K12 COMPUTER SCIENCE FRAMEWORK

K–12 Computer Science Framework Steering Committee

Association for Computing Machinery

CODE

CSTEACHERS.ORG
COMPUTER SCIENCE TEACHERS ASSOCIATION

cyber innovation center
Collaboration • Research • Technology

NATIONAL MATH + SCIENCE INITIATIVE

Suggested citation: K–12 Computer Science Framework. (2016). Retrieved from http://www.k12cs.org.

Suggested attribution: "The K–12 Computer Science Framework, led by the Association for Computing Machinery, Code.org, Computer Science Teachers Association, Cyber Innovation Center, and National Math and Science Initiative in partnership with states and districts, informed the development of this work."

Examples of programs and resources are provided for the reader's convenience and do not represent an endorsement.

Acknowledgments

The K–12 Computer Science Framework was a community effort. The following sections acknowledge the different individuals and organizations who played a significant role in the development of the framework.

Steering Committee

Thank you to Mehran Sahami of the Association for Computing Machinery, Cameron Wilson of Code.org, Mark Nelson of the Computer Science Teachers Association (CSTA), Krystal Corbett of the Cyber Innovation Center, and Deepa Muralidhar of the National Math and Science Initiative for guiding the framework's process.

States and Districts

The following states and districts participated in the development of the framework by nominating writers and providing feedback on the framework.

States

Arkansas
California
Georgia
Idaho
Indiana
Iowa
Maryland
Massachusetts
Nebraska
Nevada
New Jersey
North Carolina
Utah
Washington

Districts

Charles County Public Schools, MD
Chicago Public Schools, IL
New York City Department of Education, NY
San Francisco Unified School District, CA

Writers

The writers' biographies are provided in **Appendix B**.

Julie Alano
Computer Science Teacher, Hamilton Southeastern High School

Derek Babb
Computer Science Teacher, Omaha North Magnet High School

Julia Bell
Associate Professor of Computer Science, Walters State Community College

Tiara Booker-Dwyer
Education Program Specialist, Maryland State Department of Education

Leigh Ann DeLyser
Director of Education and Research, CSNYC

Caitlin McMunn Dooley
Deputy Superintendent for Curriculum and Instruction Georgia Department of Education; Associate Professor, Georgia State University

Diana Franklin
Director of Computer Science Education, UChicago STEM Ed

Dan Frost
Senior Lecturer, University of California, Irvine

Mark A. Gruwell
Co-Facilitator, Iowa STEM Council Computer Science Workgroup

Maya Israel
Assistant Professor, University of Illinois at Urbana Champaign

Vanessa Jones
Instructional Technology Design Coach, Austin Independent School District

Richard Kick
Mathematics and Computer Science Teacher, Newbury Park High School

Heather Lageman
Executive Director of Leadership Development, Baltimore County Public Schools

Todd Lash
Doctoral Student/Contributing Member, University of Illinois, CSTA K–8 Task Force

Irene Lee
Researcher, Massachusetts Institute of Technology

Carl Lyman
Specialist over Information Technology Class Cluster, Utah State Board of Education

Daniel Moix
Computer Science Education Specialist, Arkansas School for Mathematics, Sciences & Arts

Dianne O'Grady-Cunniff
Computer Science Teacher, La Plata High School

Anthony A. Owen
Coordinator of Computer Science, Arkansas Department of Education

Minsoo Park
Director of Teaching and Learning, Countryside School

Shaileen Crawford Pokress
Visiting Scholar, Wyss Institute at Harvard; K–12 Curriculum Designer

George Reese
Director of MSTE, MSTE Office at University of Illinois at Urbana Champaign

Hal Speed
Founder, CS4TX

Alfred Thompson
Computer Science Teacher, Bishop Guertin High School

Bryan Twarek
Computer Science Program Administrator, San Francisco Unified School District

A. Nicki Washington
Associate Professor, Computer Science, Winthrop University

David Weintrop
Postdoctoral Researcher, UChicago STEM Ed

Advisors

Alana Aaron, *New York City Department of Education*
Owen Astrachan, *Duke University*
Karen Brennan, *Harvard University*
Josh Caldwell, *Code.org*
Jill Denner, *Education Training Research*
Brian Dorn, *University of Nebraska (Omaha)*
Phillip Eaglin, *ChangeExpectations.org*
Kathi Fisler, *Worcester Polytechnic Institute*
Jeff Forbes, *Duke University*
Joanna Goode, *University of Oregon*
Shuchi Grover, *SRI International*
Mark Guzdial, *Georgia Tech*
Helen Hu, *Westminster College*
Yasmin Kafai, *University of Pennsylvania*
Fred Martin, *University of Massachusetts (Lowell), CSTA board chair-elect*
Don Miller, *New York City Department of Education*
Tammy Pirmann, *CSTA board member, School District of Springfield Township (PA)*
Meg Ray, *Cornell Tech*
Dave Reed, *Creighton University, CSTA board chair*
Deborah Seehorn, *CSTA board past chair, standards co-chair*
Ben Shapiro, *University of Colorado (Boulder)*
Chinma Uche, *Greater Hartford Academy of Math and Science, CSTA board member*
Sheena Vaidyanathan, *Los Altos School District (CA), CSTA board member*
Uri Wilensky, *Northwestern University*
Aman Yadav, *Michigan State University, CSTA board member*

Review

Thank you to the hundreds of individuals and organizations that provided feedback and support during the three public review periods for the framework. The groups that convened reviews are listed in **Appendix A**.

Special Contributions

Thank you to Jennifer Childress of Achieve for her advice and consultation during the development of the framework.

Thank you to Heidi Schweingruber of the Board on Science Education at the National Academies of Science, Engineering, and Medicine and Thomas Keller of the Maine Mathematics and Science Alliance for sharing their experience developing the National Research Council Framework for K–12 Science Education.

In addition to developing the concepts and practices of the framework, the following writers provided significant contributions to guidance chapters: Derek Babb, Leigh Ann DeLyser, Caitlin McMunn Dooley, Maya Israel, Irene Lee, and Shaileen Crawford Pokress. Thank you to Courtney K. Blackwell for contributing to the early childhood education and research chapters.

The following informal advisors provided critical feedback during the framework's development process: Peter Denning, Naval Postgraduate School; Alan Kay, Viewpoints Research Institute; Michael Lach, UChicago STEM Education at University of Chicago; and Chris Stephenson, Google.

Table of Contents

Figures and Tables

Figures

Executive Summary

The influence of computing is felt daily and experienced on a personal, societal, and global level. Computer science, the discipline that makes the use of computers possible, has driven innovation in every industry and field of study, from anthropology to zoology. Computer science is also powering approaches to many of our world's toughest challenges; some examples include decreasing automobile deaths, distributing medical vaccines, and providing platforms for rural villagers to participate in larger economies, among others.

Computer science is powering approaches to many of our world's toughest challenges.

As computing has become an integral part of our world, public demand for computer science education is high. Most parents want their child's school to offer computer science (Google & Gallup, 2015), and most Americans believe computer science is as important to learn as reading, writing, and math (Horizon Media, 2015). Many of today's students will be using computer science in their future careers, not only in science, technology, engineering, and mathematics (STEM) fields but also in non-STEM fields (Change the Equation, 2015).

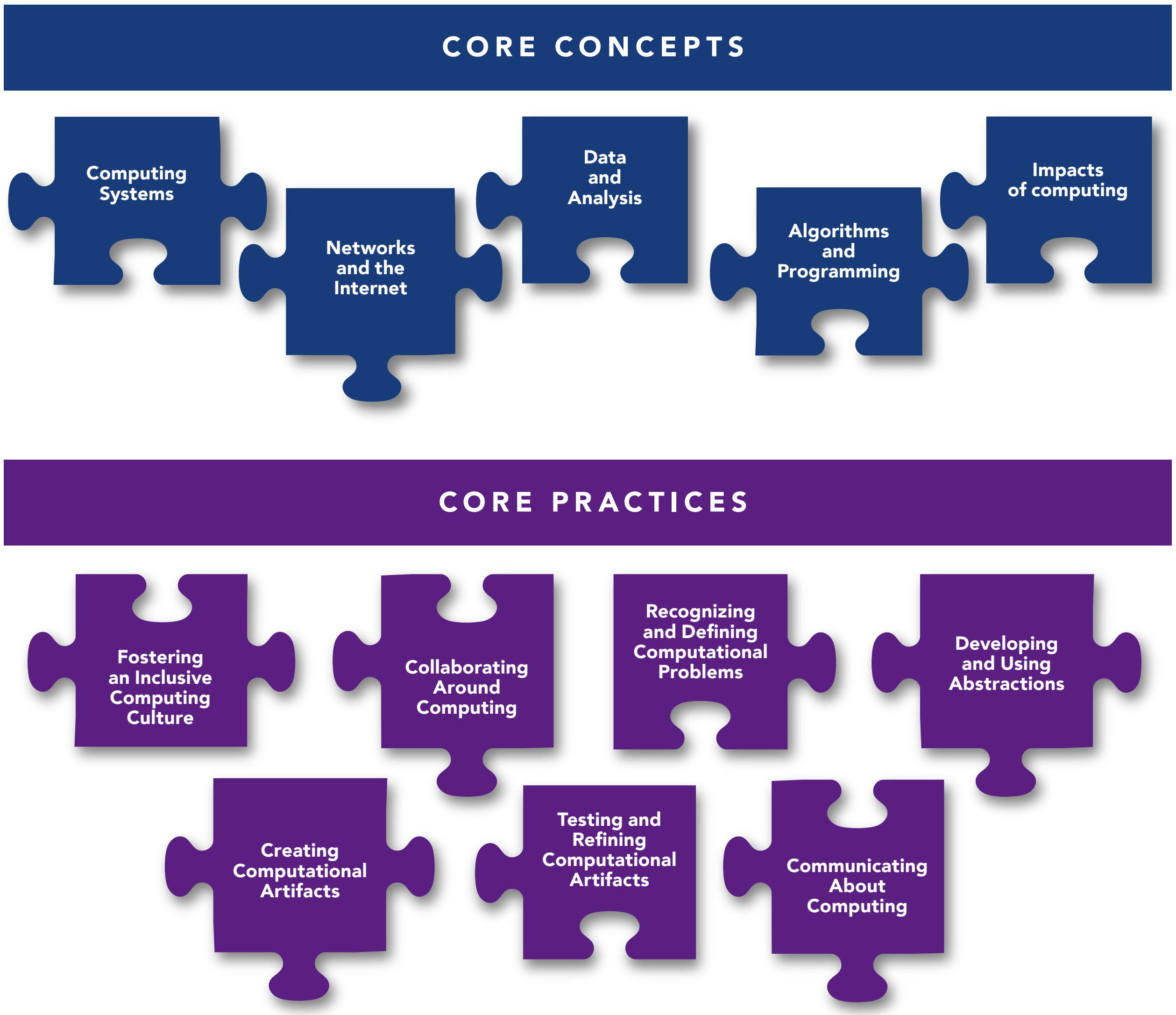
Unfortunately, the opportunity to learn computer science does not match public demand. Most U.S. schools do not offer a single course in computer science and programming (Google & Gallup, 2015), and many existing classes are not diverse and representative of our population (College Board, 2016). Many students have to wait until high school to learn computer science, even though they were born into a society dependent on computing and have never known a world without it. Although computers are increasingly available to students in our nation's schools, opportunities to learn computer science are not accessible by all. State and local education agencies have begun to adopt policies and develop key infrastructure to support computer science for all students and have expressed mutual interest for guidance in this new frontier.

The *K–12* Computer Science Framework *informs standards and curriculum, professional development, and the implementation of computer science pathways.*

The Association for Computing Machinery, Code.org, Computer Science Teachers Association, Cyber Innovation Center, and National Math and Science Initiative have answered the call by organizing states, districts, and the computer science education community to develop conceptual guidelines for computer science education. The *K–12 Computer Science Framework* was developed for states, districts, schools, and organizations to inform the development of standards and curriculum, build capacity for teaching computer science, and implement computer science pathways. The framework promotes a vision in which all students critically engage in computer science issues; approach problems in innovative ways; and create computational artifacts with a practical, personal, or societal intent.

The development of the framework was a community effort. Twenty-seven writers and twenty-five advisors developed the framework with feedback from hundreds of reviewers including teachers, researchers, higher education faculty, industry stakeholders, and informal educators. The group of writers and advisors represents states and districts from across the nation, as well as a variety of academic perspectives and experiences working with diverse student populations.

Figure 0.1: The K–12 Computer Science Framework


CORE CONCEPTS
Computing Systems
Networks and the Internet
Data and Analysis
Algorithms and Programming
Impacts of computing
CORE PRACTICES
Fostering an Inclusive Computing Culture
Collaborating Around Computing
Recognizing and Defining Computational Problems
Developing and Using Abstractions
Creating Computational Artifacts
Testing and Refining Computational Artifacts
Communicating About Computing

The K–12 Computer Science Framework illuminates the big ideas of computer science through a lens of concepts (i.e., what students should know) and practices (i.e., what students should do). The core concepts of the framework represent major content areas in the field of computer science. The core practices represent the behaviors that computationally literate students use to fully engage with the core concepts of computer science. The framework's learning progressions describe how students' conceptual understanding and practice of computer science grow more sophisticated over time. The concepts and practices are designed to be integrated to provide authentic, meaningful experiences for students engaging in computer science (see Figure 0.1).

The framework provides a unifying vision to guide computer science from a subject for the fortunate few to an opportunity for all.

A number of significant themes are interwoven throughout the framework. They include:

- **Equity.** Issues of equity, inclusion, and diversity are addressed in the framework's concepts and practices, in recommendations for standards and curriculum, and in examples of efforts to broaden participation in computer science education.
- **Powerful ideas.** The framework's concepts and practices evoke authentic, powerful ideas that can be used to solve real-world problems and connect understanding across multiple disciplines (Papert, 2000).
- **Computational thinking.** Computational thinking practices such as abstraction, modeling, and decomposition intersect with computer science concepts such as algorithms, automation, and data visualization.
- **Breadth of application.** Computer science is more than coding. It involves physical systems and networks; the collection, storage, and analysis of data; and the impact of computing on society. This broad view of computer science emphasizes the range of applications that computer science has in other fields.

The framework's chapters provide critical guidance to states, districts, and organizations in key areas of interest. Recommendations are provided to guide the development of rigorous and accessible standards for all students. Guidance for designing curriculum, assessment, course pathways, certification, and teacher development programs will inform implementation of the framework's vision. A chapter on computer science in early childhood education describes how computer science can be integrated into the prekindergarten classroom by preserving, supporting, and enhancing the early childhood focus on social-emotional learning and play. The relevant research on which the framework is based, gaps in the K–12 computer science education research literature, and opportunities for further study are described to inform future research and revisions to the framework. An appendix includes a summary of public feedback submitted during the framework's review periods and the subsequent revisions made by writers.

The K–12 Computer Science Framework comes at a time when our nation's education systems are adapting to a 21st century vision of students who are not just computer users but also computationally literate creators who are proficient in the concepts and practices of computer science. As K–12 computer science continues to pick up momentum, states, districts, and organizations can use the framework to develop standards, implement computer science pathways, and structure professional development. The framework provides a unifying vision to guide computer science from a subject for the fortunate few to an opportunity for all.

References

Change the Equation. (2015, December 7). The hidden half [Blog post]. Retrieved from http://changetheequation.org/blog/hidden-half

College Board. (2016). *AP program participation and performance data 2015* [Data file]. Retrieved from https://research.collegeboard.org/programs/ap/data/participation/ap-2015

Google & Gallup. (2015). *Searching for computer science: Access and barriers in U.S. K–12 education.* Retrieved from http://g.co/cseduresearch

Horizon Media. (2015, October 5). Horizon Media study reveals Americans prioritize STEM subjects over the arts; science is "cool," coding is new literacy. *PR Newswire.* Retrieved from http://www.prnewswire.com/news-releases/horizon-media-study-reveals-americans-prioritize-stem-subjects-over-the-arts-science-is-cool-coding-is-new-literacy-300154137.html

Papert, S. (2000). What's the big idea? Toward a pedagogy of idea power. *IBM Systems Journal*, *39* (3/4), 720–729.

A Vision for K–12 Computer Science

1

A Vision for K–12 Computer Science

The K–12 Computer Science Framework represents a vision in which all students engage in the concepts and practices of computer science. Beginning in the earliest grades and continuing through 12th grade, students will develop a foundation of computer science knowledge and learn new approaches to problem solving that harness the power of computational thinking to become both users and creators of computing technology. By applying computer science as a tool for learning and expression in a variety of disciplines and interests, students will actively participate in a world that is increasingly influenced by technology.

The Power of Computer Science

The power of computers stems from their ability to represent our physical reality as a virtual world and their capacity to follow instructions with which to manipulate that world. Ideas, images, and information can be translated into bits of data and processed by computers to create apps, animations, or autonomous cars. The variety of instructions that a computer can follow makes it an engine of innovation that is limited only by our imagination. Remarkably, computers can even follow instructions about instructions in the form of programming languages.

A computer is an engine of innovation that is limited only by our imagination.

Computers are fast, reliable, and powerful machines that allow us to digitally construct, analyze, and communicate our human experience. More than just a **tool,** computers are a readily accessible **medium** for creative and personal expression. In our digital age, computers are both the paint and the paintbrush. Computer science education creates the artists.

Schools have latched on to the promise that computers offer: to deliver instruction, serve as a productivity tool, and connect to an ever-increasing source of information. This belief that computers can improve education is apparent in the number of one-to-one device initiatives seen in our nation's school districts. Despite the availability of computers in schools, the most significant aspect of computing has been held back from most of our students: learning how to create with computers (i.e., computer science).

Literacy provides a relevant context for understanding the need for computer science education. From a young age, students are taught how to read so that they can be influenced by what has been written but also to write so that they can express ideas and influence others. Although computing is a powerful medium like literacy, most students are taught only how to use (i.e., read) the works of computing provided to them, rather than to create (i.e., write) works for themselves. Together, the "authors" who have worked in the computing medium over the last few decades have transformed our society. Learning computer science empowers students to become authors themselves and create their own poems and stories in the form of programs and software. Instead of being passive consumers of computing technologies, they can become active producers and creators. In our digital age, you can either "program or be programmed" (Rushkoff, 2011, p. 1).

In our digital age, computers are both the paint and the paintbrush. Computer science education creates the artists.

A Vision for K–12 Computer Science

From the abacus to today's smartphones, from Ada Lovelace's first computer program to Seymour Papert's powerful ideas, computing has dramatically shifted our world and holds promise to help improve education. Computer science's ways of thinking, problem solving, and creating have become invaluable to all parts of life and are important beyond ensuring that we have enough skilled technology workers. The K–12 Computer Science Framework envisions a future in which students are informed citizens who can

- critically engage in public discussion on computer science topics;
- develop as learners, users, and creators of computer science knowledge and artifacts;
- better understand the role of computing in the world around them; and
- learn, perform, and express themselves in other subjects and interests.

This vision for computer science education is best understood by imagining one of the paths that Maria (a student) could take during her K–12 computer science experience:

- In elementary school, Maria learns how to instruct computers by sequencing actions like puzzle pieces to create computer algorithms that draw beautiful designs. From a young age, she understands that computing is a creative experience and a tool for personal expression.
- In middle school, Maria grows more sophisticated in her use of computing concepts and understanding of how computing works. She uses the computer, as well as computational ideas and processes, to enhance learning experiences in other disciplines. Computing serves as a medium for representing and solving problems.
- In high school, Maria sees opportunities within her community and society for applying computing in novel ways. The concepts and practices of computer science have empowered her to create authentic change on a small and large scale and across a wide variety of interests.

This vision holds promise to enhance the K–12 experience of all students while preparing them for a wide variety of post-secondary experiences and careers. Students who graduate with a K–12 computer science foundation will go on to be computationally literate members of society who are not just consumers of technology but creators of it. They will become doctors, artists, entrepreneurs, scientists, journalists, and software developers who will drive even greater levels of innovation in these and a variety of other fields, benefiting their communities and the world. The K–12 Computer Science Framework is dedicated to making this vision of computer science education accessible to all.

Not just consumers of technology but creators.

The Case for Computer Science

The ubiquity of personal computing and our increasing reliance on technology have changed the fabric of society and day-to-day life. Regardless of their future career, many students will be using computer science at work; by one estimate, more than 7.7 million Americans use computers in complex ways in their jobs, almost half of them in fields that are not directly related to science, technology, engineering, and math (STEM) (Change the Equation, 2015). Unfortunately, K–12 students today have limited opportunity to learn about these computer science concepts and practices and to understand how computer science influences their daily lives.

When fewer than half of schools teach meaningful computer science courses (Google & Gallup, 2015b), the huge disparity in access often marginalizes traditionally underrepresented students, who already face educational inequities. This opportunity gap is reflected in an alarming lack of diversity in the technology workforce (e.g., Information is Beautiful, 2015; Sullivan, 2014). The majority of computer science classes are offered only to high school students, yet research in other STEM fields has repeatedly shown that stereotypes (Scott & Martin, 2014) about who is good at or who belongs in those fields are established from a very young age. Addressing these messages earlier and providing earlier access to computing experiences can help prevent these stereotypes from forming (Google, 2014). Early engagement in computer science also allows students to develop fluency with computer science over many years (Guzdial, Ericson, McKlin, & Engelman, 2012) and gives them opportunities to apply computer science to other subjects and interests as they go through school (Grover, 2014). The lack of opportunity is particularly discouraging, given public opinion and recent job statistics on computer science:

- Americans believe computer science is as important to learn as reading, writing, and math (Horizon Media, 2015).
- Most parents want their child's school to offer computer science (Google & Gallup, 2015b).
- Since 2010, computer science ranks as one of the fastest growing undergraduate majors of all STEM fields (Fisher, 2015), and Advanced Placement (AP®) Computer Science is the fastest growing AP exam, despite being offered in only 5% of schools (Code.org, 2015).
- Jobs that use computer science are some of the highest paying, highest growth (Bureau of Labor Statistics, 2015), and most in-demand jobs that underpin the economy (The Conference Board, 2016).
- Computer science is defined as part of a "well-rounded education" in the Every Student Succeeds Act (2015).

What Is Computer Science?

Computing education in K–12 schools includes computer literacy, educational technology, digital citizenship, information technology, and computer science. 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). The K–12 Computer Science Framework organizes this body of knowledge into five core concepts representing key content areas in computer science and seven practices representing actions that students use to engage with the concepts in rich and meaningful ways.

