include Computational Thinking Based on What Your Students Need to Know and Do

To create a lesson that includes rich tasks that integrate computational thinking with your content area, look at a standard you need to teach. What do the students need to know and do? What are the essential questions for your lesson or unit? After you have defined them, begin to think about the essential components of computational thinking that support learning the content standards or essential questions of your unit.

Are there ways that students can learn this content standard by including problem solving, collaboration, abstraction, and/or algorithmic processes? Then think about ways you can differentiate the learning with rich tasks that allow all students multiple entry points into the learning activity so that it is accessible to a wide range of learning abilities (low threshold); provides opportunities for extended learning and challenges advanced learners (high ceiling); incorporates a variety of approaches and allows diverse representations (wide walls); engenders interest in and engagement with the topic and is grounded in real-life experiences; encourages collaboration and discussion; sparks students' curiosity and promotes decision-making; and fosters creativity and individuality. As you think about including computational thinking in your lessons, always keep in mind the desired results and learning goals for your students.

One middle school technology teacher created a lesson based on sections 2.1 and 2.3 in chapter 2. She started the lesson by displaying a Kandinsky art image on the screen (similar to figure 2.2) with an opening prompt on the board that read, "What do you notice about this picture? Describe it. Describe how you could code it." She passed out paper and pencils and had students write for one minute. Using a think-pair-share pedagogical strategy, she had students turn and talk with a neighbor about their thinking. Afterward, she asked the class what they discussed and wrote down some of their ideas:

- "The circles change color when they overlap."
- "It looks like eyeballs floating in the ocean."
- "Some circles have a border."
- "You need a random circle generator."

She then prompted them for their coding ideas. They talked about the need for randomness for the colors and, after some prompting, mentioned loops to repeatedly draw the circles. Then the teacher had the students log into the computers and Google image search the artists Kandinsky and Albers, telling the students they could create a computer program in Scratch or Codesters that mimicked either artist's style or a combination of both. After about 40 minutes and walking around and working with different students, she displayed different students' projects on the screen.

- One student (working in Codesters) had static squares, circles, and triangles that were red, blue, yellow, orange, purple, and green (see figure 7.1).
- One student (working in Scratch) had squares of all different bright colors that were randomly moved and placed so the art looked like a whirlwind.
- One student (working in Codesters) took a picture of a Kandinsky picture and set it as the background image, then had different shapes appear on top of it.
- One student (in Scratch) created different squares and moved them so that they were concentric and in rainbow order.

After the lesson, the teacher reflected on her experience: "It was cool with how much range there was with what they could do. . . . I was worried that covering loops and hexadecimal and random at the same time would be too much, but it wasn't. They caught up with random quickly. There was a lot of stuff they could incorporate, [if they were done early] it was easy to think of other things they could do." She did learn that this was "easier on Codesters than Scratch, because in Scratch you have to have a lot of sprites . . . but it was good." The comment that made the teacher smile most was from a student

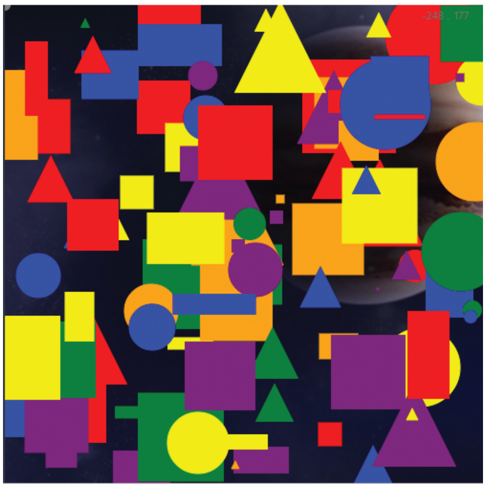

Figure 7.1
An example of the output from a student's simulation of Kandinsky's art style. Image: Courtesy of Codesters.

named Duncan, who told her, "I can't do art, but on the computer, I guess I'm not so bad." The teacher added that "it was cool for him to see that he can be an artist too. You say that the program is art, and now they know they can do art."

7.3 Learn All That You Can

Computational thinking is a way of thinking that is not new to a lot of people. Many people recognize the different components in the language of their content areas. However, integrating coding into your teaching is the newest piece of learning for many teachers. It takes time to learn new things. Cadieux Boulden et al. (2018) found that teachers' lack of time for learning how to code was the primary barrier to integrating computational thinking and coding into their teaching. Many teachers did not feel comfortable including coding in their lessons because they believed they lacked a background in computer programming. Structured time to learn new things may not be available at your school or in your school district, so you may need to start learning it on your own. There are a number of routes for overcoming this obstacle:

- Read some books about programming and its connection to teaching in the twenty-first century.
- Try out the different Hour of Code modules.
- Explore the Scratch, Codesters, or Python tutorials.
- Watch video tutorials on the internet about these coding languages. It does not have to be these languages specifically; there are a variety of programming languages out there you could learn.
- Have a tech-savvy friend or family member teach you about coding.

