CHAPTER

4

Robotics and Computational Thinking

Unless you've been hiding under a rock or perhaps just too busy to be paying attention, you'll know about the current trend in K–12 education to have schools incorporate computational thinking into classroom instruction. However, CT isn't new. Let's briefly explore its historical uses, gain a better understanding of how it's applied, and get some additional resources for CT before integrating it with robotics.

Computer scientist Seymour Papert was the first to use the term CT in education (1980) and advocated for a pragmatic approach to knowledge construction using computers and the Logo educational programming language. Papert believed this could assist learners in developing powerful critical thinking and reasoning skills. Shortly after that, theoretical physicist Ken Wilson made significant advances in the field of computational science with computer software that used detailed algorithms to carry out simulations. Consequently, leaders in computational science used CT to describe the habits of mind they developed, and many reported learning CT by designing computations, instead of studying CS (Guzdial, 2015).

In 2006, CT was introduced in education by CS professor Jeannette Wing (Wing 2006). She described CT as a thinking tool used to solve problems and promoted the idea that CT builds on both human and computing processes. The field of

computing education research became revitalized due to Wing's introduction, along with well-publicized breakthroughs in computational science and rapid digitization of society's infrastructure and industry's major functions.

Then, with the support of scientists who applied CT across various fields as well as congress, CS was included in the definition of STEM in 2015, and the National Science Foundation (NSF) supported the current movement to incorporate both CT and CS into K–12 education.

Table 4.1 **Introducing the four elements of CT to students**

CT ELEMENT	DESCRIPTION	METHOD FOR INTRODUCING
Decomposition	Decomposition helps learners break down complex problems into easier to manage pieces.	The best way to start is with a task they already do. Like brushing their teeth, tying a shoelace, or making breakfast.
Pattern Recognition	Pattern recognition is mapping similarities and differences in decomposed problems, which is an important skill to have for making predictions.	This can be introduced to students by showing them images of various animals, desserts, or even text.
Abstraction	Abstraction is simply the process of taking away or removing features from something in order to make a set of essential features. It has also been described as the process of removing unnecessary features or removing the fluff (which I like).	The abstraction process can be leveraged by having students plan a party, vacation, or trip to the movies and then solve problems by removing all irrelevant details and patterns.
Algorithm design	Algorithm design is creating step-by-step logical series of instructions (algorithms) for solving problems.	Start students out by having them create logical steps to completing a familiar task and then have them flowchart it using the recommended universal symbols.

Although there is no exact definition of CT, it is widely accepted in education that CT has four elements (decomposition, abstraction, pattern recognition, and algorithm design) and that learning and applying CT helps learners understand the logic and algorithmic processes that are the foundation of both hardware and software designs. I, therefore, believe that using CT is critical for designing and working with robotics. Furthermore, educators can leverage the power of CT as a higher-order problem-solving skill by helping students build competency in CT by developing their versatility for recognizing and applying the four elements of CT to everyday situations before doing so with robotics. Table 4.1 shares a breakdown of the four elements along with suggested methods for introducing them to students.

How CT Differs from CS

Simply put, computational thinking is a problem-solving process (or set of processes and skills). Computer science (CS) is a discipline, CT is not.

Resources for Computational Thinking

ISTE: *Computational thinking for all* (bit.ly/30cDqsV)

How-to blog with a plethora of CT resources.

ISTE: *How to develop computational thinkers* (bit.ly/2HONOjU)

How-to blog with strategies

for developing computational thinkers across the disciplines.

According to the K–12 Computer Science Framework (2016), CS is a discipline that is part of computing education. Computing education in K–12 schools includes computer literacy, educational technology, digital citizenship, information technology, and computer science (p. 13). As the foundation for all computing, computer science is "the study of computers and algorithmic processes, including their principles, their hardware and software designs, their applications, and their impact on society" (Tucker et. al, 2006, p. 2).

Because CT is a problem-solving skill, it can also be applied in disciplines other than computer science. This book aims to eliminate a lot of the guesswork for educators from the various disciplines attempting to implement CT by providing a plethora of ready-to-use resources in the forthcoming pages.