Computer science is often confused with the everyday use of computers, such as learning how to use the Internet and create digital presentations. Parents, teachers, students, and local and state administrators can share this confusion. A recent survey shows that the majority of students believe that creating documents and presentations (78%) and searching the Internet (57%) are computer science activities (Google & Gallup, 2015a). Parents, teachers, and principals are almost as bad at delineating the difference between traditional computer literacy activities and computer science, and actually more parents than students believe that doing a search on the Internet is computer science (Google & Gallup, 2015a). This confusion extends to state departments of education. A survey of individuals responsible for state certification areas concluded,

> Many states did not seem to have a clear definition or understanding of the field "Computer Science" and exhibited a tendency to confuse Computer Science with other subject areas such as: Technology Education/Educational Technology (TE/ET), Industrial or Instructional Technology (IT), Management Information Systems (MIS), or even the use of computers to support learning in other subject areas. (Khoury, 2007, p. 9)

These misconceptions about computer science pose serious challenges to offering high-quality computer science experiences for all students. The K–12 Computer Science Framework clarifies not only what computer science is but also what students should know and be able to do in computer science from kindergarten to 12th grade. Computer science builds on computer literacy, educational technology, digital citizenship, and information technology. Their differences and relationship with computer science are described below.

- Computer literacy refers to the general use of computers and programs, such as productivity software. Previously mentioned examples include performing an Internet search and creating a digital presentation.
- Educational technology applies computer literacy to school subjects. For example, students in an English class can use a web-based application to collaboratively create, edit, and store an essay online.

- Digital citizenship refers to the appropriate and responsible use of technology, such as choosing an appropriate password and keeping it secure.
- Information technology often overlaps with computer science but is mainly focused on industrial applications of computer science, such as installing software rather than creating it. Information technology professionals often have a background in computer science.

These aspects of computing are distinguished from computer science because they are focused on using computer technologies rather than understanding why they work and how to create those technologies. Knowing why and how computers work (i.e., computer science), provides the basis for a deep understanding of computer use and the relevant rights, responsibilities, and applications.

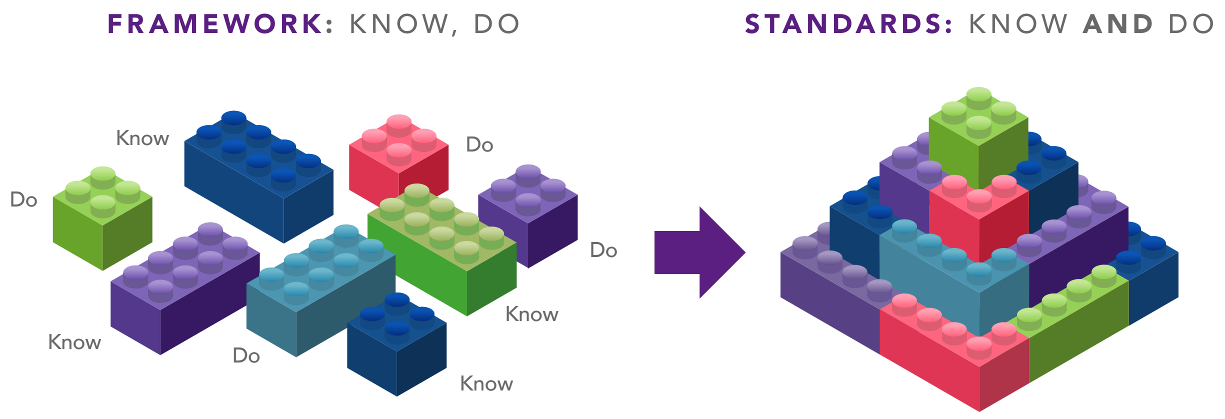
Password security is a topic that illustrates the intersection between computer science and the other aspects of computing. A student who knows how to program a computer to iterate over all of the words in a list (i.e., array) in a split second is a student who will probably not use a dictionary word for a password. In this case, understanding why and how computers work ultimately helps students make good decisions about their use of computers.

Computer science is the foundation for computing. The framework envisions a future in which being computer literate means knowing computer science.

Scope and Intended Audience

The concepts and practices of the K–12 Computer Science Framework are not specific, measurable performance expectations in the form of standards, nor are they detailed lesson plans and activities in the form of curriculum. Instead, the K–12 Computer Science Framework is a high-level guide that states, districts, and organizations can use to inform the development of their own standards and curricula. As illustrated in Figure 1.1, the framework provides building blocks of concepts (that students should know) and practices (that students should do) which can be used to create standards (performance expectations of what students should know and do).

Figure 1.1: Building blocks for standards



It should also be made clear that the framework does not provide the full scope of computer science content for advanced topics of study. The framework describes a baseline literacy for all students, so those who elect to study computer science more deeply may look to honors, AP, or specialized courses in career and technical education programs that include content beyond the framework.

The framework does not prescribe expectations for specific courses. It does not provide grade level-specific outcomes, nor does it define course structure (the scope and sequence of topics in a particular course) or course pathways (the scope of topics and sequence across multiple courses). The five core concepts of the framework were not designed to serve as independent units in a course or separate topics defining entire courses; instead, the framework's concepts and practices are meant to be integrated throughout instruction.

The framework's concepts and practices are meant to be integrated throughout instruction.

The framework was written for an audience with diverse backgrounds, including educators who are learning to teach computer science. This audience includes

- state/district policymakers and administrators;
- standards and curriculum developers (with sufficient computer science experience);
- current and new computer science teachers, including teachers from other subject areas and educators in informal settings; and
- supporting organizations (nonprofits, industry partners, and informal education).

Principles Guiding the Framework

The following principles guided the development of the framework:

1. Broaden participation in computer science.
2. Focus on the essential.
3. Do not reinvent the wheel.
4. Inform with current research and guide future research.
5. Align to nationally recognized frameworks.
6. Inspire implementation.

Broaden Participation in Computer Science

First and foremost, the K–12 Computer Science Framework is designed for all students, regardless of their age, race, gender, disability, or socioeconomic status. The structure and content of the framework reflect the need for diversity in computing and attention to issues of equity, including accessibility. The choice of *Impacts of Computing* as one of the core concepts and *Fostering an Inclusive Computing Culture* as one of the core practices make diversity, equity, and accessibility key topics of study, in addition to interweaving them through the other concepts and practices.

Focus on the Essential

The K–12 Computer Science Framework describes a foundational literacy in computer science, rather than an exhaustive list of all computer science topics that can be learned within a K–12 pathway. Although the framework describes what computer science is essential for all students, educators and curriculum developers are encouraged to create a learning experience that extends beyond the framework to encompass students' many interests, abilities, and aspirations. Additionally, the framework attempts to use jargon-free, plain language that is accessible to instructors and the general public. Where technical terms are used, they are deemed necessary to stay true to disciplinary vocabulary and to fully illustrate the relevant concepts.

Do Not Reinvent the Wheel

The K–12 Computer Science Framework is based on a history of professional research and practice in computer science education. The framework is influenced by the work of professional organizations like the Computer Science Teachers Association (CSTA Standards Task Force, 2011) and frameworks from math, science, and technology education (e.g., ISTE, 2016). Nationally recognized course frameworks like the Advanced Placement Computer Science Principles curriculum framework (College Board, 2016) and the Association for Computing Machinery's curriculum guidelines for undergraduate computer science programs provided a vision for students who may continue to advanced computer science studies. Computer science frameworks from other countries—the United Kingdom (England Department for Education, 2013), Germany (Hubwieser, 2013), Poland (Sysło & Kwiatkowska, 2015), and New Zealand (Bell, Andreae, & Robins, 2014)—were used to benchmark the concepts and practices of the framework.

Inform With Current Research and Guide Future Research

The framework reflects current research in computer science education, including learning progressions, trajectories, and computational thinking. Where specific computer science education research is lacking, the framework relies on the existing knowledge base of the practitioner community and research from other related content areas to guide decisions such as the developmental appropriateness of particular concepts. Remaining questions have guided a research agenda that will inform future revisions to the framework.

Align to Nationally Recognized Frameworks

Developing a framework for computer science education involves both defining a subject new to most schools and relying on established structures and processes used in the development of other education guidelines. Because this framework will exist alongside those from other subjects, the K–12 Computer Science Framework is intentionally structured in a similar way as other frameworks, such as the Framework for K–12 Science Education (NRC, 2012). The use of a lens of concepts and practices to view and describe K–12 computer science provides greater coherence across subject areas. The K–12 Computer Science Framework also mirrored the development process of other community-driven efforts. Transparency and inclusion were emphasized throughout the entire development process via public summaries, monthly updates, forums/webinars, conversations with stakeholders, advisor workshops, community previews, and public review periods. A summary of public feedback and subsequent revisions to the framework can be found in **Appendix A**.

Inspire Implementation

Whether a state or district is already in the process of implementing computer science for all students, or has just begun, the K–12 Computer Science Framework provides a coherent vision for inspiring further efforts. The framework contains chapters that provide guidance on a variety of key implementation steps, such as developing standards, preparing teachers, and creating curriculum that reflects the concepts and practices of the framework. Policy and implementation must go hand in hand to provide high-quality computer science opportunities for all students.

Summary

The goal of this project has been to provide a high-level framework for K–12 computer science education by identifying the core concepts and practices of computer science and describing what those concepts and practices look like for students at various grade bands. The framework provides guidance to states, districts, and organizations that want to design their own standards, curriculum, assessments, or teacher preparation programs. Computer science education is an evolving field with a growing research body at the K–12 level and many lessons to be learned as education systems take steps to increase computer science opportunities. The community that has developed and supported this project believes that the K–12 Computer Science Framework is an initial step to inform, inspire, and drive the implementation work required to make the vision of the framework a reality—computer science for all students.

References

Bell, T., Andreae, P., & Robins, A. (2014). A case study of the introduction of computer science in NZ schools. *ACM Transactions on Computing Education (TOCE), 14*(10), 10:1–10:31. doi: 10.1145/2602485

Bureau of Labor Statistics. (2015). *Employment projections* [Data file]. Retrieved from http://www.bls.gov/emp/tables.htm

Change the Equation. (2015, December 7). The hidden half [Blog post]. Retrieved from http://changetheequation.org/blog/hidden-half

Code.org. (2015, July 2). Computer science is the fastest growing AP course of the 2010s [Blog post]. Retrieved from http://blog.code.org/post/123032125688/apcs-2015

College Board. (2016). *AP Computer Science Principles course and exam description*. New York, NY: College Board. Retrieved from https://secure-media.collegeboard.org/digitalServices/pdf/ap/ap-computer-science-principles-course-and-exam-description.pdf

Computer Science Teachers Association Standards Task Force. (2011). *CSTA K–12 computer science standards, revised 2011*. New York, NY: Computer Science Teachers Association and Association for Computing Machinery. Retrieved from http://www.csteachers.org/resource/resmgr/Docs/Standards/CSTA_K-12_CSS.pdf

The Conference Board. (2016). *National demand rate and OES employment data by occupation* [Data file]. Retrieved from https://www.conference-board.org/

England Department for Education. (2013, September 11). National curriculum in England: Computing programmes of study. Retrieved from https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study/national-curriculum-in-england-computing-programmes-of-study

Every Student Succeeds Act of 2015, Pub. L. No. 114-95. 20 U.S.C.A. 6301 (2016).

Fisher, A. (2015, February 10). The fastest-growing STEM major in the U.S. *Fortune*. Retrieved from http://fortune.com/2015/02/10/college-major-statistics-fastest-growing/

Google. (2014). *Women who choose computer science—what really matters: The critical role of encouragement and exposure*. Mountain View: CA. Retrieved from https://www.google.com/edu/resources/computerscience/research/

Google & Gallup. (2015a). *Images of computer science: Perceptions among students, parents, and educators in the U.S.* Retrieved from http://g.co/cseduresearch

Google & Gallup. (2015b). *Searching for computer science: Access and barriers in U.S. K–12 education.* Retrieved from http://g.co/cseduresearch

Grover, S. (2014). *Foundations for advancing computational thinking: Balanced designs for deeper learning in an online computer science course for middle school students* (Doctoral dissertation). Stanford University, CA.

Guzdial, M., Ericson, B. J., McKlin, T., & Engelman, S. (2012). A statewide survey on computing education pathways and influences: Factors in broadening participation in computing. *Proceedings of the Ninth Annual International Conference on International Computing Education Research* (pp. 143–150). New York, NY. doi: 10.1145/2361276.2361304

Horizon Media. (2015, October 5). Horizon Media study reveals Americans prioritize STEM subjects over the arts; science is "cool," coding is new literacy. *PR Newswire*. Retrieved from http://www.prnewswire.com/news-releases/horizon-media-study-reveals-americans-prioritize-stem-subjects-over-the-arts-science-is-cool-coding-is-new-literacy-300154137.html

Hubwieser, P. (2013). The Darmstadt model: A first step towards a research framework for computer science education in schools. In I. Diethelm & R. T. Mittermeir (Eds.), *Informatics in Schools: Sustainable Informatics Education for Pupils of All Ages. Proceedings of the 6th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives (ISSEP 13), Oldenburg, Germany* (pp. 1–14). doi: 10.1007/978-3-642-36617-8_1

Information is Beautiful. (2015). *Diversity in tech: Employee breakdown of key technology companies.* Retrieved from http://www.informationisbeautiful.net/visualizations/diversity-in-tech/

International Society for Technology in Education. (2016). *ISTE standards for students.* Retrieved from https://www.iste.org/resources/product?id=3879&childProduct=3848

Khoury, G. (2007). *Computer science state certification requirements. CSTA Certification Committee report*. Retrieved from the Computer Science Teachers Association website: https://csta.acm.org/ComputerScienceTeacherCertification/sub/TeachCertRept07New.pdf

National Research Council. (2012). *A framework for K–12 science education: Practices, crosscutting concepts, and core ideas.* Committee on a Conceptual Framework for New K–12 Science Education Standards. Board on Science Education, Division of Behavioral and Social Sciences and Education. Washington, DC: The National Academies Press.

Rushkoff, D. (2010). *Program or be programmed: Ten commands for a digital age.* New York, NY: OR Books.

Scott, A., & Martin, A. (2014). Perceived barriers to higher education in science, technology, engineering, and mathematics. *Journal of Women and Minorities in Science and Engineering, 20*(3), 235–256. doi: 10.1615/JWomenMinorScienEng.2014006999

Sullivan, G. (2014, May 29). Google statistics show Silicon Valley has a diversity problem. *The Washington Post*. Retrieved from https://www.washingtonpost.com/news/morning-mix/wp/2014/05/29/most-google-employees-are-white-men-where-are-allthewomen/

Sysło, M. M., & Kwiatkowska, A. B. (2015). Introducing a new computer science curriculum for all school levels in Poland. In A. Brodnik & J. Vahrenhold (Eds.), *Informatics in Schools: Curricula, Competences, and Competitions. Proceedings of the 8th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives* (ISSEP 2015), Ljubljana, Slovenia (pp. 141–154). doi: 10.1007/978-3-319-25396-1_13

Tucker, A., McCowan, D., Deek, F., Stephenson, C., Jones, J., & Verno, A. (2006). *A model curriculum for K–12 computer science: Report of the ACM K–12 task force curriculum committee* (2nd ed.). New York, NY: Association for Computing Machinery.

Equity in Computer Science Education

2

Equity in Computer Science Education

Computer science for all students requires that equity be at the forefront of any reform effort, whether at the policy level of a framework or at the school level of instruction and classroom culture. When equity exists, there are appropriate supports based on individual students' needs so that all have the opportunity to achieve similar levels of success. Inherent in this goal is a comprehensive expectation of academic success that is accessible by and applies to every student. The purpose of equity in computer science is not to prepare all students to major in computer science and go on to careers in software engineering or technology. Instead, it is about ensuring that all students have the foundational knowledge that will allow them to productively participate in today's world and make informed decisions about their lives. Equity is not just about whether classes are available, but also about how those classes are taught, how students are recruited, and how the classroom culture supports diverse learners and promotes retention. The result of equity is a diverse classroom of students, based on factors such as race, gender, disability, socioeconomic status, and English language proficiency, all of whom have high expectations and feel empowered to learn. Computer science is more than a discipline for a select few; it is an essential 21st century literacy for all students.

Equity is not just about whether classes are available, but also about how those classes are taught, how students are recruited, and how the classroom culture supports diverse learners and promotes retention.

This chapter reviews equity issues related to computer science, describes how these issues have influenced the development of the framework, and offers brief examples of current efforts to promote equity in computer science education. While equity is the focus of this chapter, it also serves as a major theme connecting all aspects of the framework: the core concepts and practices, the guidance chapters, and even the development process. Additional recommendations for promoting equity can be found in the following chapters: **Guidance for Standards Developers, Implementation Guidance,** and **The Role of Research in the Development and Future of the Framework.**

The Need to Address Equity in Computer Science

Although concerns about equity are prevalent in science, technology, engineering, and math (STEM) education, computer science faces intense challenges in terms of access, opportunity, and culture. A study by Google and Gallup (2015b) showed that fewer than half of K–12 schools offer meaningful computer science courses that include programming. An analysis of the 2015 National Assessment of Educational Progress (NAEP) survey showed that only 44% of 12th graders attend high schools that offer any computer science courses (Change the Equation, 2016). The same NAEP data revealed that students with the least access are Native American, Black, and Latino; from lower income backgrounds; and from rural areas.

These surveys represent best-case scenarios as, according to Google and Gallup (2015a), "many students, parents, teachers and school administrators do not properly distinguish between computer science activities and general computer literacy" (p. 3). It is possible that many respondents are confusing computer science with computer literacy, decreasing the reported percentage of actual computer science courses. Across the nation, there is no common definition of computer science, and it is often conflated with computer literacy activities, such as searching the Internet and creating documents or presentations on the computer (Google & Gallup, 2015a). Students may not know any better when they get placed into so-called computer courses that have more to do with learning how to type than learning computer science (Margolis, Estrella, Goode, Holme, & Nao, 2010). According to Margolis et al. (2012), "Especially in schools with high numbers of African American and Latino/a students, computer classes too commonly offer only basic, rudimentary user skills rather than engaging students with the problem-solving and computational thinking practices that are the foundation of computer science" (p. 73).

Even when computer science courses are available, there are wide gaps in participation. For the 2015 Advanced Placement (AP®) Computer Science exam, only 21.9% of students were female, the worst female participation rate of all the AP exams (College Board, 2016). Only 3.9% were Black or African American, 9% were Hispanic or Latino, and 0.4% were American Indian.[1] Schools often shoulder the blame for inequities in education, but stereotypes and media representations of computer scientists can play a large factor in how students feel about the subject. Parents and students report that the people who do computer science in film or television are overwhelmingly male, White or Asian, and wearing glasses (Google & Gallup, 2015a). Females report that they are less likely to learn computer science, have less confidence in learning it, and are less likely to need to know computer science in their future career (Google & Gallup, 2015a).

The lack of access and participation at the K–12 level has a clear effect on representation in computer science after K–12. In 2015, only 24.7% of those employed in computer and mathematical occupations were female, 8.6% Black or African American, and 6.8% Hispanic or Latino (Bureau of Labor Statistics, 2015b). When looking specifically at students graduating with a bachelor's degree in computer science, only 17% of students are female, 8% are Black, and 9% are Hispanic (NCES, 2014). The proportion of women with bachelor's degrees in computer science has actually declined from 1993 to 2012 (NSF, 2015). Fortunately, there are positive signs that learning computer science in high school is correlated with a greater likelihood of pursuing computer science in postsecondary study. Students who take AP Computer Science in high school are six times more likely to major in computer science than students who did not take AP Computer Science in high school. Females are 10 times more likely, African American students are 7 times more likely, and Hispanic students are 8.5 times more likely (Morgan & Klaric, 2007).

Females who take AP Computer Science are 10 times more likely to major in computer science.

1 Compare this to the overall participation rate of underrepresented minorities. In 2013, for example, 9.2% of AP exam takers were Black or African American, 18.8% Hispanic or Latino, and 0.6% American Indian (College Board, 2014). *Advanced Placement® is a trademark registered and/or owned by the College Board, which was not involved in the production of, and does not endorse, this product.*

Public opinion and the role of computer science in our economy only highlight the disparity between access and the benefits of a computer science education. Most Americans believe computer science is as important to learn as reading, writing, and mathematics (Horizon Media, 2015), and most parents want their child's school to offer computer science (Google & Gallup, 2015b). More than 90% of students and parents agree that people who work in computer science have the opportunity to work on fun and exciting projects and make things that help improve people's lives (Google & Gallup, 2015a). Additionally, jobs that use computer science are some of the highest paying (Bureau of Labor Statistics, 2015c; Burning Glass Technologies, 2016), highest growth (Bureau of Labor Statistics, 2015a), and most in-demand jobs (The Conference Board, 2016) that underpin the economy.

People who work in computer science have the opportunity to work on fun and exciting projects and make things that help improve people's lives.

Equity in the Framework

Computer science is a discipline in service of society, its people, and their needs. As such, equity, inclusion, and diversity are critical factors in all aspects of computer science. When setting up a team, understanding the benefits of inclusion and diversity motivates students to actively seek out collaborators with different perspectives and backgrounds. When designing a smartphone app, a concern about equity motivates teams to think about how to design the user interface for those with visual impairments. These are just some of the ideas illustrated in the concepts and practices of the framework.

The goal of promoting diversity shaped the framework's entire development process and began with the makeup of the team of people developing the framework. The writers and advisors were demographically diverse based on gender, race, ethnicity, institutional representation (state/district agency, nonprofit, research, industry, K–12 school), as well as the populations with which they work or study. The writers and advisors represented a full range of educational experiences, from elementary school to higher education; informal and formal education settings; private and public schools; and rural, urban, and suburban locations. For example, multiple writers and advisors had experience working with Native American students on reservations. While the vast majority of the writers taught or were currently teaching computer science, they also had expertise in a variety of subjects, including other STEM subjects, as well as humanities such as language arts and social studies. The

The writing team included teachers whose professional focus was students with cognitive and physical disabilities and students at risk of academic failure.

writing team included special education researchers and teachers whose professional focus was students with cognitive and physical disabilities and students at risk of academic failure. Read the writers' biographies in **Appendix B**.

Multiple elements relating to the framework's vision, structure, and content emphasize that computer science is a discipline in which all students can engage and demonstrate proficiency. The following sections describe the ways in which equity was considered when determining the framework's essential ideas, breadth of the concepts, learning progressions in the concepts, and computer science practices.

