Coding and Computational Thinking in Math and Science

Martin, Steve; Jacobsen, Michele

Alberta Teachers’ Association

http://hdl.handle.net/107763
journal article

https://creativecommons.org/licenses/by-nc/4.0

Unless otherwise indicated, this material is protected by copyright and has been made available with authorization from the copyright owner. You may use this material in any way that is permitted by the Copyright Act or through licensing that has been assigned to the document. For uses that are not allowable under copyright legislation or licensing, you are required to seek permission.

Downloaded from PRISM: https://prism.ucalgary.ca
Coding and Computational Thinking in Math and Science

Steve Martin and Michele Jacobsen

Introduction

Coding and computational thinking are generating renewed excitement among middle and high school teachers and students and are also gaining attention and momentum in schools and school jurisdictions across Canada. Unlike the command line interfaces of the past, contemporary coding software allows students and teachers to get started quickly with programming and the creation of games, simulations and other projects as part of a computational thinking curriculum. Coding and computational thinking build upon design and programming ideas developed by pioneers in the Media Lab at MIT (Seymour Papert, Yasmin Kafai, Mitchell Resnick, Idit Harel, Andreas diSessa and so on). Industry initiatives, like the hour of code, which is a global movement introducing computer science and computer programming to millions of students in 180+ countries during Computer Science Education Week, have been developed, in part, to address the identified and growing need for coding expertise across industries and organizational contexts.

The push to get students to work on coding and computer-based problem solving has resulted in some ministries of education across Canada adding coding to provincial programs of study (eg, Nova Scotia and British Columbia). While enrolment in postsecondary computer science programs grew in the 1980s and 1990s, it has levelled off at the same time that the need for programmers continues to grow. The number of people attracted to computer science is plateauing, so there is a call to engage K–12 students in coding and design experiences to help sponsor motivation for postsecondary study in computer science (Grover and Pea 2013; Information and Communication Technology Council 2016).

In this article, we provide a definition of coding and computational thinking as part of an explanation for why coding and computational thinking skills are important for all students to develop and learn as a part of their formal schooling experiences. We also provide three examples of computational thinking and design activities that have been successfully used in Alberta classrooms to illustrate teaching and learning experiences with coding, and to highlight how learning designs that promote computational thinking, and also allow students to express or develop their understandings of specific curriculum objectives, can create exciting and rich learning opportunities for learners.

What Is Computational Thinking and How Is It Different from Coding?

Coding is the creation of code—that is, the creation of a program or set of instructions that a computer executes in order to do a task. In order to create code, one needs to apply the skills that Wing (2006) identified as a new and necessary literacy: computational thinking. However, one cannot assume that computational thinking equals coding—one can engage students in computational thinking without requiring them to actively engage in coding. Wing (2006) argues that at its heart, computational thinking is about problem solving and understanding and designing systems. One can engage in computational thinking through coding, interface design, instructional design, the design of instructional software, video game design, the creation of simulations and so on. One can also engage in computational thinking by creating and building on ideas through hands-on and iterative problem-solving processes, such as the ones aligned with the Alberta elementary science curriculum—Problem Solving Through Technology activities in each grade (Alberta Education 1996). Challenges from the Problem Solving with Technology units, such as Building with a Variety of Materials, in Grade 3, or Flight, in
Grade 6, are examples of problem-solving processes that rely on students using trial and error to solve problems based on observation, adjustment, reflection and refinement, and, as such, provide important non-coding examples of students using and developing computational thinking skills without engaging in coding on a computer.

Wing (2006) proposed computational thinking as a necessary skill set, not just for computer scientists, but for everyone. Computational thinking includes analytical thinking skills that are common to computer science, and also have much broader applications to problem solving and system design problems outside the boundaries of computer science. Wing’s (2006) key concepts of computational thinking include:

• conceptualizing, not programming;
• fundamental, not rote, skills;
• solving problems, designing systems and understanding human behaviour through computer science concepts;
• a way that humans think, not a way computers think;
• complementing mathematical and engineering thinking; and
• working on ideas, not artifacts.

Grover and Pea’s (2013) review of computational thinking in K–12 education identified the many simplified programming languages and computational thinking tools that have been developed and are being used for coding in schools, including Scratch, Alice 3D, Greenfoot, Game Maker, StarLogo and Kodu. Lye and Koh (2014) found that there are many different definitions of computational thinking and the specifics of what the literacy entails. Brennan and Resnick (2012) have defined key dimensions of computational thinking as the dimensions related to the Scratch programming language and student designers.