The K–12 Computer Science Framework is an excellent source for either beginning or improving our understanding of how to incorporate CT into instruction. The framework focuses on core concepts and core practices that describe what students should know about CT/

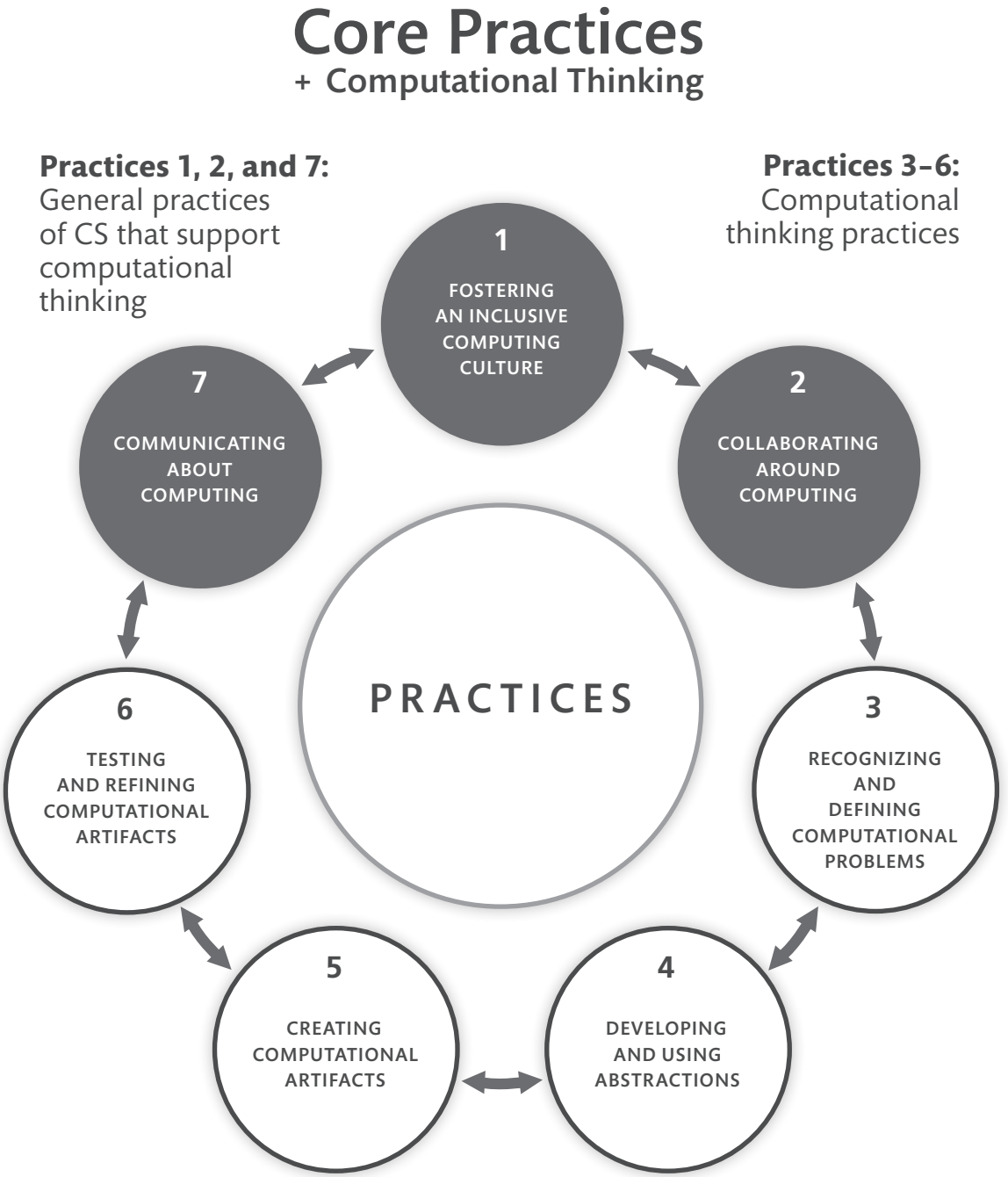

Figure 4.1: K–12 Computer Science Framework Core Practices Highlighting Computational Thinking.