Essential Ideas

The framework describes a comprehensive and essential foundation in computer science that all students can benefit from, regardless of whether or not they will go on to postsecondary education in computer science or to a career in computer science. The computer science described in the framework is not just for "gifted" or "honors" students but all students, including those who can and should advance past the expectations in the framework. The significance and importance of each concept and practice was evaluated by writers, advisors, and reviewers, with constant consideration of a diverse student population. Only ideas deemed essential were incorporated into the framework's concepts and practices. In addition, the descriptions of concepts and practices were analyzed to make sure the language was not biased based on gender, culture, or disability.

Broad Concepts

Rather than prescriptive, narrow ideas in computer science, the framework statements are conceptual and high-level. The framework does not specify the amount of instruction time for each concept or the order in which the concepts should be addressed. Instead, the conceptual nature of the framework allows for broad implementation possibilities, including in some cases, integration and application within other topics and subjects. Schools that may have trouble fitting an additional subject into the school day can integrate the framework's concepts into current course offerings. This integration is especially beneficial for students who deserve experiences in computer science yet require instructional time in traditional core subjects. A graphic describing the intersection between practices in mathematics, science, and computer science is available in the Practices chapter of the framework (see Figure 5.2).

Complete Learning Progressions

The core concepts in the framework are divided into more specific subconcepts, which provide focal points for the development of complete learning progressions from kindergarten to 12th grade. This attention to coherence and articulation between grade bands means that the framework reflects all of the key stages in a learning progression. Incomplete learning progressions require additional, out-of-school opportunities to fill gaps in knowledge, putting students without these experiences at a disadvantage.

Computer Science Practices

The computer science practices focus on *doing* computer science, in addition to *knowing* it, and provide a variety of opportunities for many different types of learners to participate on a level playing field with other students. This is especially true for English language learners who may otherwise struggle with demonstrating conceptual knowledge in traditional ways due to their limited English ability, as they can demonstrate their understanding in other ways. Additionally, the use of the practices also facilitates rich learning opportunities that can reinforce language acquisition. For example, practices such as *Communicating About Computing* describe the expectation that all students exchange ideas with diverse audiences in a variety of ways. In the process of *Collaborating Around Computing*, students solicit and provide feedback from team members. These interactions allow English language learners to build their language proficiency while engaging in authentic computer science activities.

Examples of Equity in Concept and Practice Statements

The next sections include specific examples of concepts and practices that address equity, encourage diversity, and foster inclusion. These are only a few examples of the ways that equity, diversity, and inclusion are addressed in the concept and practice statements; there are many others.

Examples: Weaving Equity Into Concepts

Equity is woven throughout the concept statements. A few examples are discussed here.

Equity is particularly apparent in the *Impacts of Computing* core concept, as the overview states, "An informed and responsible person should understand the social implications of the digital world, including equity and access to computing." Specific concept statements address equity directly. In Grades 9–12, students should understand that the "design and use of computing technologies and artifacts can improve, worsen, or maintain inequitable access to information and opportunities" (9–12. Impacts of Computing.Culture). The description for this concept statement elaborates on issues of equity and access:

> While many people have direct access to computing throughout their day, many others are still underserved or simply do not have access. Some of these challenges are related to the design of computing technologies, as in the case of technologies that are difficult for senior citizens and people with physical disabilities to use. Other equity deficits, such as minimal exposure to computing, access to education, and training opportunities, are related to larger, systemic problems in society. (9–12.Impacts of Computing.Culture)

Ties to equity, inclusion, and diversity are also found in other concept areas. For example, in Grades 6–8 under the core concept of *Algorithms and Programming*, students should understand that designing solutions involves "carefully considering the diverse needs and wants of the community"

(6–8.Algorithms and Programming.Program Development). The description provides an example of how a team that employs user-centered design may develop an app that translates hard-to-understand pronunciation from people with speech difficulties into understandable language. In Grades 9–12, students learn that "diverse teams can develop programs with a broad impact through careful review and by drawing on the strengths of members in different roles" (9–12.Algorithms and Programming.Program Development).

Examples: Weaving Equity Into Practices

Considerations of equity are included in many of the practices, but it is the main focus of the first core practice.

The practice *Fostering an Inclusive Computing Culture* is dedicated to equity, inclusion, and diversity. The associated practice statements are below:

By the end of Grade 12, students should be able to:

1. **Include the unique perspectives of others** and reflect on one's own perspectives when designing and developing computational products.
2. **Address the needs of diverse end users** during the design process to produce artifacts with broad accessibility and usability.
3. **Employ self- and peer-advocacy** to address bias in interactions, product design, and development methods.

These practice statements emphasize the role that equity, inclusion, and diversity each play in the design of computational artifacts. The *Collaborating Around Computing* practice also describes ways of promoting inclusion in computer science. The first practice statement reads, "Cultivate working relationships with individuals possessing diverse perspectives, skills, and personalities." To fulfill this recommendation, students learn strategies for including all team members' ideas, such as trying to draw out the opinions of the quieter collaborators. Other instances in the practice overviews, statements, and progressions attend to concerns about equity, diversity, and inclusion. These instances emphasize the need to practice equity and inclusion when doing computer science, such as creating an app or robot, to benefit from diverse collaborators and attention to diverse users.

Efforts to Increase Access and Opportunity

Computer science education research has described how inequity can be perpetuated on a small scale, as in a classroom, as well as on a large scale, as in course availability and scheduling. Differences in learning opportunities are often seen along the lines of gender, race, disability, and socioeconomic status and are reflected in reduced access to resources, such as ample computer labs or adequately trained teachers (Ladner & Israel, 2016; Margolis et al., 2010). The role of school structures and educators' belief systems in limiting access, recruitment, and retention of African American and Latino students in Los Angeles are well documented by Margolis and associates (2010) in *Stuck in the Shallow End*. In an article in *ACM Inroads* (2012), Margolis et al. explain, "Course offerings (or lack thereof), placement outside of the academic core curriculum, lack of teacher preparation and instructional resources, as well as the educational climate of school accountability required by state and federal legislation, all result in wide disparities and the lack of availability and quality of CS education opportunities for students of color" (p. 73). Inequities can also be propagated when programs are scaled up to meet national needs and demands for computer science, unless recruitment, expansion, and equity are actively monitored during the scaling process (Margolis, Goode, & Chapman, 2015).

Equity issues also arise at the level of student interactions within the classroom. Research confirms that pair programming, a type of collaborative structure when students program together while rotating through defined roles, can be beneficial for building computational thinking and developing programming knowledge (Denner, Werner, Campe, & Ortiz, 2014). Yet in situations of pair programming, pair compatibility and the focus of the work can lead to inequity. Pairs of students with more incompatibilities result in more course withdrawals and lower retention in beginning programming courses (Watkins & Watkins, 2009). Student pairs whose programming focuses on speed of completion, rather than quality and collaboration, often show more inequitable interactions (Lewis & Shah, 2015). Differences in equity can also appear in other student interactions outside of programming, such as those based on students' access to productive peer relationships (Shah et al., 2013) or technology-oriented peer groups (Goode, Estrella, & Margolis, 2006).

As computer science instruction moves into the mainstream classroom, programming environments and curricula need to be differentiated for a more diverse student population. Recent research has explored the use of Universal Design for Learning (UDL) to develop and refine introductory computer science experiences for a wide range of learners (Hansen, Hansen, Dwyer, Harlow & Franklin, 2016). The learning accommodations and curricular modifications demonstrate that established techniques for differentiation instruction can be readily applied in computer science to engage all students.

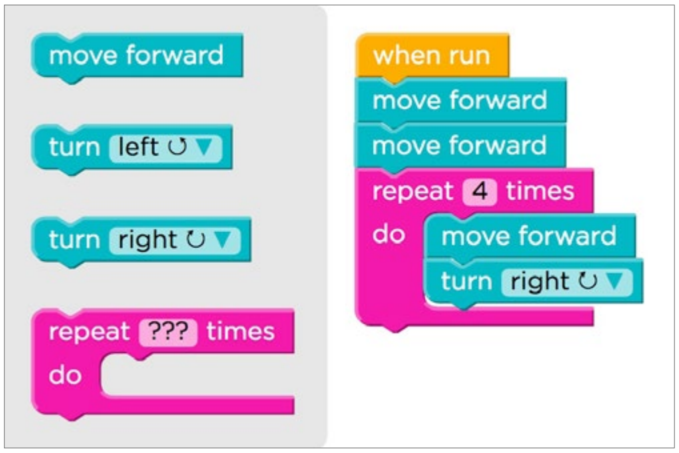
Established techniques for differentiation instruction can be readily applied in computer science to engage all students

The following sections briefly describe efforts to increase access to computer science and change curriculum, instruction, and classroom culture. Although they are not comprehensive, the following examples illustrate key, practical approaches that educators can use to increase equity in computer science. Each example demonstrates the importance of providing both access and support to present realistic opportunities for students to learn computer science.

Examples: Reaching Young Students and Beginners

A variety of approaches make programming more accessible to young learners and beginners. Visual, block-based programming languages designed for education (see Figure 2.1) allow students to program without the obstacle of syntax errors (errors in typing commands) found in traditional text-based languages.

Figure 2.1: Example of block-based programming language



These languages and environments have been designed for both young students and beginning programmers, but they also allow for students to build complex programs, games, apps, and animations. The social communities that have evolved around them allow students to support each other's development by sharing, reusing, and remixing others' creations (Brennan & Resnick, 2012; Kafai & Burke, 2014). Programming environments on tablets for kids as young as 5 years old have made programming even more accessible to younger children by reducing the number of available commands and the amount of reading required to navigate the options (Strawhacker & Bers, 2014). A robotics environment created for prekindergarten students uses physical wooden blocks to create sets of commands that can be read by the robot and executed (Elkin, Sullivan, & Bers, 2014). Games and apps that teach programming skills are even available on smartphones.

School districts around the country have transformed their elementary computer classes to focus on computer science (e.g., Lincoln Public Schools, 2016; New York City Department of Education, 2016; San Francisco Unified School District, 2016). Beginning in elementary school, students use visual, block-based programming languages and learn about variables, loops, conditional statements, functions, events, and more in the context of making projects they find fun and engaging. As they transition to older grades, they apply their skills to create projects, stories, games, apps, program robots, and more. There are nationwide curriculum and professional development efforts focused on helping elementary school teachers learn and integrate computer science into students' classroom experience (e.g., Code.org, 2016; Project Lead the Way, 2016).

Access to high-quality computer science continues to be a problem in middle school. In the transition from elementary school, classes become subject-specific and computer science is often out of place. To combat this issue, groups have created computer science curricula focused on the developmental and intellectual ability level of middle school students that can be integrated into existing courses such as mathematics, science, English, and social studies. University programs are working with middle school teachers across a variety of subjects to create lessons that use the principles of computer science to deliver their content (e.g., University of Nebraska at Omaha, 2016). The approach of teaching other subjects through the lens of computer science is exemplified by efforts that teach algebra concepts by programming video games (Bootstrap, 2016) and science through modeling and simulation (Project GUTS, 2016).

While a lack of computers and Internet access continues to be a disadvantage for reaching all students, many computer science topics, such as algorithmic thinking, searching, sorting, and logic, can be learned without computers. "Unplugged" computer science is an approach to learning computer science concepts through physical, kinesthetic experiences and can be taught independent of computer or online access (CS Unplugged, 2016). Students go on a treasure hunt to understand finite state automata, do magic tricks to learn how computers detect errors, and pass fruit around to learn about network routing and deadlock. Teachers can combine these unplugged experiences with programming exercises to provide even richer experiences for young students.

Examples: Reaching Students With Disabilities

Some research and implementation groups are focusing their work on ensuring meaningful access to computer science for students with disabilities. This work is of critical importance as 13% of students in U.S. public schools receive special education services under the Individuals with Disabilities Education Act (NCES, 2016) and another significant portion of students receive accommodations under the Americans with Disabilities Act. Because schools are legally required to meet the needs of these learners, it is critical that computer science education focus on this population of learners as well. Three recent efforts have been under way to reach students with disabilities. The Quorum programming language, developed at the University of Nevada, Las Vegas, is able to be read by existing computer screen readers, making it accessible to students with visual impairments (Quorum, 2016). In addition to accommodating students with sensory disabilities such as visual impairments, researchers at the Creative Technologies Research Lab (CTRL-Shift) at the University of Illinois at Urbana Champaign are developing and researching teaching strategies focused on increasing access to computing for students with cognitive disabilities (Creative Technologies Research Lab, 2016). Researchers in the ACCESS CS10K group at the University of Washington have developed curriculum resources and provide professional development for teachers of students with disabilities (AccessCS10K, 2016).

Examples: Reaching Females and Underrepresented Minorities

Many programs and strategies are designed to engage students from groups that are underrepresented in computer science. Exploring Computer Science (ECS, 2016), an introductory high school-level course, and AP Computer Science Principles, an introductory college-level course, are two of the most recent and prominent in-school courses. Their content and attention to equity influenced the concepts and practices of the framework. Teachers of these and other courses employ a number of strategies to combat issues, such as stereotype threat, bias, and fixed mindsets, that endanger equity in the classroom. Together, these programs and strategies exemplify an approach to changing curriculum, instruction, and classroom culture to broaden participation, especially among females and underrepresented minorities.

ECS is designed to engage high school students in computational practices. Projects and instruction are based in inquiry and equity and designed to be socially relevant and meaningful for diverse students. Margolis et. al (2012) describe the culturally relevant approach in ECS lessons:

> These types of lessons are one way to help students build personal relationships with CS concepts and applications—an important process for discovering the relevance of CS for their own lives. For example, students can learn fractal geometry through the "African Fractals" program or "[Cornrow] Braiding" program, but they can also learn physics and ways to program slopes and arcs through the "Skateboarding" program. We recognize that students are all different, culture is multi-layered, and while some students may be interested in their ancestors' cultures, others may be interested in the culture of hip hop, graphic design, skateboarding, medicine, and an endless list of different practices. (p. 76)

The accompanying professional development program is based on three major pillars: computer science content/concepts, inquiry, and equity. Although some teachers may have trouble seeing inequity in their classroom (Hu, Heiner, & McCarthy, 2016), the professional development sessions are aimed at addressing teacher belief systems and can open up thoughts about deficit thinking and preparatory privilege (Margolis, Goode, Chapman, & Ryoo, 2014). Teachers are trained to leverage students' cultural knowledge and bring it into the learning experience (Goode, 2008; Eglash, Gilbert, & Foster, 2013).

The AP Computer Science Principles course highlights seven big ideas and six computational practices in an effort to appeal to a wide student audience who may be interested in computer science beyond the traditional programming-centric experience currently in K–12 schools. Students learn about a range of topics, from how computing extends human expression to how technology affects the world. The AP Computer Science Principles course differs from ECS in a few ways. First, despite serving as an introductory course, AP Computer Science Principles is considered college-level coursework and may not be appropriate as the only computer science offering in a high school. Second, unlike ECS, there is not one singular curriculum; teachers and content providers create lesson plans and resources based on the provided course framework. Finally, students may qualify for college-level credit or placement by passing an end-of-year exam composed of multiple-choice questions as well as performance tasks that are completed as through-course assessments during the year. These performance tasks allow students to meet measurable assessment objectives by completing some tasks that allow for personal choice in choosing topics as well as encouraging collaboration with peers.

A variety of actions for changing classroom culture require a community to address. Well-researched obstacles to equity in other subjects include stereotype threat (Steele, Spencer, & Aronson, 2002), implicit (Greenwald & Krieger, 2006) or unconscious biases (Pollard-Sacks, 1999), and fixed mindsets (Dweck, 2000); their influence applies to computer science as well (Cutts, Cutts, Draper, O'Donnell, & Saffrey, 2010; Kumar, 2012; Simon et al., 2008). For example, if tutors teach students about mindsets and give them growth mindset messages on their work, test scores could improve, and stereotype threat could be mitigated (Cutts et al., 2010). Stereotype threat can be mitigated by altering the wording of test questions to be gender-neutral and using examples that are equally relevant to both males and females (Kumar, 2012). It is also important for students to have diverse role models in the field so that they can imagine themselves as a computer scientist, as well as to dispel stereotypes of what computer scientists look and act like (Goode, 2008).

Other practices that teachers can adopt and adapt to change classroom culture include

- practicing culturally relevant pedagogy that brings computer science together with students' experiences, culture, and interests (Margolis et al., 2014);
- developing relationships with students that are respectful of different backgrounds and empathetic to different needs and interests (Margolis et al., 2014);
- reflecting on beliefs and actions to address stereotypes among students and teachers alike (Margolis et al., 2014);
- applying instructional strategies that support struggling learners and those with disabilities in other content areas within computer science education (e.g., if verbal prompting helps in math instruction, it will likely help in computer science instruction as well) (Snodgrass, Israel, & Reese, 2016); and
- connecting computer science to concepts that motivate children, like fairness and social justice (Denner, Martinez, Thiry, & Adams, 2015).

Summary

A focus on equity, inclusion, and diversity in all aspects of computer science education will ensure that implementation efforts remain true to the framework's vision of computer science for all students. The structure and content of the framework reflect an attention to equity issues and provide a solid foundation for reforming computer science education. There are many current efforts to promote computer science to diverse populations, including young students, novices, students with disabilities, females, and underrepresented minorities.

A focus on equity will ensure that efforts remain true to the vision of computer science for all students.

References

AccessCS10K. (2016). *Including students with disabilities in computing education for the 21st century.* Retrieved from http://www.washington.edu/accesscomputing/accesscs10k

Bootstrap. (2016). *From intro CS & Algebra to high-level courses in computer science, for all students*. Retrieved from http://www.bootstrapworld.org/

Brennan, K., & Resnick, M. (2012). *New frameworks for studying and assessing the development of computational thinking.* Paper session presented at the annual meeting of the American Educational Research Association, Vancouver, Canada.

Bureau of Labor Statistics. (2015a). *Employment projections* [Data file]. Retrieved from http://www.bls.gov/emp/tables.htm

Bureau of Labor Statistics. (2015b). *Labor force statistics from the Current Population Survey* [Data file]. Retrieved from http://www.bls.gov/cps

Bureau of Labor Statistics. (2015c). *Occupational employment statistics* [Data file]. Retrieved from http://www.bls.gov/oes

Burning Glass Technologies. (2016). *Beyond point and click: The expanding demand for coding skills.* Retrieved from http://burning-glass.com/research/coding-skills/

Change the Equation. (2016, August 9). New data: Bridging the computer science access gap [Blog post]. Retrieved from http://changetheequation.org/blog/new-data-bridging-computer-science-access-gap-0

Code.org. (2016). *Computer Science Fundamentals* for elementary school. Retrieved from https://code.org/educate/curriculum/elementary-school

College Board. (2014). *The tenth annual AP report to the nation*. Retrieved from http://media.collegeboard.com/digitalServices/pdf/ap/rtn/10th-annual/10th-annual-ap-report-to-the-nation-single-page.pdf

College Board. (2016). *AP program participation and performance data 2015* [Data file]. Retrieved from https://research.collegeboard.org/programs/ap/data/participation/ap-2015

The Conference Board. (2016). *National demand rate and OES employment data by occupation* [Data file]. Retrieved from https://www.conference-board.org/

Creative Technology Research Lab. (2016). *CTRL-Shift: Shifting education*. Retrieved from http://ctrlshift.mste.illinois.edu/home/

CS Unplugged. (2016). Retrieved from http://csunplugged.org/

Cutts, Q., Cutts, E., Draper, S., O'Donnell, P., & Saffrey, P. (2010, March). Manipulating mindset to positively influence introductory programming performance. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (pp. 431–435). doi: 10.1145/1734263.1734409

Denner, J., Martinez, J., Thiry, H., & Adams, J. (2015). Computer science and fairness: Integrating a social justice perspective into an after school program. *Science Education and Civic Engagement: An International Journal, 6*(2): 41–54.

Denner, J., Werner, L., Campe, S., & Ortiz, E. (2014). Pair programming: Under what conditions is it advantageous for middle school students? *Journal of Research on Technology in Education, 46*(3), 277–296.

Dweck, C. S. (2000). *Self-theories: Their role in motivation, personality, and development.* Philadelphia, PA: Psychology Press.

Eglash, R., Gilbert, J. E., & Foster, E. (2013). Toward culturally responsive computing education. *Communications of the ACM, 56*(7), 33–36.

Elkin, M., Sullivan, A., & Bers, M. U. (2014). Implementing a robotics curriculum in an early childhood Montessori classroom. *Journal of Information Technology Education: Innovations in Practice, 13,* 153–169.

Exploring Computer Science. (2016). Retrieved from http://www.exploringcs.org/

Goode, J. (2008, March). Increasing diversity in K–12 computer science: Strategies from the field. *ACM SIGCSE Bulletin, 40*(1), 362–366.

Goode, J., Estrella, R., & Margolis, J. (2006). Lost in translation: Gender and high school computer science. In W. Aspray & J. M. Cohoon (Eds.), *Women and Information Technology: Research on Underrepresentation* (pp. 89–113). Cambridge, MA: MIT Press.

Google & Gallup. (2015a). *Images of computer science: Perceptions among students, parents, and educators in the U.S.* Retrieved from http://g.co/cseduresearch

Google & Gallup. (2015b). *Searching for computer science: Access and barriers in U.S. K–12 education.* Retrieved from http://g.co/cseduresearch

Greenwald, A. G., & Krieger, L. H. (2006). Implicit bias: Scientific foundations. *California Law Review, 94*(4), 945–967.

Hansen, A., Hansen, E., Dwyer, H., Harlow, D., & Franklin, D. (2016). Differentiating for diversity: Using universal design for learning in computer science education. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (pp. 376–381).

Horizon Media. (2015, October 5). Horizon Media study reveals Americans prioritize STEM subjects over the arts; science is "cool," coding is new literacy. *PR Newswire*. Retrieved from http://www.prnewswire.com/news-releases/horizon-media-study-reveals-americans-prioritize-stem-subjects-over-the-arts-science-is-cool-coding-is-new-literacy-300154137.html

Hu, H. H., Heiner, C., & McCarthy, J. (2016, February). Deploying Exploring Computer Science statewide. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (pp. 72–77).

Kafai, Y. B., & Burke, Q. (2014). *Connected code: Why children need to learn programming*. Cambridge, MA: The MIT Press.

Kumar, A. N. (2012, July). A study of stereotype threat in computer science. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education* (pp. 273–278).

Ladner, R., & Israel, M. (2016). "For all" in "computer science for all." *Communications of the ACM, 59*(9), 26–28.

Lewis, C. M., & Shah, N. (2015, July). How equity and inequity can emerge in pair programming. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research* (pp. 41–50).