The field-tested classroom activities described in this paper use Scratch, a free programming language developed by the Lifelong Kindergarten Group at MIT. This programming language was designed to be easy to use and was specifically built to reduce the barriers of syntax. Scratch is a simple programming language, one in which commands are built visually, using an interface that is error resistant. Commands are built from creating stacks of interlocking pieces, much like building a puzzle. If two commands cannot work together, then they cannot be combined in Scratch, which reduces the number of errors a programmer can make (Utting et al 2010). The designers of Scratch describe it as having a low floor (easy to be successful right away), wide walls (it is a general programming language that can be used to make a wide range of programs) and a high ceiling (although simple to learn and use at the beginning, it also can create complex and robust projects as the user gains experience) (Scratch Team 2013). Figure 1 shows the Scratch online programming environment, with its simple-to-use drag-and-drop interface and interlocking puzzle-piece commands.

The two of us have adopted Brennan and Resnick’s (2012) definition of computational thinking, given that the classroom projects discussed later in this paper have all been created using Scratch.

Contemporary software, like Scratch, LOGO and Boxer, makes coding and programming easier and more accessible to students than earlier programming environments; importantly, newer programs make coding and programming more accessible to classroom teachers. Through the prototyping and design of games, simulations and models using general programming languages such as Scratch, students and teachers can engage deeply with computational thinking concepts and practices at the same time that they are engaging in science and math concepts, ideas and challenges.

**Why Is Learning Coding and Computational Thinking Skills Important?**

One of the strengths of modern educational programming languages is that they are agent-based languages. In an agent-based programming language, a system can be broken down into individual elements, whereby each “agent” can be programmed separately. In the Scratch programming language, for example, the agents are the sprites and the stage, which students can program independently. Agent-based programming languages have evolved into languages with multiple agents, which can all be programmed separately but can also execute code simultaneously (Sengupta et al 2013). Teachers and researchers have found that embodied modelling allows the student programmer to think like an agent, because the student needs to understand the relationship between the code they create and the output of the agent, as well as the relationships that exist between agents (Sengupta et al 2015).

Coding and computational thinking are directly related to constructionism and design thinking.
Figure 1. The Scratch programming language’s drag-and-drop coding environment\textsuperscript{1}

Building upon seminal work by Seymour Papert, Mitch-ell Resnick, Idit Harel, Yasmin Kafai and Andrea diSessi from the MIT Media Lab, researchers have found that programming in agent-based languages, like Scratch and others, can be effective in helping students to learn science and math concepts that are otherwise abstract and challenging to understand (Martin 2016; Sengupta and Farris 2012).

What Are Effective Ways of Teaching Computational Thinking to Students?

Teachers can adopt signature pedagogies, which reflect “what counts as knowledge in a field and how things become known” (Shulman 2005, 54), to promote design thinking, inquiry-based learning and problem-based learning and to create flexible boundaries for coding and computational thinking work by students. During the first author’s thesis research (Martin 2016), the two of us found that when a thoughtful and purposeful task is created for students, students can be successful at using the Scratch programming language to create scientific models. The early solutions that students create to solve problems may not always be the most elegant; how-ever, as students engage in the iterative design process and the debugging process over time (both of which are key ideas in computational thinking), students begin to see more efficient and elegant solutions, especially if, as an intentional part of the learning process, they are allowed to share and demonstrate their solutions with peers.

\textsuperscript{1}All images from the Scratch website used in this paper are reused here under the Creative Commons Attribution-ShareAlike license. Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See http://scratch.mit.edu.
Table 1
Summary of Computational Thinking Framework (Brennan and Resnick 2012)