CS and what they should be able to do with that knowledge. It also shows how to connect CT/CS to other learning (i.e., ELA, math, science, engineering, etc.). The diagram of the core practices (shown in Figure 4.1) shows as shaded circles those practices (1, 2, and 7) that support CT.

To complement both the K–12 Computer Science Framework and the CSTA K–12 Computer Science Standards for Students, ISTE developed the CT Competencies for educators to help schools integrate CT across all disciplines and with all students by correlating it to what they already teach.

It is also important for teachers to understand that the CS standards and CT Competencies are not just about programming and coding; and were intentionally written to align with the academic subjects that they teach. Now that computers are part of everything we do and educational technology must be used by all teachers to augment instruction, these competencies are extremely helpful with assisting learners with becoming computational thinkers who can leverage computing to solve problems innovatively.

ISTE Computational Thinking Competencies

▶ *1. Computational Thinking (Learner)*

Educators continually improve their practice by developing an understanding of computational thinking and its application as a cross-curricular skill. Educators develop a working knowledge of core components of computational thinking: such as decomposition; gathering and analyzing data; abstraction; algorithm design; and how computing impacts people and society. Educators:

a. Set professional learning goals to explore and apply teaching strategies for integrating CT practices into learning activities in ways that enhance student learning of both the academic discipline and CS concepts.

b. Learn to recognize where and how computation can be used to enrich data or content to solve discipline-specific problems and be able to connect these opportunities to foundational CT practices and CS concepts.

c. Leverage CT and CS experts, resources and professional learning networks to continuously improve practice integrating CT across content areas.

d. Develop resilience and perseverance when approaching CS and CT learning experiences, build comfort with ambiguity and open-ended problems, and see failure as an opportunity to learn and innovate.

e. Recognize how computing and society interact to create opportunities, inequities, responsibilities and threats for individuals and organizations.

▶ *2. Equity Leader (Leader)*

All students and educators have the ability to be computational thinkers and CS learners. Educators proactively counter stereotypes that exclude students from opportunities to excel in computing and foster an inclusive and diverse classroom culture that incorporates and values unique perspectives; builds student self-efficacy and confidence around computing; addresses varying needs and strengths; and addresses bias in interactions, design and development methods. Educators:

a. Nurture a confident, competent and positive identity around computing for every student.

b. Construct and implement culturally relevant learning activities that address a diverse range of ethical, social and cultural perspectives on computing and highlight computing achievements from diverse role models and teams.

c. Choose teaching approaches that help to foster an inclusive computing culture, avoid stereotype threat and equitably engage all students.

d. Assess and manage classroom culture to drive equitable student participation, address exclusionary dynamics and counter implicit bias.

e. Communicate with students, parents and leaders about the impacts of computing in our world and across diverse roles and professional life, and why these skills are essential for all students.

▶ *3. Collaborating Around Computing (Collaborator)*

Effective collaboration around computing requires educators to incorporate diverse perspectives and unique skills when developing student learning opportunities, and recognize that collaboration skills must be explicitly taught in order to lead to better outcomes than individuals working independently. Educators work together to select tools and design activities and environments that facilitate these collaborations and outcomes. Educators:

a. Model and learn with students how to formulate computational solutions to problems and how to give and receive actionable feedback.

b. Apply effective teaching strategies to support student collaboration around computing, including pair programming, working in varying team roles, equitable workload distribution and project management.

c. Plan collaboratively with other educators to create learning activities that cross disciplines to strengthen student understanding of CT and CS concepts and transfer application of knowledge in new contexts.