Lincoln Public Schools. (2016). *Computer science.* Retrieved from http://home.lps.org/computerscience/

Margolis, J., Estrella, R., Goode, J., Holme, J. J., & Nao, K. (2010). *Stuck in the shallow end: Education, race, and computing.* Cambridge, MA: MIT Press.

Margolis, J., Goode, J., & Chapman, G. (2015). An equity lens for scaling: A critical juncture for exploring computer science. *ACM Inroads, 6*(3), 58–66.

Margolis, J., Goode, J., Chapman, G., & Ryoo, J. J. (2014). That classroom 'magic.' *Communications of the ACM*, 57(7), 31–33.

Margolis, J., Ryoo, J., Sandoval, C., Lee, C., Goode, J., & Chapman, G. (2012). Beyond access: Broadening participation in high school computer science. *ACM Inroads, 3*(4), 72–78.

Morgan, R., & Klaric, J. (2007). *AP students in college: An analysis of five-year academic careers. Research report no. 2007-4.* Retrieved from the College Board website: http://research.collegeboard.org/sites/default/files/publications/2012/7/researchreport-2007-4-ap-students-college-analysis-five-year-academic-careers.pdf

National Center for Education Statistics. (2014). *Integrated postsecondary education data system* [Data file]. Retrieved from http://nces.ed.gov/ipeds/

National Center for Education Statistics. (2016). *Children and youth with disabilities.* Retrieved from http://nces.ed.gov/programs/coe/indicator_cgg.asp

National Science Foundation. (2015). *Women, minorities, and persons with disabilities in science and engineering.* Arlington, VA: Author.

New York City Department of Education. (2016). *Software engineering program (SEP) Jr.* Retrieved from http://sepnyc.org/sepjr/

Pollard-Sacks, D. (1999). Unconscious bias and self-critical analysis: The case for a qualified evidentiary equal employment opportunity privilege. *Washington Law Review, 74,* 913–1032.

Project GUTS: Growing Up Thinking Scientifically. (2016). Retrieved from http://www.projectguts.org/.

Project Lead The Way. (2016). *PLTW launch (K–5).* Retrieved from https://www.pltw.org/our-programs/pltw-launch.

Quorum. (2016). *About Quorum.* Retrieved from https://www.quorumlanguage.com/about.php

San Francisco Unified School District. (2016). *Computer science for all students in SF.* http://www.csinsf.org/

Shah, N., Lewis, C. M., Caires, R., Khan, N., Qureshi, A., Ehsanipour, D., & Gupta, N. (2013, March). Building equitable computer science classrooms: Elements of a teaching approach. In *Proceedings of the 44th ACM Technical Symposium on Computer Science Education* (pp. 263–268).

Simon, B., Hanks, B., Murphy, L., Fitzgerald, S., McCauley, R., Thomas, L., & Zander, C. (2008, September). Saying isn't necessarily believing: Influencing self-theories in computing. In *Proceedings of the Fourth International Workshop on Computing Education Research* (pp. 173–184).

Snodgrass, M. R., Israel, M., & Reese, G. (2016). Instructional supports for students with disabilities in K–5 computing: Findings from a cross-case analysis. *Computers & Education, 100,* 1–17.

Steele, C. M., Spencer, S. J., & Aronson, J. (2002). Contending with group image: The psychology of stereotype and social identity threat. *Advances in Experimental Social Psychology, 34,* 379–440.

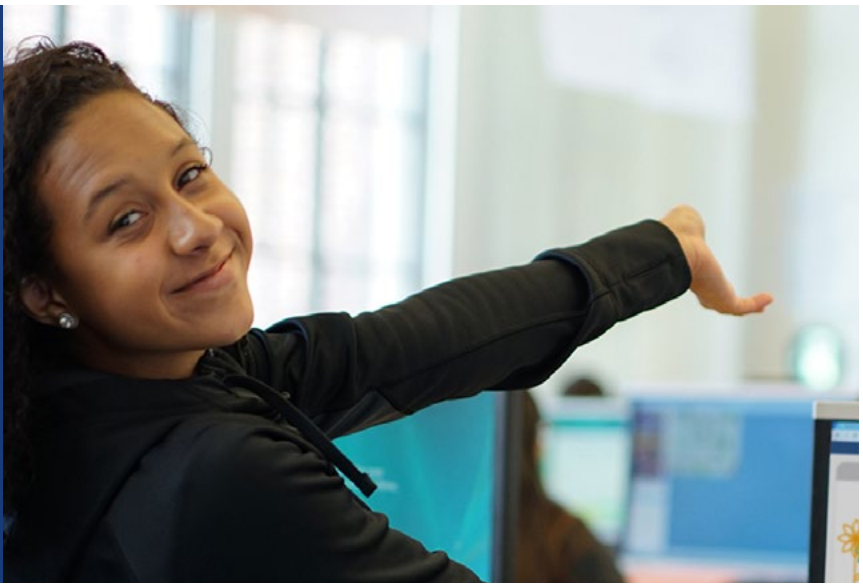
Strawhacker, A. L., & Bers, M. U. (2014, August). *ScratchJr: Computer programming in early childhood education as a pathway to academic readiness and success.* Poster presented at DR K–12 PI Meeting, Washington, DC.

University of Nebraska at Omaha. (2016). *ITEST strategic problem-based approach to rouse computer science (SPARCS).* Retrieved from http://www.unomaha.edu/college-of-information-science-and-technology/itest/index.php

Watkins, K. Z., & Watkins, M. J. (2009). Towards minimizing pair incompatibilities to help retain under-represented groups in beginning programming courses using pair programming. *Journal of Computing Sciences in Colleges, 25*(2), 221–227.

Development Process

3



Development Process

The development of the K–12 Computer Science Framework brought together educators, practitioners, researchers, organizations, and state- and district-level stakeholders to delineate concepts and practices in computer science for all students from kindergarten to 12th grade in the United States. This project was guided by a steering committee with representation from the Association for Computing Machinery (ACM), Code.org, Computer Science Teachers Association (CSTA), Cyber Innovation Center (CIC), and the National Math and Science Initiative (NMSI).

A variety of organizations played special roles during the development process. Achieve; the Maine Mathematics and Science Alliance; and staff of the Board on Science Education at the National Academies of Science, Engineering, and Medicine applied their experiences with state and national education frameworks to inform the K–12 Computer Science Framework's structure, grade-band expectations, and development process. Outlier Research & Evaluation in UChicago STEM Education at the University of Chicago documented the various meetings, convenings, and workshops; meeting summaries are available at k12cs.org.

The states that participated in the development of the K–12 Computer Science Framework were either currently engaged in or planning to begin the implementation of K–12 computer science. These states included Arkansas, California, Georgia, Idaho, Indiana, Iowa, Maryland, Massachusetts, Nebraska, Nevada, New Jersey, North Carolina, Utah, and Washington. School districts in Chicago (IL), New York City (NY), San Francisco (CA), and Charles County (MD) also participated. Each of the states and districts that committed to the project was asked to select a writer to represent the state on the writing team. State-level teams reviewed the framework and participated in convenings during the framework development process.

The development timeline consisted of a series of meetings with advisors, writers, and stakeholders from October 2015 to October 2016. The three advisor meetings focused on articulating the guiding vision and principles of the K–12 Computer Science Framework, identifying the core concepts and practices, and providing feedback during the development process. Over the course of two stakeholder convenings, one to launch the framework project and one at the midpoint, participants offered input, received development updates, and engaged in discussions around common computer science implementation issues. The seven writer workshops were opportunities for writers to collaborate in person and revise drafts of the framework based on public feedback. Writers and advisors held scheduled video conferences between the in-person gatherings. There were a total of three public review periods, as well as multiple internal reviews and focus groups covering special topics such as curriculum, computational thinking, and developmental appropriateness (including early childhood education).

This chapter details how the framework project began, how the community was involved, and how each part of the framework was developed.

Beginnings of the Framework

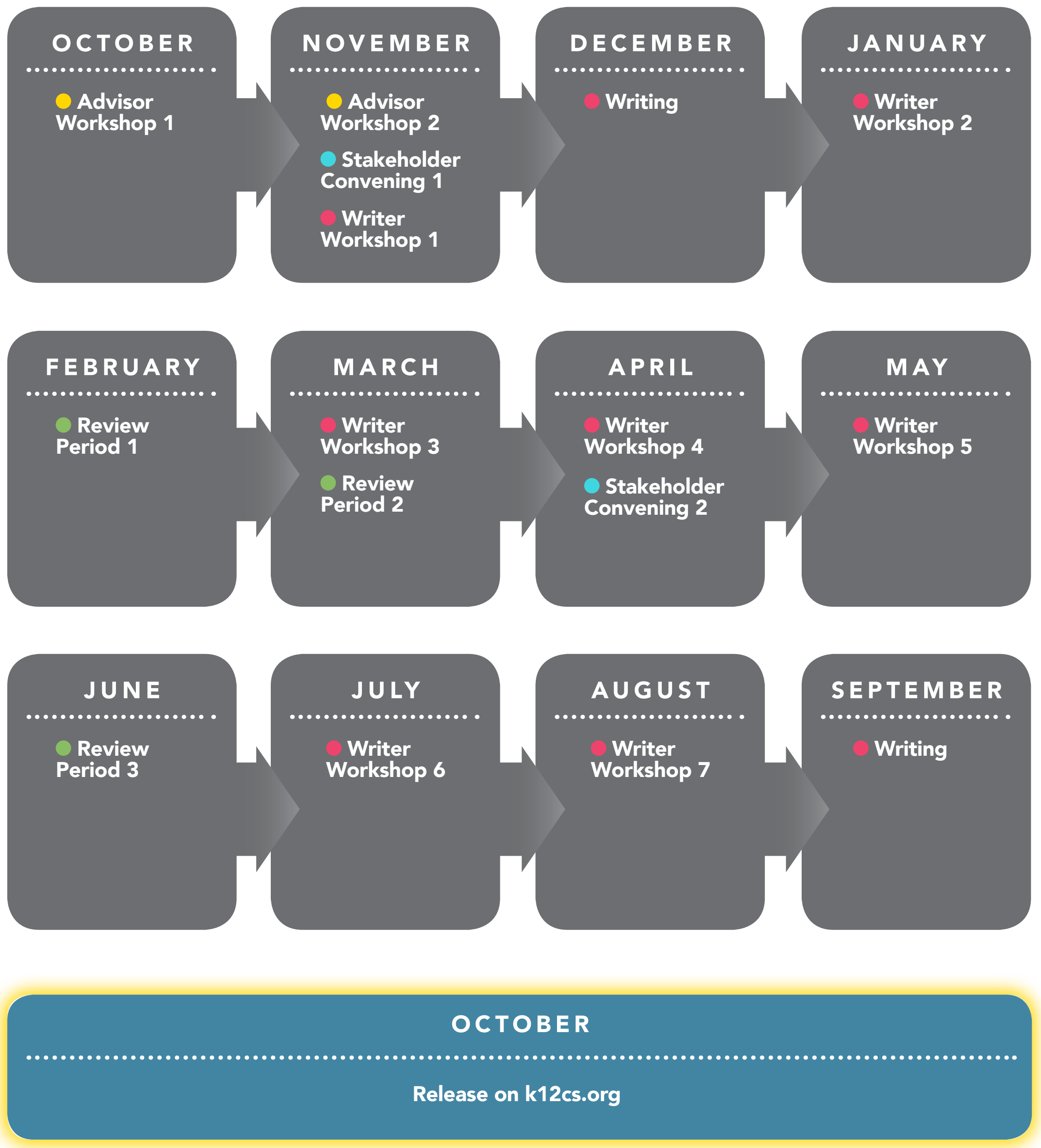
The impetus to develop the framework came from a mutual interest in K–12 computer science expressed by multiple states. With surging attention from schools, parents, and students across the nation, states were increasingly asking, "What should students be able to know and do in a K–12 computer science pathway? What does computer science look like in elementary, middle, and high school?" The K–12 Computer Science Framework represents a response to the growing need for state-focused guidance that reflects the consensus of a wide and diverse range of computer science practitioners, researchers, and organizations. Many of these states were wary of engaging in another wave of national education standards similar to those in mathematics, language arts, and science, reflecting the pace and politics of standards-based education reform over the last several years. Some states expressed fatigue from these national standards efforts and the desire for more flexible, high-level guidelines for K–12 computer science by which they could eventually create their own, state-developed guidance.

The framework builds upon previous publications that describe detailed expectations for K–12 computer science education in the U.S.: *A Model Curriculum for K–12 Computer Science, 2nd edition* (Tucker et al., 2006) and the *CSTA K–12 Computer Science Standards* (CSTA Standards Task Force, 2011). The two organizations behind these documents, the ACM and CSTA, joined with Code.org, a national nonprofit organization promoting computer science education, to form the initial steering committee for the K–12 Computer Science Framework, which would later include CIC and NMSI. The steering committee was charged with guiding the framework development process; overseeing the review process to ensure multiple opportunities for diverse community involvement; increasing coherence among the framework, standards, and other related documents; and representing the project to increase public awareness. Code.org provided staff to direct the development of the framework and manage the process. The project was funded by the ACM, Code.org, and NMSI. See a timeline of the project in Figure 3.1.

The framework builds upon previous publications that describe detailed expectations for K–12 computer science education.

The CSTA began a scheduled revision to its 2011 standards at the same time the framework's development process began. With two related, yet independent processes being undertaken in parallel in the computer science education community, it was important to ensure that the community was speaking with a coherent and consistent voice to national stakeholders. To improve alignment between the two projects, the co-chairs of the CSTA standards revision served as advisors to the K–12 Computer Science Framework, and all of the CSTA standards writers were asked to be writers of the framework, with eventually half accepting the role. In this way, both documents informed and supported each other. An interim version of the revised CSTA standards was released in July 2016 and was informed by multiple inputs, including drafts of the K–12 Computer Science Framework during the development process.

Figure 3.1: Framework development process



Community Involvement

The 27 writers of the framework represented the research community, K–12 education, professional associations, nonprofit organizations, state and district departments of education, and industry. The writing team was diverse based on grade-level experience and content expertise as well as gender, race, ethnicity, state, rural/urban/suburban experience, and institutional representation (nonprofit, research, industry, public/private education). Writers were assigned to teams based on their grade-level experience and content expertise. The five core concept teams and one practice team were composed of writers, a lead writer, and a facilitator.

There were 27 writers and 25 advisors representing states, districts, K–12, industry, and nonprofits.

There were also 25 advisors, representing K–12 and higher education, who were involved in initial meetings to identify the core concepts and practices of the framework and provided feedback during the development process. The advisors were practitioners and researchers who represented bodies of work and expertise that were valuable to the development of the framework. Advisors participated in internal reviews and worked closely with the development staff and writing teams to inform key content areas, provide research grounding, and help resolve content issues during development.

A variety of critical stakeholders were involved in the development of the framework. Participating states were represented by staff from the state department of education and/or members of the state board of education. Districts were represented by computer science coordinators and specialists. Representatives from industry, nonprofit organizations, research institutions, and national education organizations provided key input during stakeholder convenings.

Public review and feedback were essential to the development of the framework. A total of three drafts of the framework were shared for public comment in addition to a number of internal drafts that were shared with the framework's advisors. Reviews were submitted by groups and individuals representing most of the nation's states and seven international locations. Special focus groups were held to collect detailed feedback on topics that writers wanted input on and to assess the needs of particular audiences. Focus group participants and reviewers were from the computer science education community as well as the general education community and represented a variety of roles in education, such as teachers, administrators, higher education faculty, researchers, curriculum specialists, and technology coaches. At the end of each review period, the development staff reviewed all the input, identified major themes, and provided recommendations to the writing team. More information about the review process, the feedback received, and subsequent revisions can be found in **Appendix A: Feedback and Revisions**.

Structure

The decision to organize the framework by concepts and practices was based on the desire to align to and complement the structure of broadly adopted education frameworks for subjects such as math, science, and language arts. Particularly, the National Research Council's (NRC) Framework for K–12 Science Education (2012) served as a model and precedent. The NRC Framework has three dimensions: Disciplinary Core Ideas, Scientific and Engineering Practices, and Crosscutting Concepts. The K–12 Computer Science Framework shares two similar dimensions, in the form of core concepts and practices. Crosscutting concepts are not an explicit, third dimension in the K–12 Computer Science Framework but are integrated across the core concepts. The grade-band endpoints (Grades 2, 5, 8, and 12) for the K–12 Computer Science Framework concepts are also modeled off those of the NRC Framework and represent key stages of education, including recognition of the developmental difference between lower and upper elementary school students.

Core Concepts

The core concepts of the K–12 Computer Science Framework are categories that represent major content areas in the field of computer science. They represent specific areas of disciplinary importance rather than abstract, general ideas. This approach recognizes the importance of using specific content and context to organize bodies of knowledge (NRC, 2007), such as data, networks, and programming, over domain-general ideas such as abstraction and computational thinking. In a discussion about learning progressions in science, the NRC reported that a "disciplinary approach fits with the increasing recognition of the importance of specific content and context in thinking and learning and the power of theories to define and organize understandings of particular domains, something that domain-general ideas by their nature don't have the power to do" (NRC, 2007, p. 223). The criteria for selection of a core concept was that it should

- have broad significance across the field of computer science;
- serve as a useful foundation for learning or building to other ideas in computer science;
- allow young students to engage with the idea (low threshold), yet preserve the potential for progressive elaboration and sophistication (high ceiling);
- be applicable within other K–12 subjects and disciplines; and
- remain relevant in computer science over the next five to ten years, at minimum.

The core concepts of the framework:

1. Computing Systems
2. Networks and the Internet
3. Data and Analysis
4. Algorithms and Programming
5. Impacts of Computing

Writers identified subconcepts within each of the core concepts and used them to create focused, coherent learning progressions that span kindergarten to Grade 12. The selection of core concepts and subconcepts was informed by related K–12 computer science documents, including the ACM Model K–12 Curriculum (Tucker et al., 2006); the CSTA 2011 standards; the Advanced Placement® Computer Science Principles curriculum framework (College Board, 2016); the Denning Institute's Great Principles of Computing (Denning & Martell, 2015); and international frameworks, such as the United Kingdom's national computing program of study (England Department for Education, 2013). Frameworks from related disciplines, such as cybersecurity, digital citizenship, and technology literacy, also informed the learning progressions. The framework writers' experience and expertise with diverse populations of students proved invaluable when determining what ideas were essential for all students.

A learning progression describes conceptual milestones along a path that move from basic understanding to more sophisticated knowledge in a subject area. The framework's subconcepts provide a focal point for these learning progressions to connect learning across grades and articulate the essential ideas under each core concept (Hess, 2008). Rather than prioritizing the coverage of a wide range of content, the framework's learning progressions deliberately revisit a subconcept across multiple grade bands with evolving sophistication. For example, the learning progression for modularity in the *Algorithms and Programming* core concept begins with the simple understanding that tasks can be broken down into smaller tasks and that programs can be composed of parts of other programs. Eventually, students understand that a program is a system of interacting modules, including other programs.

Cybersecurity, digital citizenship, and technology literacy informed the framework's learning progressions.

The K–12 Computer Science Framework employed principles used in the construction and research of learning progressions, such as identifying emergent core ideas that can be introduced early in a child's education and elaborated on over multiple years (NRC, 2007). Where specific computer science research was lacking, especially in early grade bands, related science and math research was used to approximate the appropriate computer science progressions and guide the placement of concepts in particular grade bands. For example, procedural abstraction, in which procedures use variables as parameters to generalize behavior, was placed as an expectation by the end of eighth grade in the core concept of *Algorithms and Programming* based on the placement of a related concept in learning progressions for mathematics—writing equations with variables in the middle grades. Other examples of relying on learning progressions from other disciplines include using science learning progressions to inform the placement of concepts shared with computer science, such as models and simulations, and analogous concepts, such as bits (basic units of digital information) and particles/atoms.

A learning progression describes conceptual milestones along a path that move from basic understanding to more sophisticated knowledge in a subject area.

It should be noted that the effectiveness of the framework's learning progressions will be significantly influenced by teachers' pedagogical practices, as traditional methods of teaching computer science may not enable all students to meet the framework's expectations. Additionally, the framework's learning progressions guide, but do not fully prescribe an instructional sequence—there are flexible paths for moving between the expectations at the end of each grade band.

Crosscutting Concepts

As mentioned previously, the core concepts of the K–12 Computer Science Framework represent specific areas of disciplinary importance rather than abstract, domain-general ideas. The latter formed the basis of the framework's crosscutting concepts, ideas that have application across the different core concepts and are integrated into concept statements. These "crosscutting concepts" provide thematic connections across the core concepts. The criteria for selection of a crosscutting concept was that it should

- apply across multiple core concepts,
- illuminate connections between different core concepts of computer science,
- build familiarity with fundamental themes in computer science through repetition in different contexts, and
- create a richer understanding of a concept statement in which it is integrated.

The crosscutting concepts of the framework (listed in order of frequency in the framework):

1. Abstraction
2. System Relationships
3. Human–Computer Interaction
4. Privacy and Security
5. Communication and Coordination

Although the K–12 Computer Science Framework and the NRC Framework for K–12 Science Education define crosscutting concepts in similar ways, the K–12 Computer Science Framework integrates them into the learning progressions under each core concept rather than as a separate third dimension alongside the core concepts and practices. This integration was done for two main reasons. First, it was decided that integrating them into the existing dimension of core concepts would preserve their value and influence while making it easier for the audience of the framework to understand and implement the framework. Second, the research base and practitioner experience with crosscutting concepts in computer science did not provide enough information to create a separate learning progression for crosscutting concepts.

The framework integrates crosscutting concepts under each core concept rather than as a third dimension.

The crosscutting concepts served as an internal writing tool during the development process, and the list evolved based on feedback and as writing progressed. Writers intentionally integrated crosscutting concepts into the concept statements during the writing process. Toward the end of the development process, a small team of writers went through the entire framework and tagged concept statements by particular crosscutting concepts and also suggested revisions to concept statements to create a stronger and more explicit integration with a crosscutting concept. The crosscutting concepts listed in the descriptive material for each concept statement are not the only connections that can be made—just the ones that are the most relevant. In addition, some crosscutting concepts are more obvious than others, but all of them provide opportunities for illuminating key themes in computer science that cut across the five core concepts of the framework. Users of the framework should decide the depth to which a crosscutting concept is emphasized when a concept statement is being addressed.