There are lots of resources available to learn how to code, and soon enough you can be a resource for other users.

For example, a sixth-grade mathematics teacher, after completing the Hour of Code Learning Course E, which included a lesson called "Drawing with Loops" (https://studio.code.org/s/coursee-2019), thought the lesson would be a good review for her students and would provide an opportunity for them to apply their understanding of different geometrical properties. As she watched her students in the computer lab one December day, she noticed them struggling to draw the different shapes with the `move forward 100 pixels` and `turn left` and `turn right` commands. In particular, she noticed them struggle with drawing the rocket's square window (https://studio.code.org/s/coursee-2019/stage/2/puzzle/5) and the equilateral triangle at the top of a rocket (https://studio.code.org/s/coursee-2019/stage/2/puzzle/6). She knew her students understood the properties of a square and an equilateral triangle, yet they struggled to draw the rocket's window or top with the `turn left` or `turn right` code commands. They coded the sprite to turn the incorrect number of degrees or to turn in the wrong direction.

The teacher did, however, love watching her students collaborate in trying to figure out how to draw the rocket top correctly. When a student found the solution to a problem, the solution ran through the classroom like wildfire. The teacher did question whether her students understood what they were copying from each other. Their time in the computer lab ended with only a couple of students having completed the "Drawing with Loops" lesson. The teacher decided that when they got back to the classroom, they were going to talk about the actions the sprite was taking on the screen based on the code the students wrote. Before the students shut down their computers, she asked them to screen capture their code and send her the pictures.

When her students came back to her the following day, she first had them share their understanding of the properties of a square and equilateral triangle, which she recorded on flip chart paper. Then she handed out a couple of examples from their screenshots from the day before that exemplified their struggles with why the code was not working as they wanted. In groups of three, she asked them to act out the action in the code—to be the sprite drawing the roof. From that experience, students realized that the direction the sprite was facing or where the sprite was located on the screen influenced how the next piece of code should be placed. The next time they went into the computer lab to work on their Hour of Code lesson, they had a deeper understanding of what they were doing when drawing the square and the triangle, and they applied that learning to draw the other shapes in the lesson.

7.4 Collaborate with Colleagues across Disciplines, Schools, and/or States

Sometimes you may be the only person in your school or district who is interested in integrating computational thinking and coding into your teaching, but you are not alone. There are other people who are interested in doing the same thing you are and are interested in working together. Send that cold email to find someone in your school, district, or community who is interested in transforming their practice to include more computational thinking in their teaching. Team up. Send out another email, asking to create a user group for teachers.

If you decide to include computer programming in your teaching, know that you do not have to be an expert coder to do this. There is an amazing online community that helps each other out. The Scratch, Codesters, and Python programming environments have an online help and share community that talks about problems with their programs. You can access that community for help and in the process teach your kids the skill of how to access help online safely.

For example, a middle school art teacher created with her seventh- and eighth-grade students an interactive, hands-on garden installation that incorporated major aspects of the Claude Monet garden in Giverny, France. She began her journey of incorporating computational thinking and coding into her teaching with one small flower that incorporated LED lights and sound. She made the flower as part of a course on creative maker spaces and using Arduinos. After she successfully crafted one small flower, she began to think about building a garden of flowers with lights and sound that worked independently of each other but wasn't sure where to begin.

Through her discussions and collaborations with a local technology education consultant and a technology specialist, the teacher learned of a garden created by MIT students that was used to teach coding. That information, in addition to the original flower prototype and her passion to integrate art with technological innovation, was enough to propel her. She proposed to her seventh-grade students that they make a garden of their own for their spring Fine Arts Festival. As part of this three-month unit, she introduced them to the artwork of French Impressionist Claude Monet and his garden in Giverny, France. Together, the teacher and her students identified three major areas of Monet's garden: the Japanese footbridge with pond, the willow tree, and the formal gardens. Through discussion and collaboration, the students also decided they wanted to create an arbor to welcome guests to their garden.

This teacher had never undertaken a project of this magnitude with her middle school students, yet the teacher and her students were invested in this art installation, and together they were determined to create their garden. The teacher assigned project managers (students whose primary role was to update their task lists and help others identify their next task to be completed) to help manage the time and the tasks.[1]

As the students began constructing their garden, the teacher realized she did not have all the answers to the many questions they asked. However, she saw this as an opportunity to teach her middle school students how to tackle unknown issues and devise solutions to their obstacles. She would say things like "I have no idea how we can make that. Let's do some research" or "Sounds like we need to build a prototype first." She was very transparent with her students regarding what she knew and didn't know. In order for the students to complete their project, they had to realize that their teacher did not have all the answers, nor could she do the project alone. As a collective unit, the students and teacher worked together collaboratively to complete their art installation.