▶ *4. Creativity & Design (Designer)*

Computational thinking skills can empower students to create computational artifacts that allow for personal expression. Educators recognize that design and creativity can encourage a growth mindset and work to create meaningful CS learning experiences and environments that inspire students to build their skills and confidence around computing in ways that reflect their interests and experiences. Educators:

a. Design CT activities where data can be obtained, analyzed and represented to support problem-solving and learning in other content areas.

b. Design authentic learning activities that ask students to leverage a design process to solve problems with awareness of technical and human constraints and defend their design choices.

c. Guide students on the importance of diverse perspectives and human-centered design in developing computational artifacts with broad accessibility and usability.

d. Create CS and CT learning environments that value and encourage varied viewpoints, student agency, creativity, engagement, joy and fun.

▶ *5. Integrating Computational Thinking (Facilitator)*

Educators facilitate learning by integrating computational thinking practices into the classroom. Since computational thinking is a foundational skill, educators develop every student's ability to recognize opportunities to apply computational thinking in their environment. Educators:

a. Evaluate and use CS and CT curricula, resources and tools that account for learner variability to meet the needs of all students.

b. Empower students to select personally meaningful computational projects.

c. Use a variety of instructional approaches to help students frame problems in ways that can be represented as computational steps or algorithms to be performed by a computer.

d. Establish criteria for evaluating CT practices and content learning that use a variety of formative and alternative assessments to enable students to demonstrate their understanding of age-appropriate CS and CT vocabulary, practices and concepts.

As you can see, there is a lot to unpack within the indicators of the CT competencies. In the context of robotics, my suggestion is to focus on how you want to teach CT and see evidence of its elements as students plan, build robots, program robots, and test and evaluate their designs and final solutions. Also, bear in mind that these competencies serve as guidelines for you to unpack and correlate to the standards in your content area—in tandem with the many of the concepts and practices found in the K–12 Computer Science Framework and the CSTA K–12 Computer Science Standards for Students.

Introduction to Computational Thinking for Every Educator (ISTE U Course)

To assist educators with building and exploring digital age competencies (spanning hot topics in both computer science and edtech), ISTE has created ISTE U.

ISTE U is a curation of engaging professional learning experiences via a virtual hub for educators to experience anywhere and at their own pace. The courses are eligible for graduate-level credit through the Continuing Education and Professional Development Department at Dominican University of California.

The Introduction to Computational Thinking for Every Educator course was developed in collaboration with Google to guide and supports educators as they design and plan lessons that integrate CT across all disciplines and grade levels. Moreover, the course increases both awareness and understanding of how CT can be incorporated into a school's curricula plan.

Integrate CT with the ISTE CT Competencies

ISTE Computational Thinking Competencies for Educators (bit.ly/2CVPdBt)

Integrate CT across disciplines and with all students.

Get support for CT integration with ISTE U!

ISTE U: ***Introduction to Computational Thinking for Every Educator*** (bit.ly/2V91MGA) Online course assisting educators with integrating computational thinking across all disciplines and grade levels.

Start Teaching Computational Thinking with Unplugged Lessons

The purpose of beginning CT with unplugged activities is to help students make connections between CT and previous learning, to clear up any misconceptions about CT, and to promote critical thinking and problem-solving in fun and engaging ways while building the right background knowledge needed for CS skills such as designing and programming robots. The lessons are labeled unplugged because they are taught without using computers or tech tools.

So, if you're wanting to engage students who are new to CT and CS and do not have any prior knowledge, look no further than CS Unplugged (csunplugged.org) for a variety of unplugged activities. Some of which include the following:

1. **Binary numbers:** Six lessons, ages 5–10 with connections to art, literacy, and Music.
2. **Kidbots:** Four lessons, ages 5–10 with four curriculum integrations and 50 programming challenges.
3. **Sorting networks:** Four lessons, ages 5–14 with four curriculum integrations.
4. **Error detection and correction:** Three lessons, ages 5–10 with five curriculum integrations and 24 programming challenges.
5. **Searching algorithms:** Six lessons, ages 5–10 with four curriculum integrations.