<table>
<thead>
<tr>
<th>Computational Thinking Key Dimension</th>
<th>Concept</th>
<th>Definition of Key Understanding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concepts found in Scratch that can be used to design.</td>
<td>Sequence</td>
<td>A sequence contains a set of steps that a computer executes in order.</td>
</tr>
<tr>
<td></td>
<td>Loop</td>
<td>A sequence can be repeated in a loop. This can be an iteration of a particular number of times or an infinite number of times.</td>
</tr>
<tr>
<td></td>
<td>Events</td>
<td>Something that happens on the computer can cause something else to happen.</td>
</tr>
<tr>
<td></td>
<td>Parallelism</td>
<td>Multiple sequences can happen and run at the same time.</td>
</tr>
<tr>
<td></td>
<td>Conditionals</td>
<td>A program can be written that allows for multiple outcomes, often based on a test using the word <em>if</em>.</td>
</tr>
<tr>
<td></td>
<td>Operators</td>
<td>Functions in a programming language that use mathematics, logic and/or strings (text based).</td>
</tr>
<tr>
<td></td>
<td>Data</td>
<td>Information can be stored, retrieved and updated.</td>
</tr>
<tr>
<td>Practices used by Scratchers in designing.</td>
<td>Incremental and iterative</td>
<td>Designing a program is a process that involves adapting and changing. A plan may change as ideas are developed.</td>
</tr>
<tr>
<td></td>
<td>Testing and debugging</td>
<td>A program often does not work as intended right away. Finding errors in logic, mistakes in the code are part of developing a program.</td>
</tr>
<tr>
<td></td>
<td>Reusing and remixing</td>
<td>Building on other peoples’ work. Sharing your work with others.</td>
</tr>
<tr>
<td></td>
<td>Abstracting and modularizing</td>
<td>Building something larger by working first with smaller parts.</td>
</tr>
<tr>
<td>Perspectives of students involved in design</td>
<td>Expressing</td>
<td>Designing is about creating something and sharing it.</td>
</tr>
<tr>
<td></td>
<td>Connecting</td>
<td>Designing is a social experience; working with others enriches the experience.</td>
</tr>
<tr>
<td></td>
<td>Questioning</td>
<td>Wondering about how design is used in other situations.</td>
</tr>
</tbody>
</table>

We found that students find the most success with programming when the tasks
• are based on a problem that can be broken down into smaller tasks (key computational thinking understanding),
• showcase or build on previous core curricular understandings and extend that learning through further inquiry questions,
• have flexible boundaries (and there are multiple solutions) and can be differentiated based on student experience, and
• incorporate an explicit design thinking process, and students use that process when generating their ideas and solutions.

What follows are some field-tested classroom examples of science and mathematics curriculum projects, using the Scratch programming language, that involve students in computational thinking and coding. The examples are discussed in the context of both Alberta Education’s programs of study and Wing’s (2006) ideas about computational thinking.
Classroom Examples of Coding/Computational Thinking Projects

Grade 6—Model of the Earth/Moon/Sun System (Science)
(seven hours of class time)

One of Wing’s (2006) key ideas in computational thinking is that students need to look at designing systems as a problem-solving exercise. Martin (2016) reports on a mixed-methods descriptive and exploratory case study that examined coding and computational thinking in two Grade 6 classes in an elementary school (39 students) where the students were designing a simulation of a system. Over 85 per cent of the students in the study were new to computer programming, and their classroom teacher had not worked with computer programming as part of her program.

After the students had completed their Sky Science unit with their teacher, and after four 45-minute lessons using the Scratch Challenge Cards (Rusk 2011) and the Scratch Curriculum Guide (Brennan, Chung and Hawson 2011), the teacher and researcher gave students a design task linked to the Sky Science unit in the Alberta elementary science curriculum (Alberta Education 1996). The design task was to create, in Scratch, a working model of the Sun–Earth–Moon system to demonstrate how the positions of those three bodies were related to the phases of the Moon as seen from Earth.

The design task was created in collaboration between the classroom teacher and the researcher, and then broken down into the seven main components that would be required to simulate the Sun–Earth–Moon system (Table 2).