The following examples illustrate the crosscutting concept System Relationships in statements from different core concept areas. An expectation by the end of 12th grade in the *Algorithms and Programming* core concept begins, "Complex programs are designed as systems of interacting modules, each with a specific role, coordinating for a common overall purpose. These modules can be procedures within a program; combinations of data and procedures; or independent, but interrelated, programs" (9–12.Algorithms and Programming.Modularity). This example illustrates how the different parts of a program create a system and interact for a common purpose. In the *Data and Analysis* core concept, by the end of 12th grade, students are expected to understand the following: "Data can be composed of multiple data elements that relate to one another. . . . People make choices about how data elements are organized and where data is stored" (9–12.Data and Analysis. Storage). This statement applies the idea of systems to organizations of data and the relationships among different data elements. Most obviously, the idea of systems comes up often in the *Computing Systems* core concept: "Levels of interaction exist between the hardware, software, and user of a computing system. The most common levels of software that a user interacts with include system software and applications" (9–12.Computing Systems.Hardware and Software). This statement describes the role of system software as well as the relationships among the hardware, software, and user in a computing system. These three examples show the power of a crosscutting concept to illuminate a key aspect of computer science across the core concepts of the framework.

Detailed descriptions of each crosscutting concept can be found in the preface of the **Concepts** chapter.

Connections Within the Framework

Many concept statements in the framework relate to other concept statements, whether in the same grade band or a different one. Writing teams assigned to particular concepts intentionally worked with other teams to align content across the framework. The bolded phrases in Figure 3.2 highlight the relationship between the concept statement at the 9–12 grade band of the *Networks and the Internet* core concept (hierarchy in the Internet) and the statement in the same grade band of the *Computing Systems* core concept (layers of interaction in hardware and software).

Figure 3.2: Example of connection between two concepts in the same grade band

9–12.Networks and the Internet.Network Communication and Organization	9–12.Computing Systems.Hardware and Software
Network topology is determined, in part, by how many devices can be supported. Each device is assigned an address that uniquely identifies it on the network. The scalability and reliability of the Internet are enabled by the **hierarchy** and redundancy in networks.	**Levels of interaction exist between the hardware, software, and user of a computing system**. The most common levels of software that a user interacts with include system software and applications. System software controls the flow of information between hardware components used for input, output, storage, and processing.

Connections also exist between statements in different grade bands and describe how one concept may build to another. For example, the concept statement at the end of the K–2 grade band of the *Algorithms and Programming* core concept (programs are developed to express ideas and address problems) provides a foundation for understanding a statement at the end of the 3–5 grade band of the *Impacts of Computing* core concept (development of computing technology is driven by people's needs and wants and influences cultural practices). See Figure 3.3 for details.

Figure 3.3: Example of connection between two concepts in different grade bands

K–2.Algorithms and Programming.Program Development	3–5.Impacts of Computing.Culture
People develop programs collaboratively and for a purpose, **such as expressing ideas or addressing problems**.	The **development and modification of computing technology is driven by people's needs and wants** and can affect groups differently. **Computing technologies influence, and are influenced by, cultural practices.**

Concept statements within the same grade band and core concept are inherently connected. For example, the Modularity and Program Development concept statements for the 3–5 grade band in *Algorithms and Programming* are related (see Figure 3.4).

Figure 3.4: Example of connection between two statements in the same core concept and grade band

3–5.Algorithms and Programming. Modularity	3–5.Algorithms and Programming.Program Development
Programs can be broken down into smaller parts to facilitate their design, implementation, and review. **Programs can also be created by incorporating smaller portions of programs that have already been created**.	People develop programs using an iterative process involving design, implementation, and review. **Design often involves reusing existing code or remixing other programs within a community.** People continuously review whether programs work as expected, and they fix, or debug, parts that do not. Repeating these steps enables people to refine and improve programs.

Core Practices

The K–12 Computer Science Framework's practices are the behaviors that computationally literate students use to fully engage with the core concepts of computer science. Concepts and practices are integrated to provide complete experiences for students engaging in computer science. The criteria for selection of a practice was that it should

- capture important behaviors that computer scientists engage in,
- be helpful to fully explore and understand the framework concepts,
- help students engage with course content through the development of artifacts, and
- be based on processes and proficiencies with importance in computer science.

Like the core concepts of the framework, the framework's practices were also informed by the descriptions of practices in national frameworks and standards in other subjects. The practices intentionally overlap with those in other disciplines and use similar language to help teachers make connections between computer science and disciplines they are more familiar with and to make the framework more accessible to a wide audience.

The practices intentionally overlap with those in other disciplines and use similar language.

The core practices of the framework:

1. Fostering an Inclusive Computing Culture
2. Collaborating Around Computing
3. Recognizing and Defining Computational Problems
4. Developing and Using Abstractions
5. Creating Computational Artifacts
6. Testing and Refining Computational Artifacts
7. Communicating About Computing

Computational thinking plays a key role in the computer science practices of the framework as it encompasses practices 3, 4, 5, and 6. Practices 1, 2, and 7 are independent, general practices in computer science that complement computational thinking. Multiple research articles and documents informed the delineation of computational thinking practices, such as *Operational Definition of Computational Thinking for K–12 Education* (ISTE & CSTA, 2011) and *Assessment Design Patterns for Computational Thinking Practices in Secondary Computer Science* (Bienkowski, Snow, Rutstein, & Grover, 2015).

The practice statements delineate specific expectations by the end of 12th grade and are followed by a narrative that describes the progression leading to those end points. This structure differs from the grade-band delineation of the core concepts because the current research base and practitioner experience with practices in computer science do not provide enough information to create clear, grade-banded expectations. The narratives describe the practice progressions in a manner that is less prescriptive about developmental appropriateness to emphasize flexible expectations.

References

Bienkowski, M., Snow, E., Rutstein, D. W., & Grover, S. (2015). *Assessment design patterns for computational thinking practices in secondary computer science: A first look* (SRI technical report). Menlo Park, CA: SRI International. Retrieved from http://pact.sri.com/resources.html

College Board. (2016). *AP Computer Science Principles course and exam description*. New York, NY: College Board. Retrieved from https://secure-media.collegeboard.org/digitalServices/pdf/ap/ap-computer-science-principles-course-and-exam-description.pdf

Computer Science Teachers Association Standards Task Force. (2011). *CSTA K–12 computer science standards, revised 2011.* New York, NY: Computer Science Teachers Association and Association for Computing Machinery. Retrieved from http://www.csteachers.org/resource/resmgr/Docs/Standards/CSTA_K-12_CSS.pdf

Denning, P. J., & Martell, C. (2015). *Great principles of computing.* Cambridge, MA: MIT Press.

England Department for Education. (2013, September 11). *National curriculum in England: Computing programmes of study.* Retrieved from https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study/national-curriculum-in-england-computing-programmes-of-study

Hess, K. (2008). *Developing and using learning progressions as a schema for measuring progress.* Dover, NH: National Center for the Improvement of Educational Assessment. Retrieved from http://www.nciea.org/publications/CCSSO2_KH08.pdf

International Society for Technology in Education & Computer Science Teachers Association. (2011). *Operational definition of computational thinking for K–12 education.* Retrieved from https://csta.acm.org/Curriculum/sub/CurrFiles/CompThinkingFlyer.pdf

National Research Council. (2007). *Taking science to school: Learning and teaching science in grades K–8.* Committee on Science Learning-Kindergarten Through Eighth Grade. R. A. Duschl, H. A. Schweingruber, & A. W. Shouse (Eds.). Board on Science Education, Center for Education. Division of Behavioral and Social Sciences and Education. Washington, DC: The National Academies Press.

National Research Council. (2012). *A framework for K–12 science education: Practices, crosscutting concepts, and core ideas.* Committee on a Conceptual Framework for New K–12 Science Education Standards. Board on Science Education, Division of Behavioral and Social Sciences and Education. Washington, DC: The National Academies Press.

Tucker, A., McCowan, D., Deek, F., Stephenson, C., Jones, J., & Verno, A. (2006). *A model curriculum for K–12 computer science: Report of the ACM K–12 task force curriculum committee* (2nd ed.). New York, NY: Association for Computing Machinery.

Navigating the Framework

4

Navigating the Framework

The purpose of this chapter is to help you navigate the parts of the framework. First, this chapter describes the content of the framework, which is chiefly made up of practices, concepts, and guidance chapters. Second, this chapter explains the different ways you can view the framework online so that you can have the information organized according to your purpose.

Practices

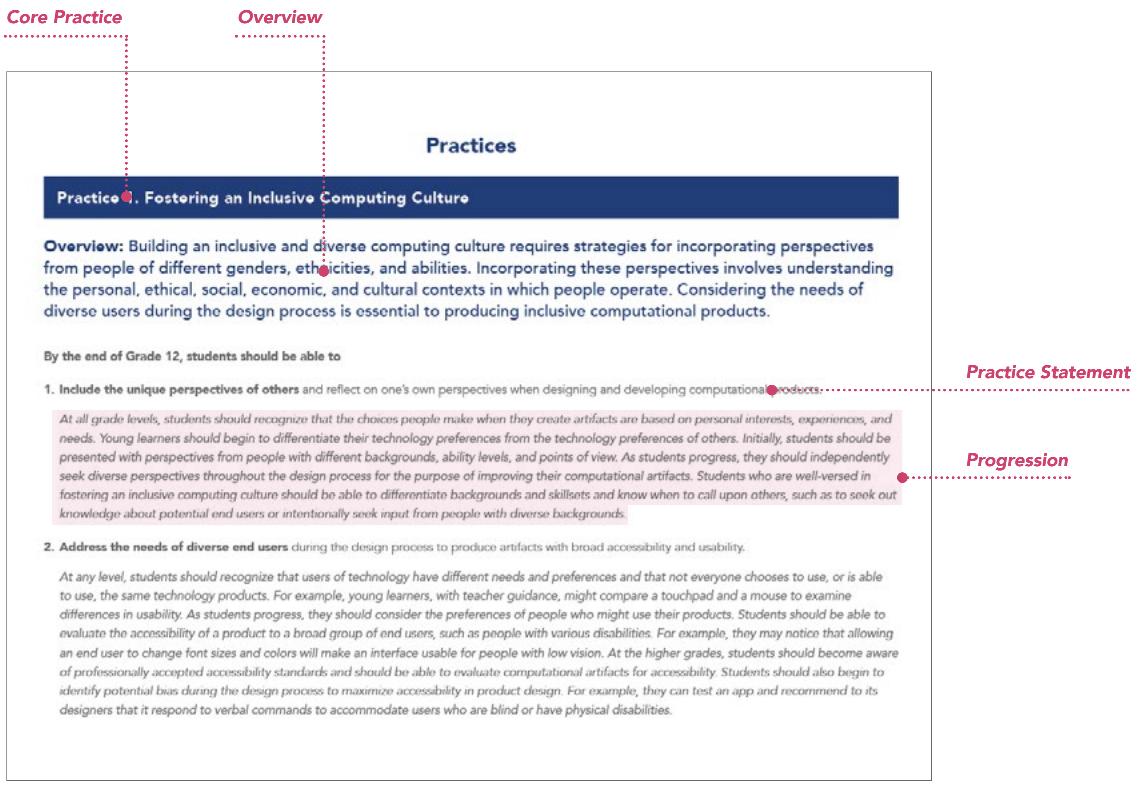
The K–12 Computer Science Framework's practices are the behaviors that computationally literate students use to fully engage with the core concepts of computer science. Concepts and practices are integrated to provide complete experiences for students engaging in computer science. While the practices naturally integrate and overlap with one another, they are displayed in an order that suggests a process for developing computational artifacts. Four of the practices are also called out as aspects of computational thinking.

There are seven **core practices**:

1. Fostering an Inclusive Computing Culture
2. Collaborating Around Computing
3. Recognizing and Defining Computational Problems
4. Developing and Using Abstractions
5. Creating Computational Artifacts
6. Testing and Refining Computational Artifacts
7. Communicating About Computing

How to Read the Practices

Figure 4.1: How to read the practices


Core Practice
Overview
Practice Statement
Progression

Practices

Practice 1. Fostering an Inclusive Computing Culture

Overview: Building an inclusive and diverse computing culture requires strategies for incorporating perspectives from people of different genders, ethnicities, and abilities. Incorporating these perspectives involves understanding the personal, ethical, social, economic, and cultural contexts in which people operate. Considering the needs of diverse users during the design process is essential to producing inclusive computational products.

By the end of Grade 12, students should be able to

1. **Include the unique perspectives of others** and reflect on one's own perspectives when designing and developing computational products.

   At all grade levels, students should recognize that the choices people make when they create artifacts are based on personal interests, experiences, and needs. Young learners should begin to differentiate their technology preferences from the technology preferences of others. Initially, students should be presented with perspectives from people with different backgrounds, ability levels, and points of view. As students progress, they should independently seek diverse perspectives throughout the design process for the purpose of improving their computational artifacts. Students who are well-versed in fostering an inclusive computing culture should be able to differentiate backgrounds and skillsets and know when to call upon others, such as to seek out knowledge about potential end users or intentionally seek input from people with diverse backgrounds.

2. **Address the needs of diverse end users** during the design process to produce artifacts with broad accessibility and usability.

   At any level, students should recognize that users of technology have different needs and preferences and that not everyone chooses to use, or is able to use, the same technology products. For example, young learners, with teacher guidance, might compare a touchpad and a mouse to examine differences in usability. As students progress, they should consider the preferences of people who might use their products. Students should be able to evaluate the accessibility of a product to a broad group of end users, such as people with various disabilities. For example, they may notice that allowing an end user to change font sizes and colors will make an interface usable for people with low vision. At the higher grades, students should become aware of professionally accepted accessibility standards and should be able to evaluate computational artifacts for accessibility. Students should also begin to identify potential bias during the design process to maximize accessibility in product design. For example, they can test an app and recommend to its designers that it respond to verbal commands to accommodate users who are blind or have physical disabilities.

Each practice contains three parts: (1) **overview**, (2) **practice statement**, and (3) **progression**.

1. The **overview** describes the practice.
2. The **practice statement** describes what students should be able to do when exiting Grade 12.
3. The **progression** under each goal describes how students should be exhibiting the specific practice with increasing sophistication from kindergarten to Grade 12. Rather than grade bands, the progressions use a narrative format to emphasize the different paths students may take in their development of the practices. The examples in the progressions describe what all students could do but are not mandatory.

How to Refer to the Practices

When referring to a particular practice statement, use the following notation:

P[Practice Number].[Core Practice].[Practice Statement Number]

Examples:

- P4.Developing and Using Abstractions.1
- P2.Collaborating Around Computing.3

When a practice statement is referenced within the narrative of another practice, it is denoted without the name of the core practice (e.g., P4.1).

Concepts

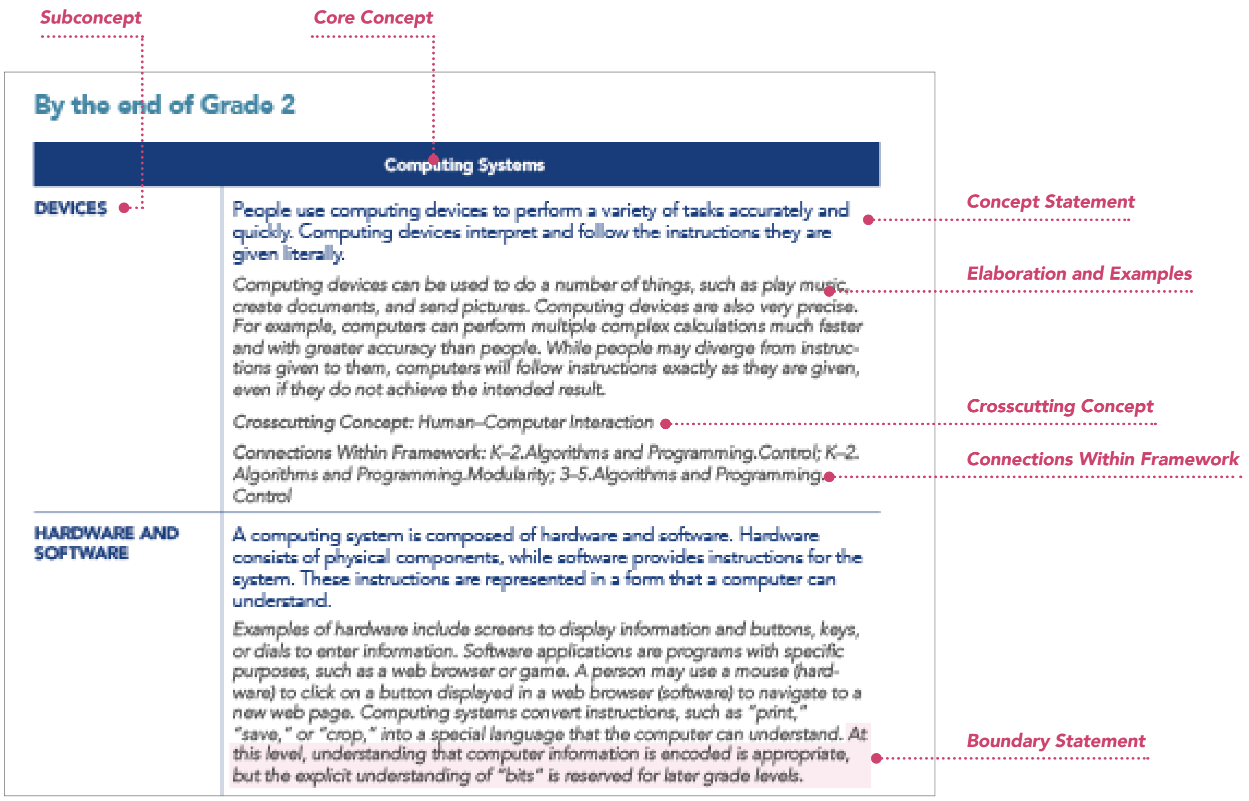
A core concept represents a specific area of disciplinary importance in computer science. There are five **core concepts**:

1. Computing Systems
2. Networks and the Internet
3. Data and Analysis
4. Algorithms and Programming
5. Impacts of Computing

Each core concept is described in an overview and delineated by multiple **subconcepts** that represent the specific ideas within that core concept. For example, the Data and Analysis core concept contains four subconcepts: Collection, Storage, Visualization and Transformation, and Inference and Models. **Subconcept overviews** are provided to describe the subconcepts and summarize how learning progresses across multiple grade bands.

How to Read the Concepts

Figure 4.2: How to read the concepts



By the end of Grade 2

Computing Systems

DEVICES	People use computing devices to perform a variety of tasks accurately and quickly. Computing devices interpret and follow the instructions they are given literally. *Computing devices can be used to do a number of things, such as play music, create documents, and send pictures. Computing devices are also very precise. For example, computers can perform multiple complex calculations much faster and with greater accuracy than people. While people may diverge from instructions given to them, computers will follow instructions exactly as they are given, even if they do not achieve the intended result.* *Crosscutting Concept: Human–Computer Interaction* *Connections Within Framework: K–2.Algorithms and Programming.Control; K–2.Algorithms and Programming.Modularity; 3–5.Algorithms and Programming.Control*
HARDWARE AND SOFTWARE	A computing system is composed of hardware and software. Hardware consists of physical components, while software provides instructions for the system. These instructions are represented in a form that a computer can understand. *Examples of hardware include screens to display information and buttons, keys, or dials to enter information. Software applications are programs with specific purposes, such as a web browser or game. A person may use a mouse (hardware) to click on a button displayed in a web browser (software) to navigate to a new web page. Computing systems convert instructions, such as "print," "save," or "crop," into a special language that the computer can understand. At this level, understanding that computer information is encoded is appropriate, but the explicit understanding of "bits" is reserved for later grade levels.*

The **concept statements** in the framework describe conceptual milestones at different grade-band endpoints: Grades 2, 5, 8, and 12. Each concept statement encompasses a significant, essential idea in computer science, and all are considered equally important.

Each concept statement is accompanied by descriptive material, including **elaboration and examples**. Some concept statements also include **crosscutting concepts** and **connections within the framework.**

1. The **elaboration and examples** add detail and depth to the concept statements. **Boundary statements** are included to clarify what is not expected to be learned at that grade level.
2. **Crosscutting concepts** illuminate thematic connections across the different core concepts and are integrated into concept statements as relevant and appropriate. When multiple crosscutting concepts are listed under a statement, they are in order of significance rather than alphabetical order.
   There are five crosscutting concepts:
   1. Abstraction
   2. System Relationships
   3. Human–Computer Interaction
   4. Privacy and Security
   5. Communication and Coordination
3. **Connections within the framework** are provided to illuminate relationships among concept statements across different core concepts. Connections in the same grade band help guide when concepts can be addressed together.

How to Refer to the Concepts

When referring to a particular concept statement, use the following notation: [Grade Band].[Core Concept].[Subconcept]

Examples:

- 3–5.Impacts of Computing.Culture
- K–2.Algorithms and Programming.Program Development

Other Parts of the Framework

The entire framework document consists of concepts, practices, and guidance chapters. The framework document also includes a glossary of key technical terms used in the concept and practice statements (**Appendix C**). Other resources are available on the website, such as handouts, with more to be posted as they are created.

Viewing the Framework

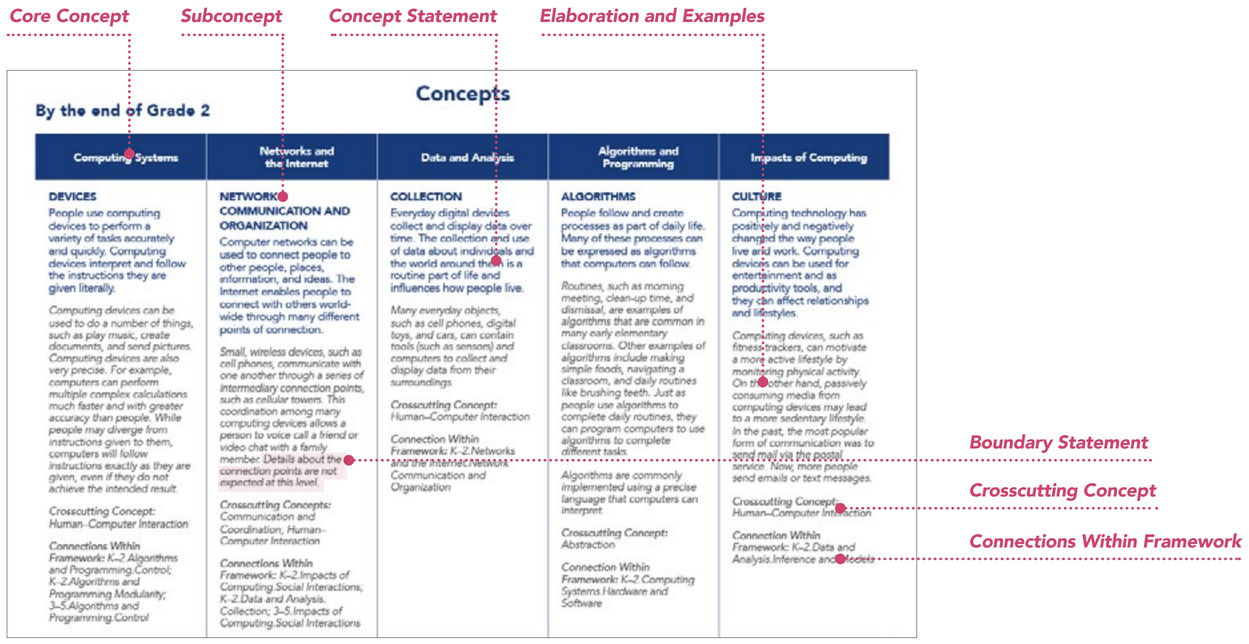
It is expected that people viewing the framework will have different goals. Therefore, the framework is viewable online in a variety of ways to fit your needs.