RESOURCE

CS Unplugged Lessons and Printables

CS Unplugged: ***Unplugged computer science lessons***

(csunplugged.org) Teach computer science without a computer!

Unplugged Lesson

Computational Thinking

The following Code.org unplugged lesson is one that I highly recommend teachers use when starting to teach CT. The lesson doesn't have a specific grade level and can be adapted to most K–8 scenarios and content areas. However, K–2 teachers will need to provide more scaffolds to support students who are not yet able to read.

Unplugged Lesson: Computational Thinking
code.org/curriculum/course3/1/Teacher

Overview

In this lesson, students build competence for the four CT elements (decomposition, pattern matching, abstraction, and algorithms) by using examples of what fictional players have done to figure out how to play an actual game. As students learn to put into practice the four elements of CT in one cohesive activity, the lesson provides them with the foundational problem-solving skills needed for designing and programming with robotics. Think of it as stacking building blocks to form the foundation for a much bigger and more complex structure.

The lesson provides steps for teachers to take for the following:

- Unpacking the vocabulary for CT
- Reinforcing the CT practices for students with user experience scripts found in the CT kit
- Pattern matching and abstraction with a color, animals, and an object
- Pattern matching is augmented mathematically when adding and finally multiplying. Here teachers can help students follow the same steps but with different numbers as the lesson suggests, ensuring they understand the concepts but in different scenarios
- CT assessment

Duration

The basic lesson time is 25 minutes and only includes the activity. If time permits, introductory and wrap-up suggestions can be implemented to dive deeper into the subject matter, extend work time, and allow you to make concrete connections to robotics and programming. It is also important to note that time will vary depending on your students' reading and writing ability. My suggestion is to build in some flex time and allow for multiple opportunities to display mastery.

Objectives

STUDENTS WILL

- Analyze information to draw conclusions
- Match identical portions of similar phrases to match patterns
- Identify differences in similar phrases and abstract them out

Vocabulary

The lesson introduces students to the 4 elements of CT.

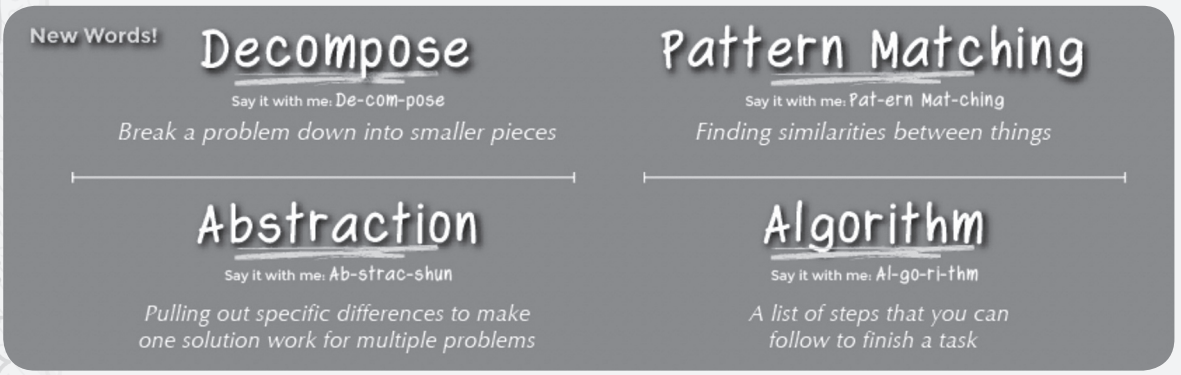

Figure 4.2: Computational thinking lesson vocabulary.

Students who are not independent readers yet will need assistance with sounding out the elements. Tips on sounding out each of the elements by syllable are provided along with the definitions (see Figure 4.2).

Materials, Resources, and Teacher Prep

FOR STUDENTS

- One die per group
- One Computational Thinking Kit per group
- Pens, pencils, and scissors
- Computational Thinking Assessment for each student

FOR TEACHER

- Lesson video
- Teacher lesson guide
- One printed Computational Thinking Kit for each group
- One printed Computational Thinking Assessment for each student

Warm Up

Inform students that they will sum up all the numbers between 1 and 200.