<table>
<thead>
<tr>
<th>Tasks for Student-Designed Simulation/Models as Presented to Students</th>
<th>Alberta Elementary Science Program of Study Objectives, Topic C: Sky Science General Learner Expectations (GLEs) (Alberta Education 1996, 32)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The Earth revolves once per day.</td>
<td>GLE 6-7.3 Recognize that the apparent movement of objects in the night sky is regular and predictable, and explain how this apparent movement is related to Earth’s rotation.</td>
</tr>
<tr>
<td>2. The Earth has night and day represented.</td>
<td>Not explicitly mentioned in curriculum, although it is generally taught as part of the Earth–Moon system.</td>
</tr>
<tr>
<td>3. There is a sprite that shows a picture of the Moon phases for each of the 28 days of the lunar cycle.</td>
<td>GLE 6-7.8 Illustrate the phases of the Moon in drawings and by using improvised models.</td>
</tr>
<tr>
<td>4. There is a label or text somewhere on the screen that names each phase as it occurs in your simulation.</td>
<td>GLE 6-7.7 Recognize that the Moon’s phases are regular and predictable, and describe the cycle of its phases.</td>
</tr>
<tr>
<td>5. The Moon orbits the Earth in 28 days.</td>
<td>GLE 6-7.7 Recognize that the apparent movement of objects in the night sky is regular and predictable, and explain how this apparent movement is related to Earth’s rotation.</td>
</tr>
<tr>
<td>6. There is an arrow on the provided sprite of the Moon that always points at Earth.</td>
<td>Not explicitly mentioned, but generally taught as part of the Earth–Moon system.</td>
</tr>
<tr>
<td>7. The Moon Needs to have night and Day</td>
<td>GLE 6-7.1 Recognize that the Sun and stars emit the light by which they are seen and that most other bodies in space, including Earth’s Moon, planets and their Moons, comets, and asteroids, are seen by reflected light.</td>
</tr>
</tbody>
</table>
The students were given seven hours of class time to work with partners to create a program that met as many of the seven requirements as possible. Students were given a construction kit that contained the relevant graphics for their program (Figure 2).

Within the time period allowed for the design task, all of the Grade 6 students experienced at least some success at meeting the challenge. All of the groups successfully programmed at least three of the tasks required for their model, and several groups (6 out of 14) completed all seven of the required tasks.

The group projects were analyzed and reviewed for evidence that the students demonstrated the computational thinking concepts as proposed by Brennan and Resnick (2012). We found that, even without explicit instruction in the computational thinking concepts, all of the students' projects had evidence that students engaged with the concepts of sequence, simple events, broadcast events, conditionals and operators. Further, we also found that more than 80 per cent of the students had used loops and parallelism. The only concept that was used by only a small number of students was the data concept.

The observational data that was collected supported the finding that students had engaged in several of the computational thinking practices proposed by Brennan and Resnick (2012). Multiple examples were found of students using the incremental and iterative process and the testing and debugging process as they developed their models. As for the abstracting and modularizing process, the task’s initial design gave the student the model for how to break up larger concepts into small modularized components and was evident as students worked on developing the pieces of code that would make the program execute the tasks.

A good example of the computational thinking processes being used involved the students solving the problem of showing night and day on the Earth as the Earth rotated. The initial solution used by many groups involved shading in half of the Earth, much as they had done on their paper-and-pencil diagrams. However, running the simulation created a night that rotated with the Earth, putting the shaded in half on the side nearest the sun. This solution resulted in an interesting discussion with students about how to simulate night and day.

Figure 2. Screen shot of construction kit with graphics given to students
Researcher: Tell me about the night and day on Earth [referring to their program].
Student K1: It is a shadow. Dark at night. Light in day.
Researcher: So, how does it relate to the sun?
Student K1: What do you mean?
Researcher: Well, how does where the sun is in space make it night or day on Earth?
Student K1: Well, the light from the sun hits the Earth here.
They run their program.
Student K2: Oh ... wait a minute. The shadow needs to be on the side away from the sun.
Students quickly realized that creating a shadow would require a new sprite. In their model, the sun and the shadow did not move, but the Earth revolved underneath. Testing and debugging, an iterative and incremental process, and abstracting and modularizing are all necessary analytical skills needed for the students to figure out a solution that allowed them model night and day, even though we did not explicitly inform the students that that was required.

In addition to the computational thinking experiences identified in this project, we found that many ideas that students had about the content they had learned during the Sky Science unit were solidified as the students worked through the programming tasks. One teacher–student discussion that highlights the value of this type of project in solidifying understanding was about the direction of the Earth’s revolution. Scratch uses arrows of direction to show rotation, rather than words.

Student A1: Which way does the Earth rotate?
Teacher: Oh, come on, you know the answer to this. It’s in your notes.
Student A1: No, I know it is counter-clockwise.
Teacher: Right.
Student A1: But which one is that in Scratch? (The student is pointing to the motion programming blocks.)

Although the student was able to state the correct vocabulary to describe the motion, the practical example of needing to make the model work revealed a crucial piece of information that we assume a student can answer when they state that something revolves counter-clockwise.

In an online survey conducted after the project was over, 74 per cent of the students agreed or strongly agreed that “creating a program about the Moon and the Earth helped me understand how lunar phases work.”