The concepts and practices can be viewed online in three different views. All three views include the practices first and then display the concepts by the selected view. All three views also allow the user to filter by grade bands, core concepts, core practices, and crosscutting concepts.

Each view of the framework can be downloaded as a separate PDF file.

Grade Band View

Figure 4.3: Grade band view



Concepts

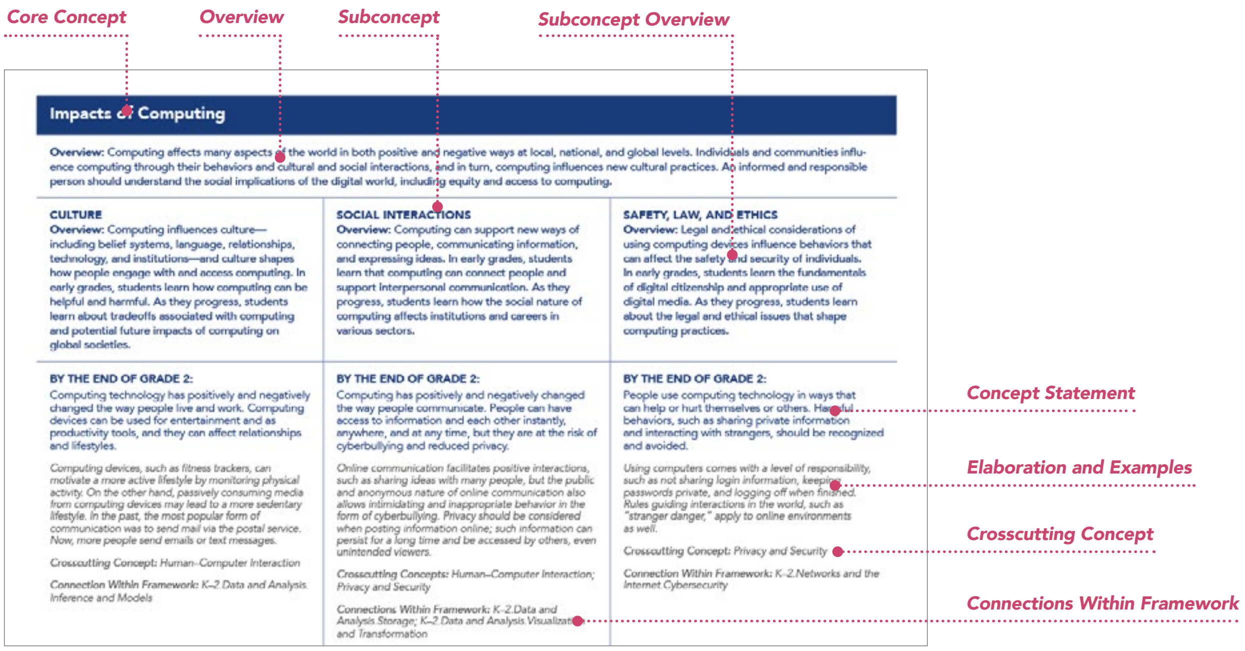
By the end of Grade 2

Computing Systems	Networks and the Internet	Data and Analysis	Algorithms and Programming	Impacts of Computing
DEVICES People use computing devices to perform a variety of tasks accurately and quickly. Computing devices interpret and follow the instructions they are given literally. *Computing devices can be used to do a number of things, such as play music, create documents, and send pictures. Computing devices are also very precise. For example, computers can perform multiple complex calculations much faster and with greater accuracy than people. While people may diverge from instructions given to them, computers will follow instructions exactly as they are given, even if they do not achieve the intended result.* *Crosscutting Concept: Human–Computer Interaction* *Connections Within Framework: K–2.Algorithms and Programming.Control; K–2.Algorithms and Programming.Modularity; 3–5.Algorithms and Programming.Control*	**NETWORK COMMUNICATION AND ORGANIZATION** Computer networks can be used to connect people to other people, places, information, and ideas. The Internet enables people to connect with others worldwide through many different points of connection. *Small, wireless devices, such as cell phones, communicate with one another through a series of intermediary connection points, such as cellular towers. This coordination among many computing devices allows a person to voice call a friend or video chat with a family member. Details about the connection points are not expected at this level.* *Crosscutting Concepts: Communication and Coordination; Human–Computer Interaction* *Connections Within Framework: K–2.Impacts of Computing.Social Interactions; K–2.Data and Analysis.Collection; 3–5.Impacts of Computing.Social Interactions*	**COLLECTION** Everyday digital devices collect and display data over time. The collection and use of data about individuals and the world around them is a routine part of life and influences how people live. *Many everyday objects, such as cell phones, digital toys, and cars, can contain tools (such as sensors) and computers to collect and display data from their surroundings.* *Crosscutting Concept: Human–Computer Interaction* *Connection Within Framework: K–2.Networks and the Internet.Network Communication and Organization*	**ALGORITHMS** People follow and create processes as part of daily life. Many of these processes can be expressed as algorithms that computers can follow. *Routines, such as morning meeting, clean-up time, and dismissal, are examples of algorithms that are common in many early elementary classrooms. Other examples of algorithms include making simple foods, navigating a classroom, and daily routines like brushing teeth. Just as people use algorithms to complete daily routines, they can program computers to use algorithms to complete different tasks.* *Algorithms are commonly implemented using a precise language that computers can interpret.* *Crosscutting Concept: Abstraction* *Connection Within Framework: K–2.Computing Systems.Hardware and Software*	**CULTURE** Computing technology has positively and negatively changed the way people live and work. Computing devices can be used for entertainment and as productivity tools, and they can affect relationships and lifestyles. *Computing devices, such as fitness trackers, can motivate a more active lifestyle by monitoring physical activity. On the other hand, passively consuming media from computing devices may lead to a more sedentary lifestyle. In the past, the most popular form of communication was to send mail via the postal service. Now, more people send emails or text messages.* *Crosscutting Concept: Human–Computer Interaction* *Connection Within Framework: K–2.Data and Analysis.Inference and Models*

Viewing by grade band allows you to view statements by grade band first. Under each grade band, concept statements are organized by core concept and listed under the associated subconcept. This organization is useful for a user who wants to view a single grade band, such as K–2 only or 6–8 only, for example.

Progression View

Figure 4.4: Progression view



Impacts of Computing

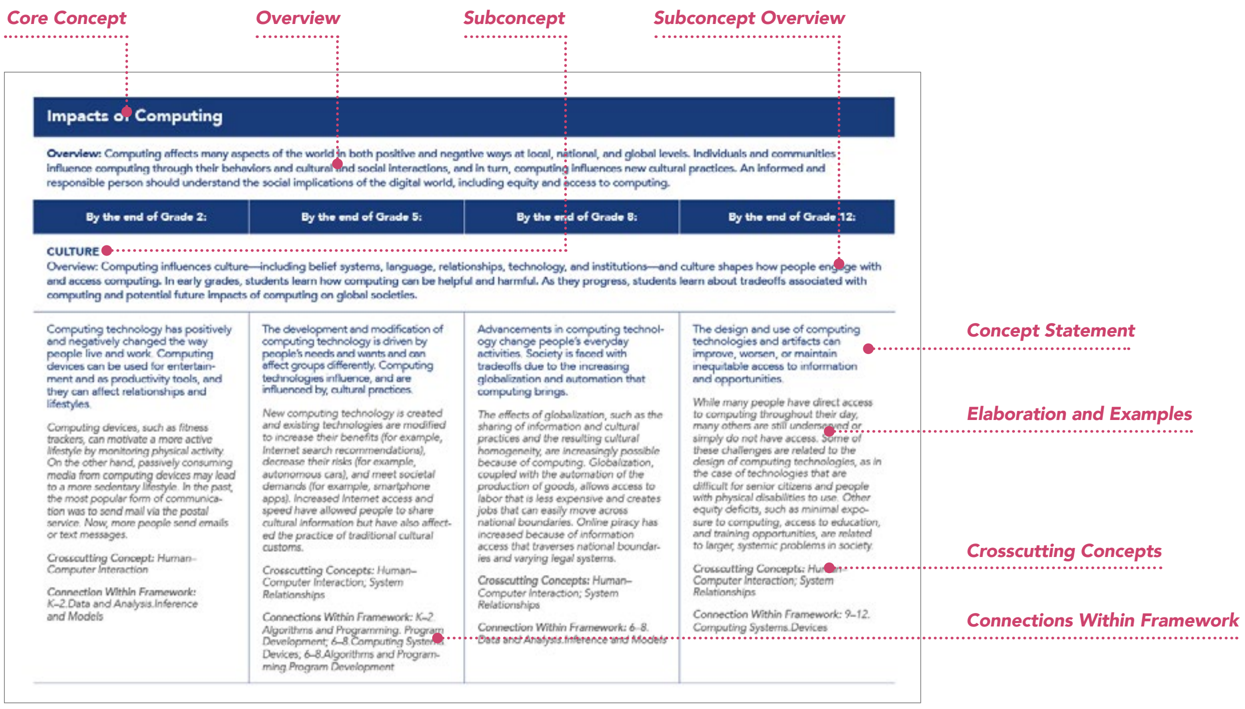
Overview: Computing affects many aspects of the world in both positive and negative ways at local, national, and global levels. Individuals and communities influence computing through their behaviors and cultural and social interactions, and in turn, computing influences new cultural practices. An informed and responsible person should understand the social implications of the digital world, including equity and access to computing.

CULTURE	**SOCIAL INTERACTIONS**	**SAFETY, LAW, AND ETHICS**
Overview: Computing influences culture—including belief systems, language, relationships, technology, and institutions—and culture shapes how people engage with and access computing. In early grades, students learn how computing can be helpful and harmful. As they progress, students learn about tradeoffs associated with computing and potential future impacts of computing on global societies.	**Overview:** Computing can support new ways of connecting people, communicating information, and expressing ideas. In early grades, students learn that computing can connect people and support interpersonal communication. As they progress, students learn how the social nature of computing affects institutions and careers in various sectors.	**Overview:** Legal and ethical considerations of using computing devices influence behaviors that can affect the safety and security of individuals. In early grades, students learn the fundamentals of digital citizenship and appropriate use of digital media. As they progress, students learn about the legal and ethical issues that shape computing practices.
BY THE END OF GRADE 2: Computing technology has positively and negatively changed the way people live and work. Computing devices can be used for entertainment and as productivity tools, and they can affect relationships and lifestyles.	**BY THE END OF GRADE 2:** Computing has positively and negatively changed the way people communicate. People can have access to information and each other instantly, anywhere, and at any time, but they are at the risk of cyberbullying and reduced privacy.	**BY THE END OF GRADE 2:** People use computing technology in ways that can help or hurt themselves or others. Harmful behaviors, such as sharing private information and interacting with strangers, should be recognized and avoided.
Computing devices, such as fitness trackers, can motivate a more active lifestyle by monitoring physical activity. On the other hand, passively consuming media from computing devices may lead to a more sedentary lifestyle. In the past, the most popular form of communication was to send mail via the postal service. Now, more people send emails or text messages.	*Online communication facilitates positive interactions, such as sharing ideas with many people, but the public and anonymous nature of online communication also allows intimidating and inappropriate behavior in the form of cyberbullying. Privacy should be considered when posting information online; such information can persist for a long time and be accessed by others, even unintended viewers.*	*Using computers comes with a level of responsibility, such as not sharing login information, keeping passwords private, and logging off when finished. Rules guiding interactions in the world, such as "stranger danger," apply to online environments as well.*
Crosscutting Concept: Human–Computer Interaction	*Crosscutting Concepts: Human–Computer Interaction; Privacy and Security*	*Crosscutting Concept: Privacy and Security*
Connection Within Framework: K–2.Data and Analysis.Inference and Models	*Connections Within Framework: K–2.Data and Analysis.Storage; K–2.Data and Analysis.Visualization and Transformation*	*Connection Within Framework: K–2.Networks and the Internet.Cybersecurity*

Viewing by progression allows you to view the concept statements organized by core concept first and then by subconcept. This view also displays the overviews of the core concepts and subconcepts. This view is good for seeing the progression of student understanding across the grade bands.

Concept View

Figure 4.5: Concept view



Impacts of Computing

Overview: Computing affects many aspects of the world in both positive and negative ways at local, national, and global levels. Individuals and communities influence computing through their behaviors and cultural and social interactions, and in turn, computing influences new cultural practices. An informed and responsible person should understand the social implications of the digital world, including equity and access to computing.

By the end of Grade 2:	By the end of Grade 5:	By the end of Grade 8:	By the end of Grade 12:
CULTURE Overview: Computing influences culture—including belief systems, language, relationships, technology, and institutions—and culture shapes how people engage with and access computing. In early grades, students learn how computing can be helpful and harmful. As they progress, students learn about tradeoffs associated with computing and potential future impacts of computing on global societies.			
Computing technology has positively and negatively changed the way people live and work. Computing devices can be used for entertainment and as productivity tools, and they can affect relationships and lifestyles.	The development and modification of computing technology is driven by people's needs and wants and can affect groups differently. Computing technologies influence, and are influenced by, cultural practices.	Advancements in computing technology change people's everyday activities. Society is faced with tradeoffs due to the increasing globalization and automation that computing brings.	The design and use of computing technologies and artifacts can improve, worsen, or maintain inequitable access to information and opportunities.
Computing devices, such as fitness trackers, can motivate a more active lifestyle by monitoring physical activity. On the other hand, passively consuming media from computing devices may lead to a more sedentary lifestyle. In the past, the most popular form of communication was to send mail via the postal service. Now, more people send emails or text messages.	*New computing technology is created and existing technologies are modified to increase their benefits (for example, Internet search recommendations), decrease their risks (for example, autonomous cars), and meet societal demands (for example, smartphone apps). Increased Internet access and speed have allowed people to share cultural information but have also affected the practice of traditional cultural customs.*	*The effects of globalization, such as the sharing of information and cultural practices and the resulting cultural homogeneity, are increasingly possible because of computing. Globalization, coupled with the automation of the production of goods, allows access to labor that is less expensive and creates jobs that can easily move across national boundaries. Online piracy has increased because of information access that traverses national boundaries and varying legal systems.*	*While many people have direct access to computing throughout their day, many others are still underserved or simply do not have access. Some of these challenges are related to the design of computing technologies, as in the case of technologies that are difficult for senior citizens and people with physical disabilities to use. Other equity deficits, such as minimal exposure to computing, access to education, and training opportunities, are related to larger, systemic problems in society.*
Crosscutting Concept: *Human–Computer Interaction*	Crosscutting Concepts: *Human–Computer Interaction; System Relationships*	Crosscutting Concepts: *Human–Computer Interaction; System Relationships*	Crosscutting Concepts: *Human–Computer Interaction; System Relationships*
Connection Within Framework: *K–2.Data and Analysis.Inference and Models*	Connections Within Framework: *K–2 Algorithms and Programming. Program Development; 6–8.Computing Systems.Devices; 6–8.Algorithms and Programming.Program Development*	Connection Within Framework: *6–8.Data and Analysis.Inference and Models*	Connection Within Framework: *9–12.Computing Systems.Devices*

Viewing by concept allows you to see the statements organized by core concept first and then grade band. This view is useful for getting a picture of everything students in a particular grade level would need to know in a given core concept—for example, what a student in Grades 3–5 would need to know in the *Computing Systems* core concept.

Accessing the Complete Framework

The entire framework document, with concepts, practices, and guidance chapters, is available as a download at k12cs.org. Guidance chapters of the framework can be viewed online in an abridged format.

Practices

Including Computational Thinking

Practices *Including Computational Thinking*

Preface

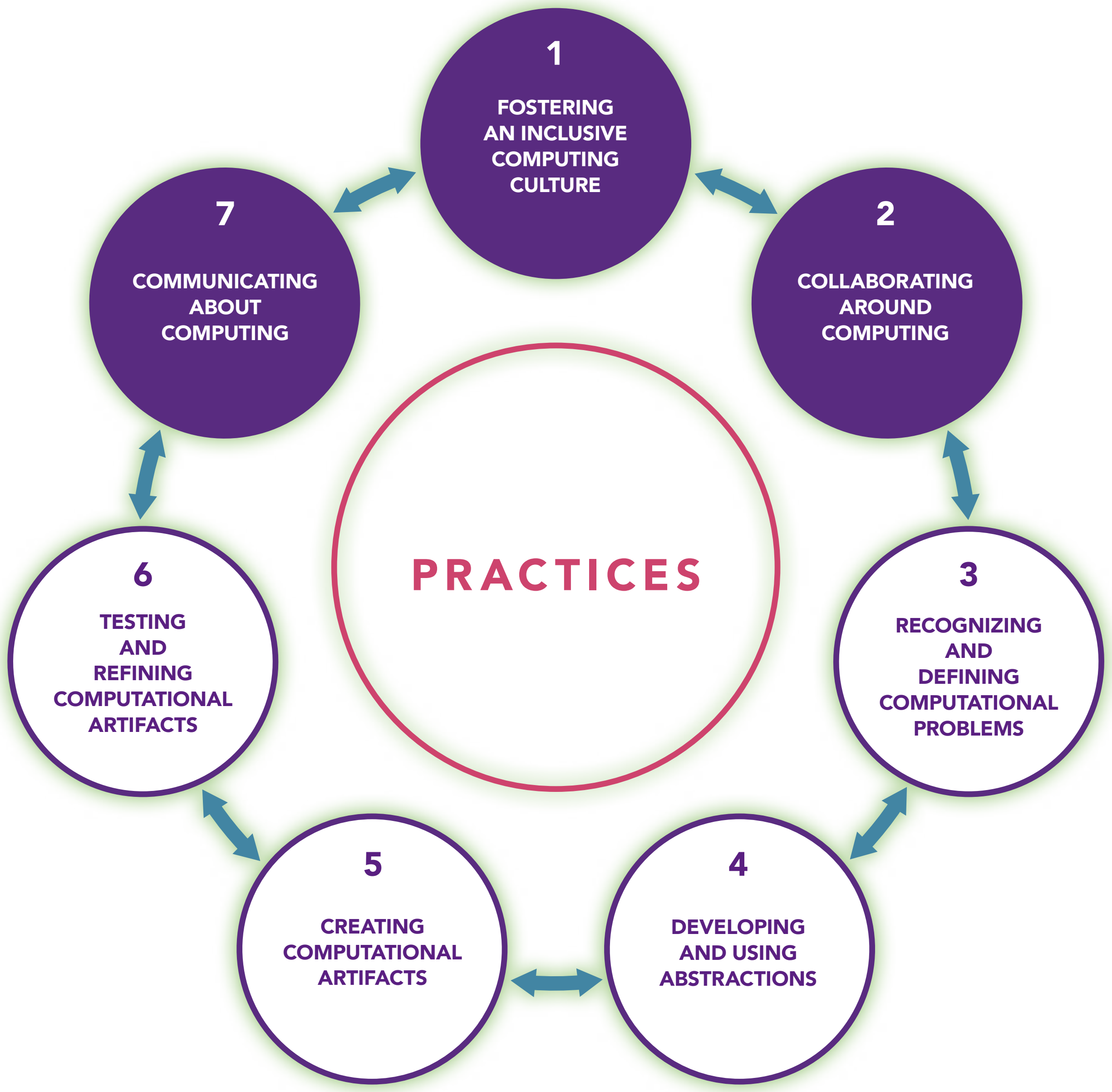
The seven core practices of computer science describe the behaviors and ways of thinking that computationally literate students use to fully engage in today's data-rich and interconnected world. The practices naturally integrate with one another and contain language that intentionally overlaps to illuminate the connections among them. They are displayed in an order that suggests a process for developing computational artifacts. This process is cyclical and can follow many paths; in the framework, it begins with recognizing diverse users and valuing others' perspectives and ends with communicating the results to broad audiences (see Figure 5.1).

Unlike the core conacepts, the practices are not delineated by grade bands. Rather, the practices use a narrative to describe how students should exhibit each practice with increasing sophistication from kindergarten to Grade 12. In addition to describing the progression, these narratives also provide some examples of the interrelatedness of the practice statements and the ways in which these statements build upon one another.

Computational Thinking

Computational thinking is at the heart of the computer science practices and is delineated by practices 3–6. Practices 1, 2, and 7 are independent, general practices in computer science that complement computational thinking.

Figure 5.1: Core practices including computational thinking


PRACTICES
1
FOSTERING AN INCLUSIVE COMPUTING CULTURE
2
COLLABORATING AROUND COMPUTING
3
RECOGNIZING AND DEFINING COMPUTATIONAL PROBLEMS
4
DEVELOPING AND USING ABSTRACTIONS
5
CREATING COMPUTATIONAL ARTIFACTS
6
TESTING AND REFINING COMPUTATIONAL ARTIFACTS
7
COMMUNICATING ABOUT COMPUTING

Defining Computational Thinking

Computational thinking refers to the thought processes involved in expressing solutions as computational steps or algorithms that can be carried out by a computer (Cuny, Snyder, & Wing, 2010; Aho, 2011; Lee, 2016). This definition draws on the idea of formulating problems and solutions in a form that can be carried out by an information-processing agent (Cuny, Snyder, & Wing, 2010) and the idea that the solutions should take the specific form of computational steps and algorithms to be executed by a computer (Aho, 2011; Lee, 2016). Computational thinking requires understanding the capabilities of computers, formulating problems to be addressed by a computer, and designing algorithms that a computer can execute. The most effective context and approach for developing computational thinking is learning computer science; they are intrinsically connected.