- To ease any anxieties that they may have, be sure to express that this is not a graded exercise.
- Now, inform them that they must do it all in their heads.
- Add the time constraint of thirty seconds.
- They may feel overwhelmed. This is intentional. You can indicate with your tone and demeanor that you might be crazy asking this of them but begin timing with a resounding: "Starting NOW."
- Watch the class as you keep time. How many are lost in thought?

When time is up, ask if anyone was able to get the total.

- Ask if there is anyone who thought the problem was so hard that they didn't even attempt it.
- Did anyone attempt it and just not finish? What did they try?
- Guide students toward thinking a little smaller.

Explain, "If we break the problem up into smaller pieces, it becomes easier to manage."

- Let's start at the two ends. What is 200 + 1?
- What is 199 + 2?
- What is 198 + 3?
- See a pattern?

Ask, "How many of these pairs will we have?"

- What is the last pair we will find? 100 + 101.
- That means that we have 100 total pairs.
- If we have 100 total pairs of sums of 201, how do we find the final total?
- What is 100 × 201?

Ask, "Now, what if we wanted to find the trick to do this with other numbers?"

- Can we do it easily with 2,000?
- How about 20,000?
- What stays the same? What is different?
- If we use abstractions to make our end goal something that can change (say we name it "blank") then we can make an algorithm that will work for any number.
- Work through the problem until you ultimately arrive at

  ? = ("blank"/2) × ("blank"+1)

- Do a few simple examples to show that the algorithm is correct for

  blanks = 2, 3, 4, and 5

Finally, you can say something like this: "This is all to show that if you use the tools of computational thinking (decomposition, pattern matching, abstraction, and algorithms), you can figure out how to solve problems that no one has already taught you how to solve . . . just like we did here! This will be an extremely powerful skill for the rest of your life!"

Activities (25 min)

Computational Thinking

This lesson is all about a "Game with No Instructions." Students will be charged with figuring out how to play the game as a small group. The small details of their final algorithm are unimportant. What *is* important is that they were able to take a huge task like figuring out how to play a game on their own and take small steps toward achieving the goal.

Students will be guided toward discovering the rules using the steps of computational thinking. Resist the temptation to point the students toward "doing it right" and instead allow them to just do it on their own. If they feel stumped or confused, encourage the students to look at the information that has been given to them, or if they must, allow them to ask a classmate.

Directions

1. Divide students into groups of 2–4.
2. Have the groups read over user experiences to get an idea of how other students have played the "Game with No Instructions."
3. Encourage them to pattern match between each experience by circling the sections of words that are identical from player to player.
4. Next, have them abstract away differences from each experience by underlining words that change from player to player.

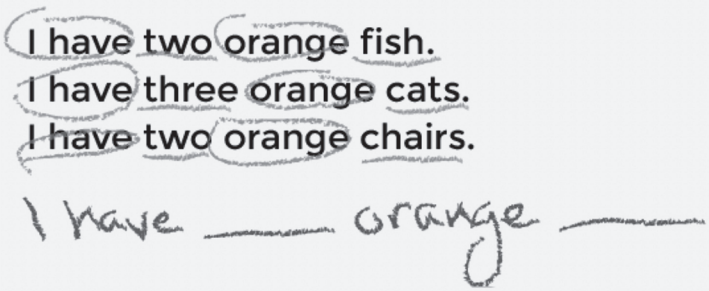

Figure 4.3: Finding patterns in computational thinking unplugged lesson.

5. Using pattern matching and abstraction, have them make a script template for gameplay by writing up the circled parts of the other students' experiences and leaving the underlined sections as blanks (see Figure 4.3).

6. Give students a blank sheet of paper to write a list of instructions for how they think this game should be played based on the user experiences that they just read. This will be their algorithm.