Many studies involving the use of the Scratch programming language, specifically in core curriculum subjects, have found that students experience positive feelings about using Scratch to show an understanding of curricular concepts and were excited and eager to use Scratch (Baytak and Land 2011; Burke 2012; Calder 2010; Wolz et al 2011). We found that the majority of students enjoyed using Scratch, 82 per cent agreeing that Scratch was easy to use and 82 per cent agreeing that computer programming was fun.

Grade 8—Developing Medical Sensors (Science and CTF)
(10 hours of class time)

Students’ use of abstraction and modularizing was one of the key learning focuses in a science/CTF project about medical sensors. As part of the first author’s work with the Design the Shift project through the Calgary Board of Education Summer Institute in 2014, he designed a cross-curricular project to go with his classes’ yearlong essential question in Grade 8 math/science: “How do we know when something ‘makes sense?’” Framed within this inquiry question, and integrated with the Cells and Body Systems unit in Science 8, students were given the following challenge:

You will create a noninvasive sensor, either from scratch or by repurposing an already existing sensor (say, a microphone), that will send information to a computer program (on computer or cellphone/tablet) or a mechanical device of your creation. When the computer program or device receives the information from your sensor it will display the information in a meaningful way to the people who are using the sensor. The following constraints exist in this project:

• Your sensor may not make direct contact with a body fluid. You may not collect fluid or tissue samples for your sensor.
• Your sensor proposal must be approved by the teacher before it is constructed, to ensure that your proposed sensor poses no risk of injury.
• The representation of the data that is shown on the computer or mechanical device is changed as the information arrives in real time.
• You need to decide if your program reports only in real time or if it also records the data.
• You need to develop a page explaining how the user’s information is used by the program, including any actions that you are taking to make information private or public and the rationale for your choice.

The key understandings in computational thinking that are required for students to make this project a success are abstracting and modularizing the problem. Students have to think about the information they want to collect and then develop several prototypes of the sensor as they try to get their software/computer to receive the information. Once the information is received, the student has to develop a way to present the information and/or preserve the information over time so that finally they can present the data that they have collected. This type of project planning was new to many of the students who were used to coming up with a single solution or looking at the problem holistically.

During this task, Grade 8 students created several interesting successful projects, including
• a sensor made out of tinfoil and connected to Makey Makey to time the patellar reflex (ie, the reflex that occurs when one is hit just below the knee and the leg moves) and to collect information about the average reaction time in Grade 8 boys and girls,
• a sensor using a microphone and a paper towel tube that heard a heart beating and ran an animation of the heart beating in real time, and
• a breath rate calculator, using a microphone and a paper bag, that timed how many breaths in and out a Grade 8 student took in one minute.

Unsuccessful projects often showed ingenious potential and great ideas, even if the project proved to be beyond the teacher’s and students’ technical ability to create the project. Unsuccessful projects often resulted in a greater understanding of the system as a whole, as groups were asked to think about what changes in design or what different technologies we would need to have access to in order to make the projects successful. Examples of great ideas include
• a camera one could use on urine in a toilet bowl to determine how hydrated a person was,
• shining a light from a cellphone LED through a person’s finger to detect one’s pulse and
• building a blood pressure cuff and having it report to a computer the person’s blood pressure.

Although only 75 per cent of the groups were successful in meeting the goals for the entire challenge, all students gained an appreciation for both the complexity of the human body a realization of how the data that was collected needed to be interpreted as part of a medical practitioner’s work, and a deeper understanding of how their miniature computational system was similar to the other systems that the class had been studying throughout the year in science. Finally, the student reflections showed a developing understanding of how a computer cannot make sense of the data it receives without a human programming it to make sense of the data it receives.

A key element of the medical sensors project was challenging students to discuss not just the coding and how the computer would “make sense” of the data it was receiving from the sensors, but also to provide an opportunity for students to think about the ethics questions surrounding the information they collected. Questions such as, Who owns the information? Does the researcher own it? Does the school own it because the data was on the school computer? and Do you own the information collected from your body? required students to engage in thoughtful inquiry about broader ethical issues. With the use of wearable fitness technology, the class engaged in many rich discussions about exactly how the data these types of devices collect is used and by whom. Each of the ethical discussions engaged students in rich discourse and ongoing reflection about the ramifications of their data collection. Noncoding examples of computational thinking occurred as students had to think about their projects in the context of the wider system of the real world, governments and ethics. The students’ ideas were quickly extrapolated to wearable technologies and cellphones that collected information about the people using them, and this became the foundation of logical arguments using if/then statements about how we should think about protecting this data.