Computational thinking is essentially a problem-solving process that involves designing solutions that capitalize on the power of computers; this process begins before a single line of code is written. Computers provide benefits in terms of memory, speed, and accuracy of execution. Computers also require people to express their thinking in a formal structure, such as a programming language. Similar to writing notes on a piece of paper to "get your thoughts down," creating a program allows people to externalize their thoughts in a form that can be manipulated and scrutinized. Programming allows students to think about their thinking; by debugging a program, students debug their own thinking (Papert, 1980).

Computational thinking refers to the thought processes involved in expressing solutions as computational steps or algorithms that can be carried out by a computer.

Despite what the name implies, computational thinking is fundamentally a human ability. In other words, "[h]umans process information; humans compute" (Wing, 2008, p. 3718). This nuance is the basis for "unplugged" approaches to computer science (i.e., teaching computer science without computers) and explains how computational thinking can apply beyond the borders of computer science to a variety of disciplines, such as science, technology, engineering, and mathematics (STEM), but also the arts and humanities (Bundy, 2007).

Distinguishing Computational Thinking

The description of computational thinking in the K–12 Computer Science Framework extends beyond the general use of computers or technology in education to include specific skills such as designing algorithms, decomposing problems, and modeling phenomena. If computational thinking can take place without a computer, conversely, using a computer in class does not necessarily constitute computational thinking. For example, a student is not necessarily using computational thinking when he or she enters data into a spreadsheet and creates a chart. However, this action can include computational thinking if the student creates algorithms to automate the transformation of the data or to power an interactive data visualization.

A computational artifact must be distinguished by evaluating the process used to create it (i.e., computational thinking), in addition to the characteristics of the product itself. For example, the same digital animation may be the result of carefully constructing algorithms that control when characters move and how they interact or simply selecting characters and actions from a predesignated template. In this example, it is the process used to create the animation that defines whether it can be considered a computational artifact. The assessment of computational thinking can be improved by having students explain their decisions and development process (Brennan & Resnick, 2012).

Computer Science Practices and Other Subject Areas

The framework is grounded in the belief that computer science offers unique opportunities for developing computational thinking and that the framework's practices can be applied to other subjects beyond computer science. As Barr and Stephenson (2011) have noted, the "computer science education community can play an important role in highlighting algorithmic problem solving practices and applications of computing across disciplines, and help integrate the application of computational methods and tools across diverse areas of learning" (p. 49).

While computational thinking is a focus in computer science, it is also included in standards for other subjects. For example, computational thinking is explicitly referenced in the practices of many state science standards[1] and implicitly in state math standards.[2] Additionally, the recent revision to the International Society for Technology in Education Standards for Students (ISTE, 2016) describes computational thinking in a similar way as the framework. All of these documents share the vision that computational thinking is important for all students.

Computational thinking is a fundamental skill for everyone, not just for computer scientists. To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability (Wing, 2006, p. 33).

Figure 5.2 on the next page describes the intersection among practices in computer science, science and engineering, and mathematics. Explicit instruction is required to create the connections illustrated in the figure.

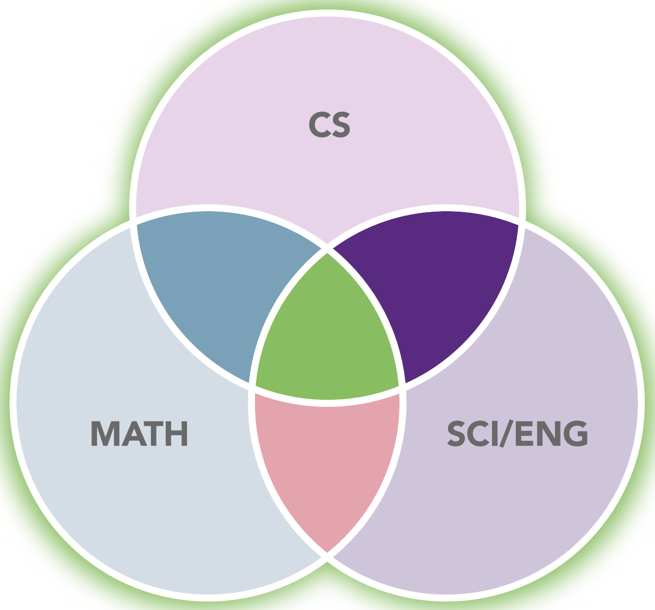
Acknowledgments

The writing team thanks the Computational Thinking Task Force of the Computer Science Teachers Association for its contribution to this section.

[1] Practice 5: Using Mathematics and Computational Thinking (Next Generation Science Standards Lead States, 2013).

[2] CCSS.Math.Practice.MP2: Reason abstractly and quantitatively (NGA Center for Best Practices & CCSSO, 2010).

Figure 5.2: Relationships between computer science, science and engineering, and math practices



🟦

CS + Math

- **Develop and use abstractions**
  M2. Reason abstractly and quantitatively
  M7. Look for and make use of structure
  M8. Look for and express regularity in repeated reasoning
  CS4. Developing and Using Abstractions

- **Use tools when collaborating**
  M5. Use appropriate tools strategically
  CS2. Collaborating Around Computing

- **Communicate precisely**
  M6. Attend to precision
  CS7. Communicating About Computing

🟪

CS + Sci/Eng

- **Communicate with data**
  S4. Analyze and interpret data
  CS7. Communicating About Computing

- **Create artifacts**
  S3. Plan and carry out investigations
  S6. Construct explanations and design solutions
  CS4. Developing and Using Abstractions
  CS5. Creating Computational Artifacts
  CS6. Testing and Refining Computational Artifacts

🟩

CS + Math + Sci/Eng

- **Model**
  S2. Develop and use models
  M4. Model with mathematics
  CS4. Developing and Using Abstractions
  CS6. Testing and Refining Computational Artifacts

- **Use computational thinking**
  S5. Use mathematics and computational thinking
  CS3. Recognizing and Defining Computational Problems
  CS4. Developing and Using Abstractions
  CS5. Creating Computational Artifacts

- **Define problems**
  S1. Ask questions and define problems
  M1. Make sense of problems and persevere in solving them
  CS3. Recognizing and Defining Computational Problems

- **Communicate rationale**
  S7. Engage in argument from evidence
  S8. Obtain, evaluate, and communicate information
  M3. Construct viable arguments and critique the reasoning of others
  CS7. Communicating About Computing

* Computer science practices also overlap with practices in other domains, including English language arts. For example, *CS1. Fostering an Inclusive Computing Culture* and *CS2. Collaborating Around Computing* overlap with *E7. Come to understand other perspectives and cultures through reading, listening, and collaborations.*

References

Aho, A.V. (2011, January) Computation and Computational Thinking. *ACM Ubiquity*, 1, 1-8.

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K–12: What is involved and what is the role of the computer science education community? *ACM Inroads, 2*, 48–54.

Brennan, K., & Resnick, M. (2012). *Using artifact-based interviews to study the development of computational thinking in interactive media design*. Paper presented at the annual meeting of the American Educational Research Association, Vancouver, BC, Canada.

Bundy, A. (2007). Computational thinking is pervasive. *Journal of Scientific and Practical Computing, 1*, 67–69.

Cuny, J., Snyder, L., & Wing, J.M. (2010). Demystifying computational thinking for non-computer scientists. Unpublished manuscript in progress. Retrieved from http://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf

International Society for Technology in Education. (2016). *ISTE standards for students*. Retrieved from https://www.iste.org/resources/product?id=3879&childProduct=3848

Lee, I. (2016). Reclaiming the roots of CT. *CSTA Voice: The Voice of K–12 Computer Science Education and Its Educators, 12*(1), 3–4. Retrieved from http://www.csteachers.org/resource/resmgr/Voice/csta_voice_03_2016.pdf

National Governors Association Center for Best Practices & Council of Chief State School Officers. (2010). *Common core state standards for mathematics*. Washington DC: Author.

Next Generation Science Standards Lead States. (2013). *Next generation science standards: For states, by states*. Washington, DC: The National Academies Press.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas.* NY: Basic Books.

Wing, J. M. (2006, March). Computational thinking. *Communications of the ACM, 49*(3), 33–35.

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society, 366*(1881), 3717–3725.

Practices

Practice 1. Fostering an Inclusive Computing Culture

Overview: Building an inclusive and diverse computing culture requires strategies for incorporating perspectives from people of different genders, ethnicities, and abilities. Incorporating these perspectives involves understanding the personal, ethical, social, economic, and cultural contexts in which people operate. Considering the needs of diverse users during the design process is essential to producing inclusive computational products.

By the end of Grade 12, students should be able to

1. **Include the unique perspectives of others** and reflect on one's own perspectives when designing and developing computational products.

   At all grade levels, students should recognize that the choices people make when they create artifacts are based on personal interests, experiences, and needs. Young learners should begin to differentiate their technology preferences from the technology preferences of others. Initially, students should be presented with perspectives from people with different backgrounds, ability levels, and points of view. As students progress, they should independently seek diverse perspectives throughout the design process for the purpose of improving their computational artifacts. Students who are well-versed in fostering an inclusive computing culture should be able to differentiate backgrounds and skillsets and know when to call upon others, such as to seek out knowledge about potential end users or intentionally seek input from people with diverse backgrounds.

2. **Address the needs of diverse end users** during the design process to produce artifacts with broad accessibility and usability.

   At any level, students should recognize that users of technology have different needs and preferences and that not everyone chooses to use, or is able to use, the same technology products. For example, young learners, with teacher guidance, might compare a touchpad and a mouse to examine differences in usability. As students progress, they should consider the preferences of people who might use their products. Students should be able to evaluate the accessibility of a product to a broad group of end users, such as people with various disabilities. For example, they may notice that allowing an end user to change font sizes and colors will make an interface usable for people with low vision. At the higher grades, students should become aware of professionally accepted accessibility standards and should be able to evaluate computational artifacts for accessibility. Students should also begin to identify potential bias during the design process to maximize accessibility in product design. For example, they can test an app and recommend to its designers that it respond to verbal commands to accommodate users who are blind or have physical disabilities.

3. **Employ self- and peer-advocacy** to address bias in interactions, product design, and development methods.

   After students have experience identifying diverse perspectives and including unique perspectives (P1.1), they should begin to employ self-advocacy strategies, such as speaking for themselves if their needs are not met. As students progress, they should advocate for their peers when accommodations, such as an assistive-technology peripheral device, are needed for someone to use a computational artifact. Eventually, students should regularly advocate for both themselves and others.

Practice 2. Collaborating Around Computing

Overview: Collaborative computing is the process of performing a computational task by working in pairs and on teams. Because it involves asking for the contributions and feedback of others, effective collaboration can lead to better outcomes than working independently. Collaboration requires individuals to navigate and incorporate diverse perspectives, conflicting ideas, disparate skills, and distinct personalities. Students should use collaborative tools to effectively work together and to create complex artifacts.

By the end of Grade 12, students should be able to

1. **Cultivate working relationships** with individuals possessing diverse perspectives, skills, and personalities.

   At any grade level, students should work collaboratively with others. Early on, they should learn strategies for working with team members who possess varying individual strengths. For example, with teacher support, students should begin to give each team member opportunities to contribute and to work with each other as co-learners. Eventually, students should become more sophisticated at applying strategies for mutual encouragement and support. They should express their ideas with logical reasoning and find ways to reconcile differences cooperatively. For example, when they disagree, they should ask others to explain their reasoning and work together to make respectful, mutual decisions. As they progress, students should use methods for giving all group members a chance to participate. Older students should strive to improve team efficiency and effectiveness by regularly evaluating group dynamics. They should use multiple strategies to make team dynamics more productive. For example, they can ask for the opinions of quieter team members, minimize interruptions by more talkative members, and give individuals credit for their ideas and their work.

2. **Create team norms, expectations, and equitable workloads** to increase efficiency and effectiveness.

   After students have had experience cultivating working relationships within teams (P2.1), they should gain experience working in particular team roles. Early on, teachers may help guide this process by providing collaborative structures. For example, students may take turns in different roles on the project, such as note taker, facilitator, or "driver" of the computer. As students progress, they should become less dependent on the teacher assigning roles and become more adept at assigning roles within their teams. For example, they should decide together how to take turns in different roles. Eventually, students should independently organize their own teams and create common goals, expectations, and equitable workloads. They should also manage project workflow using agendas and timelines and should evaluate workflow to identify areas for improvement.

3. **Solicit and incorporate feedback** from, and provide constructive feedback to, team members and other stakeholders.

   At any level, students should ask questions of others and listen to their opinions. Early on, with teacher scaffolding, students should seek help and share ideas to achieve a particular purpose. As they progress in school, students should provide and receive feedback related to computing in constructive ways. For example, pair programming is a collaborative process that promotes giving and receiving feedback. Older students should engage in active listening by using questioning skills and should respond empathetically to others. As they progress, students should be able to receive feedback from multiple peers and should be able to differentiate opinions. Eventually, students should seek contributors from various environments. These contributors may include end users, experts, or general audiences from online forums.

4. **Evaluate and select technological tools** that can be used to collaborate on a project.

   At any level, students should be able to use tools and methods for collaboration on a project. For example, in the early grades, students could collaboratively brainstorm by writing on a whiteboard. As students progress, they should use technological collaboration tools to manage teamwork, such as knowledge-sharing tools and online project spaces. They should also begin to make decisions about which tools would be best to use and when to use them. Eventually, students should use different collaborative tools and methods to solicit input from not only team members and classmates but also others, such as participants in online forums or local communities.

Practice 3. Recognizing and Defining Computational Problems

Overview: The ability to recognize appropriate and worthwhile opportunities to apply computation is a skill that develops over time and is central to computing. Solving a problem with a computational approach requires defining the problem, breaking it down into parts, and evaluating each part to determine whether a computational solution is appropriate.

By the end of Grade 12, students should be able to

1. **Identify complex, interdisciplinary, real-world problems** that can be solved computationally.

   At any level, students should be able to identify problems that have been solved computationally. For example, young students can discuss a technology that has changed the world, such as email or mobile phones. As they progress, they should ask clarifying questions to understand whether a problem or part of a problem can be solved using a computational approach. For example, before attempting to write an algorithm to sort a large list of names, students may ask questions about how the names are entered and what type of sorting is desired. Older students should identify more complex problems that involve multiple criteria and constraints. Eventually, students should be able to identify real-world problems that span multiple disciplines, such as increasing bike safety with new helmet technology, and can be solved computationally.

2. **Decompose complex real-world problems** into manageable subproblems that could integrate existing solutions or procedures.

   At any grade level, students should be able to break problems down into their component parts. In the early grade levels, students should focus on breaking down simple problems. For example, in a visual programming environment, students could break down (or decompose) the steps needed to draw a shape. As students progress, they should decompose larger problems into manageable smaller problems. For example, young students may think of an animation as multiple scenes and thus create each scene independently. Students can also break down a program into subgoals: getting input from the user, processing the data, and displaying the result to the user. Eventually, as students encounter complex real-world problems that span multiple disciplines or social systems, they should decompose complex problems into manageable subproblems that could potentially be solved with programs or procedures that already exist. For example, students could create an app to solve a community problem that connects to an online database through an application programming interface (API).

3. **Evaluate whether it is appropriate and feasible** to solve a problem computationally.

   After students have had some experience breaking problems down (P3.2) and identifying sub-problems that can be solved computationally (P3.1), they should begin to evaluate whether a computational solution is the most appropriate solution for a particular problem. For example, students might question whether using a computer to determine whether someone is telling the truth would be advantageous. As students progress, they should systematically evaluate the feasibility of using computational tools to solve given problems or subproblems, such as through a cost-benefit analysis. Eventually, students should include more factors in their evaluations, such as how efficiency affects feasibility or whether a proposed approach raises ethical concerns.

Practice 4. Developing and Using Abstractions

Overview: Abstractions are formed by identifying patterns and extracting common features from specific examples to create generalizations. Using generalized solutions and parts of solutions designed for broad reuse simplifies the development process by managing complexity.

By the end of Grade 12, students should be able to

1. **Extract common features** from a set of interrelated processes or complex phenomena.

   Students at all grade levels should be able to recognize patterns. Young learners should be able to identify and describe repeated sequences in data or code through analogy to visual patterns or physical sequences of objects. As they progress, students should identify patterns as opportunities for abstraction, such as recognizing repeated patterns of code that could be more efficiently implemented as a loop. Eventually, students should extract common features from more complex phenomena or processes. For example, students should be able to identify common features in multiple segments of code and substitute a single segment that uses variables to account for the differences. In a procedure, the variables would take the form of parameters. When working with data, students should be able to identify important aspects and find patterns in related data sets such as crop output, fertilization methods, and climate conditions.

2. **Evaluate existing technological functionalities** and **incorporate t**hem into new designs.

   At all levels, students should be able to use well-defined abstractions that hide complexity. Just as a car hides operating details, such as the mechanics of the engine, a computer program's "move" command relies on hidden details that cause an object to change location on the screen. As they progress, students should incorporate predefined functions into their designs, understanding that they do not need to know the underlying implementation details of the abstractions that they use.

Eventually, students should understand the advantages of, and be comfortable using, existing functionalities (abstractions) including technological resources created by other people, such as libraries and application programming interfaces (APIs). Students should be able to evaluate existing abstractions to determine which should be incorporated into designs and how they should be incorporated. For example, students could build powerful apps by incorporating existing services, such as online databases that return geolocation coordinates of street names or food nutrition information.

3. **Create modules** and **develop points of interaction** that can apply to multiple situations and reduce complexity.

   After students have had some experience identifying patterns (P4.1), decomposing problems (P3.2), using abstractions (P4.2), and taking advantage of existing resources (P4.2), they should begin to develop their own abstractions. As they progress, students should take advantage of opportunities to develop generalizable modules. For example, students could write more efficient programs by designing procedures that are used multiple times in the program. These procedures can be generalized by defining parameters that create different outputs for a wide range of inputs. Later on, students should be able to design systems of interacting modules, each with a well-defined role, that coordinate to accomplish a common goal. Within an object-oriented programming context, module design may include defining the interactions among objects. At this stage, these modules, which combine both data and procedures, can be designed and documented for reuse in other programs. Additionally, students can design points of interaction, such as a simple user interface, either text or graphical, that reduces the complexity of a solution and hides lower-level implementation details.

4. **Model phenomena and processes** and **simulate systems** to understand and evaluate potential outcomes.

   Students at all grade levels should be able to represent patterns, processes, or phenomena. With guidance, young students can draw pictures to describe a simple pattern, such as sunrise and sunset, or show the stages in a process, such as brushing your teeth. They can also create an animation to model a phenomenon, such as evaporation. As they progress, students should understand that computers can model real-world phenomena, and they should use existing computer simulations to learn about real-world systems. For example, they may use a preprogrammed model to explore how parameters affect a system, such as how rapidly a disease spreads. Older students should model phenomena as systems, with rules governing the interactions within the system. Students should analyze and evaluate these models against real-world observations. For example, students might create a simple producer–consumer ecosystem model using a programming tool. Eventually, they could progress to creating more complex and realistic interactions between species, such as predation, competition, or symbiosis, and evaluate the model based on data gathered from nature.

Practice 5. Creating Computational Artifacts

Overview: The process of developing computational artifacts embraces both creative expression and the exploration of ideas to create prototypes and solve computational problems. Students create artifacts that are personally relevant or beneficial to their community and beyond. Computational artifacts can be created by combining and modifying existing artifacts or by developing new artifacts. Examples of computational artifacts include programs, simulations, visualizations, digital animations, robotic systems, and apps.

By the end of Grade 12, students should be able to

1. **Plan the development** of a computational artifact using an iterative process that includes reflection on and modification of the plan, taking into account key features, time and resource constraints, and user expectations.

   At any grade level, students should participate in project planning and the creation of brainstorming documents. The youngest students may do so with the help of teachers. With scaffolding, students should gain greater independence and sophistication in the planning, design, and evaluation of artifacts. As learning progresses, students should systematically plan the development of a program or artifact and intentionally apply computational techniques, such as decomposition and abstraction, along with knowledge about existing approaches to artifact design. Students should be capable of reflecting on and, if necessary, modifying the plan to accommodate end goals.

2. **Create a computational artifact** for practical intent, personal expression, or to address a societal issue.

   Students at all grade levels should develop artifacts in response to a task or a computational problem. At the earliest grade levels, students should be able to choose from a set of given commands to create simple animated stories or solve pre-existing problems. Younger students should focus on artifacts of personal importance. As they progress, student expressions should become more complex and of increasingly broader significance. Eventually, students should engage in independent, systematic use of design processes to create artifacts that solve problems with social significance by seeking input from broad audiences.

3. **Modify an existing artifact** to improve or customize it.

   At all grade levels, students should be able to examine existing artifacts to understand what they do. As they progress, students should attempt to use existing solutions to accomplish a desired *goal. For example, students could attach a programmable light sensor to a physical artifact they have created to make it respond to light. Later on, they should modify or remix parts of existing programs to develop something new or to add more advanced features and complexity. For example, students could modify prewritten code from a single-player game to create a two-player game with slightly different rules.*

Practice 6. Testing and Refining Computational Artifacts

Overview: Testing and refinement is the deliberate and iterative process of improving a computational artifact. This process includes debugging (identifying and fixing errors) and comparing actual outcomes to intended outcomes. Students also respond to the changing needs and expectations of end users and improve the performance, reliability, usability, and accessibility of artifacts.

By the end of Grade 12, students should be able to

1. **Systematically test** computational artifacts by considering all scenarios and using test cases.

   At any grade level, students should be able to compare results to intended outcomes. Young students should verify whether given criteria and constraints have been met. As students progress, they should test computational artifacts by considering potential errors, such as what will happen if a user enters invalid input. Eventually, testing should become a deliberate process that is more iterative, systematic, and proactive. Older students should be able to anticipate errors and use that knowledge to drive development. For example, students can test their program with inputs associated with all potential scenarios.

2. **Identify and fix errors** using a systematic process.

   At any grade level, students should be able to identify and fix errors in programs (debugging) and use strategies to solve problems with computing systems (troubleshooting). Young students could use trial and error to fix simple errors. For example, a student may try reordering the sequence of commands in a program. In a hardware context, students could try to fix a device by resetting it or checking whether it is connected to a network. As students progress, they should become more adept at debugging programs and begin to consider logic errors: cases in which a program works, but not as desired. In this way, students will examine and correct their own thinking. For example, they might step through their program, line by line, to identify a loop that does not terminate as expected. Eventually, older students should progress to using more complex strategies for identifying and fixing errors, such as printing the value of a counter variable while a loop is running to determine how many times the loop runs.

3. **Evaluate and refine** a computational artifact multiple times to enhance its performance, reliability, usability, and accessibility.