7. Have students play the game using the algorithm that they just made. Each player should get at least two turns.

Reflection and Wrap-Up (5 min)

What did we learn? Intended to get students thinking about the big picture and how the lesson relates to real-world situations, these questions can be discussed as a class, in groups, or among partners.

- What should you try to do when you're asked to do something and you don't know how?
- If a problem is too hard, what should you try to do?
- If you find similarities in lots of solutions to different problems, what does that probably tell you?
- If you have a problem that is just a little different from a problem that you have a solution for, what would you do?

Assessment (10 min)

- Hand out the assessment worksheet and allow students to complete the activity independently after the instructions have been well explained. This should feel familiar, thanks to the previous activities.

Standards Addressed

ISTE Standards for Students

- **1.a.** Articulate and set personal learning goals, develop strategies leveraging technology to achieve them and reflect on the learning process itself to improve learning outcomes.

- **3.d.** Build knowledge by actively exploring real-world issues and problems, developing ideas and theories and pursuing answers and solutions.
- **4.a.** Build knowledge by actively exploring real-world issues and problems, developing ideas and theories and pursuing answers and solutions.
- **5.a.** Formulate problem definitions suited for technology-assisted methods such as data analysis, abstract models and algorithmic thinking in exploring and finding solutions.
- **5.c.** Break problems into component parts, extract key information, and develop descriptive models to understand complex systems or facilitate problem-solving.

CSTA K-12 Computer Science Standards

- **CPP.L1:6-05.** Construct a program as a set of step-by-step instructions to be acted out.
- **CT.L1:6-02.** Develop a simple understanding of an algorithm using computer-free exercises.
- **CT.L2-01.** Use the basic steps in algorithmic problem solving to design solutions.
- **CT.L2-06.** Describe and analyze a sequence of instructions being followed.
- **CT.L2-08.** Use visual representations of problem states, structures, and data.
- **CT.L2-12.** Use abstraction to decompose a problem into sub problems.
- **CT.L2-14.** Examine connections between elements of mathematics and computer science including binary numbers, logic, sets, and functions.

NGSS Science and Engineering Practices

- **3-5-ETS1-2.** Generate and compare multiple possible solutions to a problem based on how well each is likely to meet the criteria and constraints of the problem.

Common Core Mathematical Practices

- Make sense of problems and persevere in solving them.
- Reason abstractly and quantitatively.
- Construct viable arguments and critique the reasoning of others.

- Attend to precision.
- Look for and make use of structure.
- Look for and express regularity in repeated reasoning.

Common Core Math Standards

- **3.OA.3.** Use multiplication and division within 100 to solve word problems in situations involving equal groups, arrays, and measurement quantities.
- **4.NBT.B.4.** Fluently add and subtract multi-digit whole numbers using the standard algorithm.
- **5.NBT.B.5.** Fluently multiply multi-digit whole numbers using the standard algorithm.

Common Core Language Arts Standards

- **SL.3.1.** Engage effectively in a range of collaborative discussions (one-on-one, in groups, and teacher-led) with diverse partners on grade 3 topics and texts, building on others' ideas and expressing their own clearly.
- **SL.3.3.** Ask and answer questions about information from a speaker, offering appropriate elaboration and detail.
- **L.3.6.** Acquire and use accurately grade-appropriate conversational, general academic, and domain-specific words and phrases, including those that signal spatial and temporal relationships.
- **SL.4.1.** Engage effectively in a range of collaborative discussions (one-on-one, in groups, and teacher-led) with diverse partners on grade 4 topics and texts, building on others' ideas and expressing their own clearly.
- **L.4.6.** Acquire and use accurately grade-appropriate general academic and domain-specific words and phrases, including those that signal precise actions, emotions, or states of being and that are basic to a particular topic.
- **SL.5.1.** Engage effectively in a range of collaborative discussions (one-on-one, in groups, and teacher-led) with diverse partners on grade 5 topics and texts, building on others' ideas and expressing their own clearly.
- **L.5.6.** Acquire and use accurately grade-appropriate general academic and domain-specific words and phrases, including those that signal contrast, addition, and other logical relationships.