Grade 8—Pythagorean Triple Calculator (Mathematics)

(1 hour of class time)

Wing (2006) describes one of the skills in computational thinking as “reformulating a seemingly difficult problem into one we know how to solve, perhaps by reduction, embedding, transformation, or simulation” (p 33). In this mathematics example, the challenging
problem of identifying Pythagorean triples is solved by the students creating simple software by relying on operators and variables to reduce the problem into something that is easiest to test. Unlike the other two projects described in this paper, the Pythagorean Triple Calculator task is an example of a relatively quick application of computational thinking to the curriculum.

A Pythagorean triple is a set of three integers that are the sides of a right angle triangle. For example, the smallest Pythagorean triple is 3, 4 and 5. After finding a few triples in class and using just our calculators and the Pythagorean theorem, the teacher and students decided they needed a larger data set to seek patterns. The group wanted to know if there was a way to predict triples in advance or if there was a formula that would calculate triples.

The teacher challenged the students to use Scratch and their existing knowledge of the Pythagorean theorem to create a program that would allow them to find all of the triples where \( c \leq 100 \). Although the students did not have a programming technique that would allow them to instruct the computer to find all of the sets of triples, they did have a basic understanding of how to use variables and formulas. Existing knowledge of variables and formulas allowed students to use a trial-and-error method to test whether the numbers they had were a true triple. The most common solution to this challenge involved a visual inspection of the numbers by the students (Figure 3).

Once students had figured out a way to control or input variables for \( a \) and \( b \), they used the Pythagorean theorem to calculate \( c \). An example of the code used for this task is shown in Figure 4. As part of the task, students were required to determine how they would know if their formula was working. For example, students needed to figure out whether Scratch followed the order of operations. When the teacher asked students who were using their program how they knew they were getting accurate results, the students started to realize that they needed to test their program against known results, which is a key part of computational thinking, rather than just assuming that their formula was correct. Once the students determined through testing that their code was indeed working as expected and they were confident that the results they were generating were accurate, they proceeded to hunt for triples.

Most of the students' programs, either by using a timer or using the arrow keys, showed each combination in which \( a \) and \( b \) were integers that could be checked. If the calculation resulted in a non-integer for \( c \) (as shown in the number beside the dinosaur, Figure 3) the students knew that it was not a triple. The students' knowledge about non-integers later became useful when students needed to test their hypothesis for patterns. Students could take their set of numbers generated from their hypothetical pattern and quickly test to see whether it was correct.

**Conclusion**

The current call for educators to provide coding and computational thinking experiences for all of our students is one the authors argue that Alberta educators need to answer, and soon. Along with ministries of education in other Canadian provinces, Alberta Education needs to consider adding coding and computational thinking to the program of
students. Research and practice on coding and computational thinking in schools has documented the learning benefits when young people are engaged in meaningful work that is challenging and worthwhile and when they are supported by engaged teachers who provide regular feedback on their learning (Jacobsen, Lock and Friesen 2013). Based on current research and three classroom projects in Alberta that have combined programming with science and mathematics, the authors have illustrated the learning benefits of coding; these include computational thinking and programming tasks as rich discipline-based inquiry, problem-based learning and inquiry-based learning experiences, and also as design tasks that enable students to explore mathematics and science concepts and to build on their understanding of the world around them. Our research has demonstrated that when carefully thought out and well-designed tasks are given to Alberta students as part of how they demonstrate their understanding of curriculum concepts, students can gain experiences in computational thinking, even if they are not explicitly taught step-by-step methods for creating a program, and solidify their understanding of the curricular concepts as well. Through the classroom examples provided, one can conclude that, even if the students have very limited coding experience and create solutions that lack elegance, student engagement in design tasks allows students to gain quality computational thinking experiences and develop a deeper understanding of how the information learned in class can be applied using technology. Teachers’ designs of design tasks and creating opportunities for students to use coding to learn and demonstrate their understanding of concepts from the math and science curriculum, rather than searching for opportunities for students to learn to code, will allow students to develop a deep and rich understanding of computational thinking as a tool to understand the world around them.

References