Across multiple classes during their school day and various groups of students staying after school, they all pulled together to create their version of Monet's garden. In those three months, students learned to solder, create flowers with clay, and code. They painstakingly made over 80 LED flowers by hand and wired them to three breadboard Arduinos. There was even an interactive touch component to their garden, using Makey-Makey; guests of all ages were able to touch the art in order to hear different recorded sounds. This journey of teaching and learning resulted in an art installation that was a big success at the Fine Arts Festival. The teacher wrote, "I was proud of the final project, but what made me the happiest was the pride [the students] had in their artwork and their new skills. People from around the district were so impressed and my kids felt like rock stars. . . . It was an amazing experience, and I am so grateful for the support I received throughout the process but most importantly the hard work and investment of my students." After the Fine Arts Festival, the teacher and a small group of her students were invited to present their garden at Dynamic Landscapes, a technology education conference (personal communication, November 26, 2018).

This teacher's experience integrating computational thinking into her art class began with an initial idea, an interest, and a question, and it blossomed from there. The school year concluded with the students, the teacher, and the community seeing real-life learning take place in that classroom. Since then, it has led to a new idea, a new interest, and a new question—the cycle of inquiry built on this prior experience has become part of this teacher's teaching practice.

7.5 Keep Calm and Struggle On

Productive struggle is a good thing, not a bad thing, and not something you should shy away from. When you are trying out something new, you are going to struggle, and your students are going to struggle. It is totally normal. The part that can be destructive is when that struggle becomes frustration. If you are new to programming, let your students see how you troubleshoot and solve problems that arise.

As teachers, we can help equip our students with an awareness of the different phases and processes that occur in solving problems. Acknowledging and discussing students' emotional responses—both negative and positive—can help them realize that what they are experiencing is normal (Boaler 2015; Sousa 2016). We want students to experience productive struggle so that they can learn to persevere through problem-solving situations. One of the best ways for students to learn to overcome struggle productively is to watch their teachers learn from their own mistakes and get beyond their struggle points.

We have no doubt that a teacher's ability to incorporate computational thinking with unplugged activities is strong. Where we have witnessed teachers struggle is with the coding. When teachers experience struggle, it is often because the outcome of the code is not the outcome that they wanted, or a mistake was made. We want everyone to know that this is normal. When the code is not doing what you thought it was doing, go through the code line by line and figure out where the logic went awry. When you are debugging code, acting out the action in the code exactly as it is written often helps you find the errors. Also, don't be afraid to make mistakes when you are trying something new; use them as opportunities to learn and grow as an educator. Loucks-Horsley et al. (2009), Darling-Hammond et al. (2015), and Shulman (1986) argued that effective teaching requires a recursive process of planning, teaching, and reflecting on one's practice.

7.6 Summary

In making all of you aware of the ubiquity of computational thinking and computer science principles that drive twenty-first-century life, we hope that you see how it exists within every content domain and that it is accessible to all within the different content topics. By adapting the learning experiences in your classroom, you can inspire innovation in your students and help them rise to the challenge of being comfortable with facing and addressing challenging and complex problems. Computational thinking fosters skill development that is dynamic, creative, and applies critical thinking, productive struggle, perseverance, learning from mistakes, team building, and valuing the work of others. We hope that you will take the activities and rich tasks provided in these chapters and try some of them out with your students.

The MIT Press would like to thank the anonymous peer reviewers who provided comments on drafts of this book. The generous work of academic experts is essential for establishing the authority and quality of our publications. We acknowledge with gratitude the contributions of these otherwise uncredited readers.

This book was set in Stone Serif and Stone Sans by Westchester Publishing Services.

Library of Congress Cataloging-in-Publication Data

Names: Neumann, Maureen D., author. | Dion, Lisa (Computer scientist), author.
Title: Teaching computational thinking : an integrative approach for middle and high school learning / Maureen D. Neumann and Lisa Dion with Robert Snapp.
Description: Cambridge, Massachusetts : The MIT Press, 2021. | Includes bibliographical references.
Identifiers: LCCN 2021000766 | ISBN 9780262045056 (paperback)
Subjects: LCSH: Computer science—Study and teaching (Secondary) | Critical thinking—Study and teaching (Secondary)
Classification: LCC QA76.27 .N474 2021 | DDC 004.071—dc23
LC record available at https://lccn.loc.gov/2021000766