   After students have gained experience testing (P6.2), debugging, and revising (P6.1), they should begin to evaluate and refine their computational artifacts. As students progress, the process of evaluation and refinement should focus on improving performance and reliability. For example, students could observe a robot in a variety of lighting conditions to determine that a light sensor should be less sensitive. Later on, evaluation and refinement should become an iterative process that also encompasses making artifacts more usable and accessible (P1.2). For example, students can incorporate feedback from a variety of end users to help guide the size and placement of menus and buttons in a user interface.

Practice 7. Communicating About Computing

Overview: Communication involves personal expression and exchanging ideas with others. In computer science, students communicate with diverse audiences about the use and effects of computation and the appropriateness of computational choices. Students write clear comments, document their work, and communicate their ideas through multiple forms of media. Clear communication includes using precise language and carefully considering possible audiences.

By the end of Grade 12, students should be able to

1. **Select, organize, and interpret** large data sets from multiple sources to support a claim.

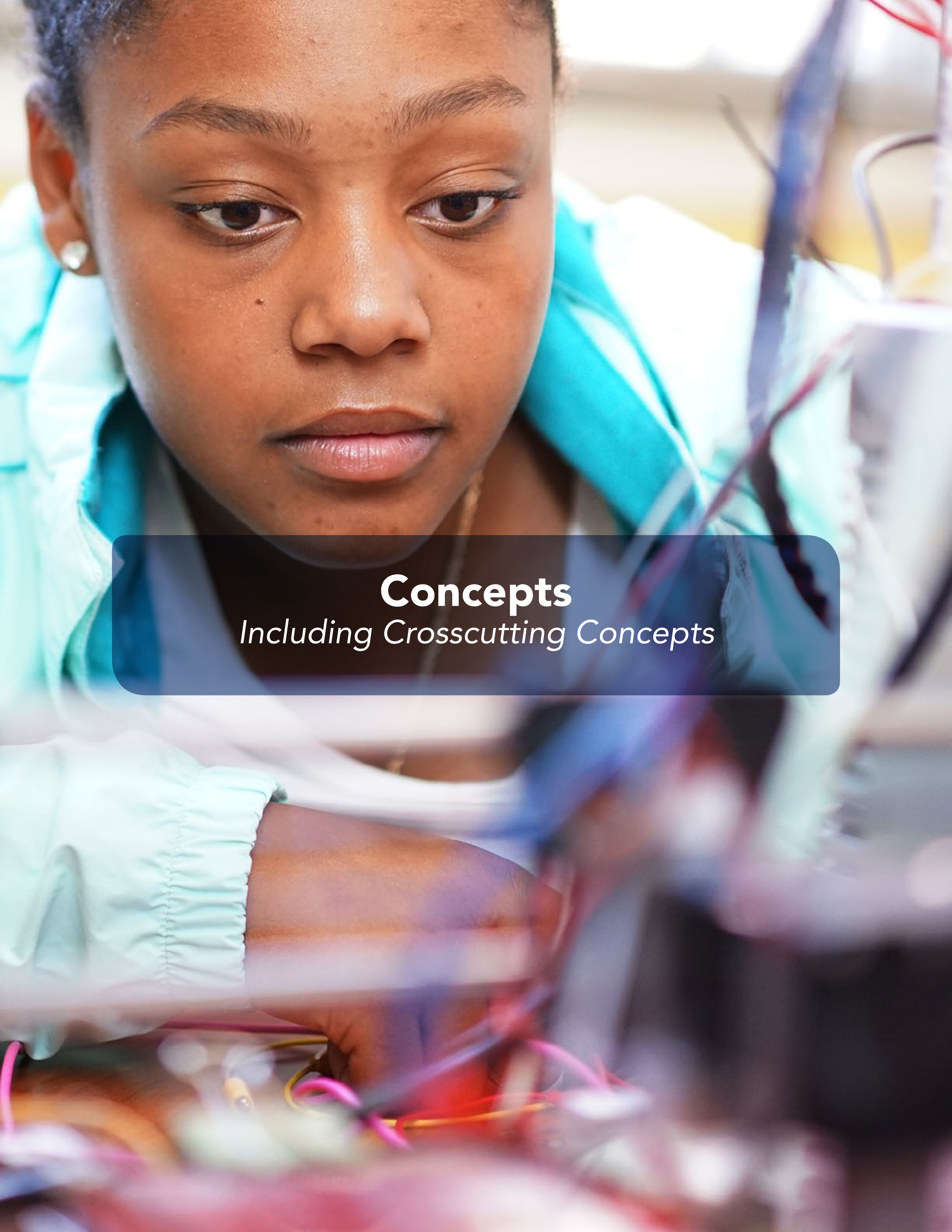
   At any grade level, students should be able to refer to data when communicating an idea. Early on, students should, with guidance, present basic data through the use of visual representations, such as storyboards, flowcharts, and graphs. As students progress, they should work with larger data sets and organize the data in those larger sets to make interpreting and communicating it to others easier, such as through the creation of basic data representations. Eventually, students should be able to select relevant data from large or complex data sets in support of a claim or to communicate the information in a more sophisticated manner.

2. **Describe, justify, and document** computational processes and solutions using appropriate terminology consistent with the intended audience and purpose.

   At any grade level, students should be able to talk about choices they make while designing a computational artifact. Early on, they should use language that articulates what they are doing and identifies devices and concepts they are using with correct terminology (e.g., program, input, and *debug). Younger students should identify the goals and expected outcomes of their solutions. Along the way, students should provide documentation for end users that explains their artifacts and how they function, and they should both give and receive feedback. For example, students could provide a project overview and ask for input from users. As students progress, they should incorporate clear comments in their product and document their process using text, graphics, presentations, and demonstrations.*

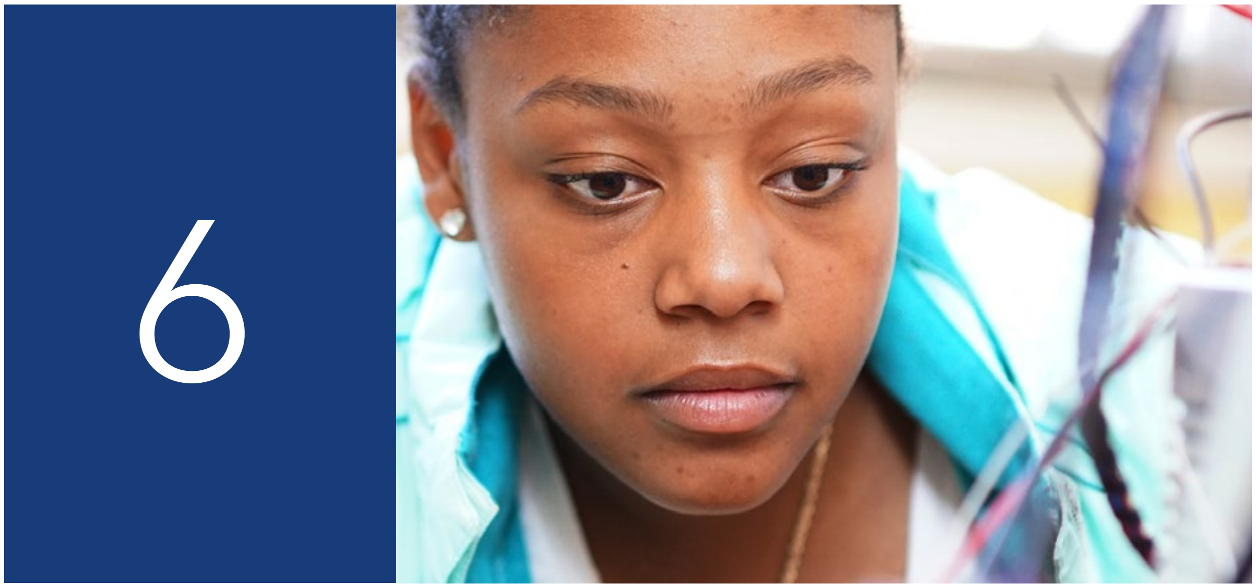
3. **Articulate ideas responsibly** by observing intellectual property rights and giving appropriate attribution.

   All students should be able to explain the concepts of ownership and sharing. Early on, students should apply these concepts to computational ideas and creations. They should identify instances of remixing, when ideas are borrowed and iterated upon, and give proper attribution. They should also recognize the contributions of collaborators. Eventually, students should consider common licenses that place limitations or restrictions on the use of computational artifacts. For example, a download-ed image may have restrictions that prohibit modification of an image or using it for commercial purposes.



Concepts

Including Crosscutting Concepts



6 Concepts *Including Crosscutting Concepts*

Preface

The core concepts of the K–12 Computer Science Framework represent major content areas in the field of computer science. The core concepts are delineated by multiple subconcepts that represent specific ideas within each concept. The learning progressions for each subconcept provide a thread connecting student learning from kindergarten to 12th grade.

Core concepts of the framework:

1. Computing Systems
2. Networks and the Internet
3. Data and Analysis
4. Algorithms and Programming
5. Impacts of Computing

Crosscutting concepts are themes that illustrate connections among different concept statements. They are integrated into concept statements, instead of existing as an independent dimension of the framework. The crosscutting concepts that are represented in each concept statement are noted in the statement's descriptive material.

Crosscutting concepts of the framework:

- Abstraction
- System Relationships
- Human–Computer Interaction
- Privacy and Security
- Communication and Coordination

Abstraction: An abstraction is the result of reducing a process or set of information to a set of important characteristics for computational use. Abstractions establish interactions at a level of reduced complexity by managing the more complex details below the level of interaction. An abstraction can be created to generalize a range of situations by picking out common properties minus specific implementation details.

System Relationships: The parts of a system are interdependent and organized for a common purpose. A systems perspective provides the opportunity to decompose complex problems into parts that are easier to understand, develop, fix, and maintain. General systems principles include feedback, control, efficiency, modularity, synthesis, emergence, and hierarchy.

Human–Computer Interaction: Humans interact directly with computers such as laptops and smartphones but also other devices, such as cars and home appliances, which have embedded computers. Developing effective and accessible user interfaces involves the integration of technical knowledge and social science and encompasses both designer and user perspectives.

Privacy and Security: Privacy is the ability to seclude information and express it selectively. It includes controls for the collection, access, use, storage, sharing, and alteration of information. Security refers to the safeguards surrounding information systems and includes protection from theft or damage to hardware, software, and the information in the systems. Security supports privacy.

Communication and Coordination: Processes in computing are characterized by the reliable exchange of information between autonomous agents (communication) that cooperate toward common outcomes that no agent could produce alone (coordination). Communication and coordination are distinct but not independent processes. What is special about computing is the scale at which communication and coordination work.

Core Concepts and Subconcepts Overviews

Computing Systems

Overview: People interact with a wide variety of computing devices that collect, store, analyze, and act upon information in ways that can affect human capabilities both positively and negatively. The physical components (hardware) and instructions (software) that make up a computing system communicate and process information in digital form. An understanding of hardware and software is useful when troubleshooting a computing system that does not work as intended.

Devices	**Overview**: Many everyday objects contain computational components that sense and act on the world. In early grades, students learn features and applications of common computing devices. As they progress, students learn about connected systems and how the interaction between humans and devices influences design decisions.
Hardware and Software	**Overview:** Computing systems use hardware and software to communicate and process information in digital form. In early grades, students learn how systems use both hardware and software to represent and process information. As they progress, students gain a deeper understanding of the interaction between hardware and software at multiple levels within computing systems.
Troubleshooting	**Overview:** When computing systems do not work as intended, troubleshooting strategies help people solve the problem. In early grades, students learn that identifying the problem is the first step to fixing it. As they progress, students learn systematic problem-solving processes and how to develop their own troubleshooting strategies based on a deeper understanding of how computing systems work.

Networks and the Internet

Overview: Computing devices typically do not operate in isolation. Networks connect computing devices to share information and resources and are an increasingly integral part of computing. Networks and communication systems provide greater connectivity in the computing world by providing fast, secure communication and facilitating innovation.

Network Communication and Organization	**Overview:** Computing devices communicate with each other across networks to share information. In early grades, students learn that computers connect them to other people, places, and things around the world. As they progress, students gain a deeper understanding of how information is sent and received across different types of networks.
Cybersecurity	**Overview:** Transmitting information securely across networks requires appropriate protection. In early grades, students learn how to protect their personal information. As they progress, students learn increasingly complex ways to protect information sent across networks.

Data and Analysis

Overview: Computing systems exist to process data. The amount of digital data generated in the world is rapidly expanding, so the need to process data effectively is increasingly important. Data is collected and stored so that it can be analyzed to better understand the world and make more accurate predictions.

Collection	**Overview:** Data is collected with both computational and noncomputational tools and processes. In early grades, students learn how data about themselves and their world is collected and used. As they progress, students learn the effects of collecting data with computational and automated tools.
Storage	**Overview:** Core functions of computers are storing, representing, and retrieving data. In early grades, students learn how data is stored on computers. As they progress, students learn how to evaluate different storage methods, including the tradeoffs associated with those methods.
Visualization and Transformation	**Overview:** Data is transformed throughout the process of collection, digital representation, and analysis. In early grades, students learn how transformations can be used to simplify data. As they progress, students learn about more complex operations to discover patterns and trends and communicate them to others.
Inference and Models	**Overview:** Data science is one example where computer science serves many fields. Computer science and science use data to make inferences, theories, or predictions based upon the data collected from users or simulations. In early grades, students learn about the use of data to make simple predictions. As they progress, students learn how models and simulations can be used to examine theories and understand systems and how predictions and inferences are affected by more complex and larger data sets.

Algorithms and Programming

Overview: An algorithm is a sequence of steps designed to accomplish a specific task. Algorithms are translated into programs, or code, to provide instructions for computing devices. Algorithms and programming control all computing systems, empowering people to communicate with the world in new ways and solve compelling problems. The development process to create meaningful and efficient programs involves choosing which information to use and how to process and store it, breaking apart large problems into smaller ones, recombining existing solutions, and analyzing different solutions.

Algorithms	**Overview:** Algorithms are designed to be carried out by both humans and computers. In early grades, students learn about age-appropriate algorithms from the real world. As they progress, students learn about the development, combination, and decomposition of algorithms, as well as the evaluation of competing algorithms.
Variables	**Overview:** Computer programs store and manipulate data using variables. In early grades, students learn that different types of data, such as words, numbers, or pictures, can be used in different ways. As they progress, students learn about variables and ways to organize large collections of data into data structures of increasing complexity.
Control	**Overview:** Control structures specify the order in which instructions are executed within an algorithm or program. In early grades, students learn about sequential execution and simple control structures. As they progress, students expand their understanding to combinations of structures that support complex execution.
Modularity	**Overview:** Modularity involves breaking down tasks into simpler tasks and combining simple tasks to create something more complex. In early grades, students learn that algorithms and programs can be designed by breaking tasks into smaller parts and recombining existing solutions. As they progress, students learn about recognizing patterns to make use of general, reusable solutions for commonly occurring scenarios and clearly describing tasks in ways that are widely usable.
Program Development	**Overview:** Programs are developed through a design process that is often repeated until the programmer is satisfied with the solution. In early grades, students learn how and why people develop programs. As they progress, students learn about the tradeoffs in program design associated with complex decisions involving user constraints, efficiency, ethics, and testing.

Impacts of Computing

Overview: Computing affects many aspects of the world in both positive and negative ways at local, national, and global levels. Individuals and communities influence computing through their behaviors and cultural and social interactions, and in turn, computing influences new cultural practices. An informed and responsible person should understand the social implications of the digital world, including equity and access to computing.

Culture	**Overview:** Computing influences culture—including belief systems, language, relationships, technology, and institutions—and culture shapes how people engage with and access computing. In early grades, students learn how computing can be helpful and harmful. As they progress, students learn about tradeoffs associated with computing and potential future impacts of computing on global societies.
Social Interactions	**Overview:** Computing can support new ways of connecting people, communicating information, and expressing ideas. In early grades, students learn that computing can connect people and support interpersonal communication. As they progress, students learn how the social nature of computing affects institutions and careers in various sectors.
Safety, Law, and Ethics	**Overview:** Legal and ethical considerations of using computing devices influence behaviors that can affect the safety and security of individuals. In early grades, students learn the fundamentals of digital citizenship and appropriate use of digital media. As they progress, students learn about the legal and ethical issues that shape computing practices.

Concepts

By the end of Grade 2

Computing Systems	
DEVICES	People use computing devices to perform a variety of tasks accurately and quickly. Computing devices interpret and follow the instructions they are given literally. *Computing devices can be used to do a number of things, such as play music, create documents, and send pictures. Computing devices are also very precise. For example, computers can perform multiple complex calculations much faster and with greater accuracy than people. While people may diverge from instructions given to them, computers will follow instructions exactly as they are given, even if they do not achieve the intended result.* ***Crosscutting Concept:*** *Human–Computer Interaction* ***Connections Within Framework:*** *K–2.Algorithms and Programming.Control; K–2. Algorithms and Programming.Modularity; 3–5.Algorithms and Programming.Control*
HARDWARE AND SOFTWARE	A computing system is composed of hardware and software. Hardware consists of physical components, while software provides instructions for the system. These instructions are represented in a form that a computer can understand. *Examples of hardware include screens to display information and buttons, keys, or dials to enter information. Software applications are programs with specific purposes, such as a web browser or game. A person may use a mouse (hard-ware) to click on a button displayed in a web browser (software) to navigate to a new web page. Computing systems convert instructions, such as "print," "save," or "crop," into a special language that the computer can understand. At this level, understanding that computer information is encoded is appropriate, but the explicit understanding of "bits" is reserved for later grade levels.* ***Crosscutting Concept:*** *Communication and Coordination* ***Connections Within Framework:*** *K–2.Algorithms and Programming. Algorithms; K–2.Algorithms and Programming.Control*
TROUBLESHOOTING	Computing systems might not work as expected because of hardware or software problems. Clearly describing a problem is the first step toward finding a solution. *Problems with computing systems have different causes, such as hardware settings, programming errors, or faulty connections to other devices. Developmentally appropriate ways to solve problems include debugging simple programs and seeking help by clearly describing a problem (for example, "The computer won't turn on," "The pointer on the screen won't move," or "I lost the web page.") Knowing how to diagnose or troubleshoot a problem with a computing system is not expected.* ***Crosscutting Concept:*** *System Relationships* ***Connection Within Framework:*** *3–5.Algorithms and Programming.Program Development*

By the end of Grade 2: continued from previous page

Networks and the Internet	
NETWORK COMMUNICATION AND ORGANIZATION	Computer networks can be used to connect people to other people, places, information, and ideas. The Internet enables people to connect with others worldwide through many different points of connection. *Small, wireless devices, such as cell phones, communicate with one another through a series of intermediary connection points, such as cellular towers. This coordination among many computing devices allows a person to voice call a friend or video chat with a family member. Details about the connection points are not expected at this level.* ***Crosscutting Concepts:*** *Communication and Coordination; Human–Computer Interaction* ***Connections Within Framework:*** *K–2.Impacts of Computing.Social Interactions; K–2.Data and Analysis.Collection; 3–5.Impacts of Computing.Social Interactions*
CYBERSECURITY	Connecting devices to a network or the Internet provides great benefit, care must be taken to use authentication measures, such as strong passwords, to protect devices and information from unauthorized access. *Authentication is the ability to verify the identity of a person or entity. Usernames and passwords, such as those on computing devices or Wi-Fi networks, provide a way of authenticating a user's identity. Because computers make guessing weak passwords easy, strong passwords have characteristics that make them more time-intensive to break.* ***Crosscutting Concepts:*** *Privacy and Security; Communication and Coordination* ***Connection Within Framework:*** *K–2.Impacts of Computing.Safety, Law, and Ethics*

By the end of Grade 2: continued from previous page

Data and Analysis	
COLLECTION	Everyday digital devices collect and display data over time. The collection and use of data about individuals and the world around them is a routine part of life and influences how people live. *Many everyday objects, such as cell phones, digital toys, and cars, can contain tools (such as sensors) and computers to collect and display data from their surroundings.* ***Crosscutting Concept:*** *Human–Computer Interaction* ***Connection Within Framework:*** *K–2.Networks and the Internet.Network Communication and Organization*
STORAGE	Computers store data that can be retrieved later. Identical copies of data can be made and stored in multiple locations for a variety of reasons, such as to protect against loss. *For example, pictures can be stored on a cell phone and viewed later, or progress in a game can be saved and continued later. The advantage of recording data digitally, such as in images or a spreadsheet, versus on a physical space, such as a whiteboard, is that old data (states of data collected over time) can be easily retrieved, copied, and stored in multiple places. This is why personal information put online can persist for a long time. Understanding local versus online storage is not expected at this level.* ***Crosscutting Concepts:*** *System Relationships; Privacy and Security* ***Connections Within Framework:*** *K–2.Impacts of Computing.Social Interactions; K–2.Algorithms and Programming.Variables*
VISUALIZATION AND TRANSFORMATION	Data can be displayed for communication in many ways. People use computers to transform data into new forms, such as graphs and charts. *Examples of displays include picture graphs, bar charts, or histograms. A data table that records a tally of students' favorite colors can be displayed as a chart on a computer.* ***Crosscutting Concept:*** *Abstraction* ***Connection Within Framework:*** *K–2.Impacts of Computing.Social Interactions*
INFERENCE AND MODELS	Data can be used to make inferences or predictions about the world. Inferences, statements about something that cannot be readily observed, are often based on observed data. Predictions, statements about future events, are based on patterns in data and can be made by looking at data visualizations, such as charts and graphs. *Observations of people's clothing (jackets and coats) can be used to make an inference about the weather (it is cold outside). Patterns in past data can be identified and extrapolated to make predictions. For example, a person's lunch menu selection can be predicted by using data on past lunch selections.* ***Crosscutting Concept:*** *Abstraction* ***Connection Within Framework:*** *K–2.Impacts of Computing.Culture*

By the end of Grade 2: continued from previous page

Algorithms and Programming	
ALGORITHMS	People follow and create processes as part of daily life. Many of these processes can be expressed as algorithms that computers can follow. *Routines, such as morning meeting, clean-up time, and dismissal, are examples of algorithms that are common in many early elementary classrooms. Other examples of algorithms include making simple foods, navigating a classroom, and daily routines like brushing teeth. Just as people use algorithms to complete daily routines, they can program computers to use algorithms to complete different tasks. Algorithms are commonly implemented using a precise language that computers can interpret.* ***Crosscutting Concept:*** *Abstraction* ***Connection Within Framework:*** *K–2.Computing Systems.Hardware and Software*
VARIABLES	Information in the real world can be represented in computer programs. Programs store and manipulate data, such as numbers, words, colors, and images. The type of data determines the actions and attributes associated with it. *Different actions are available for different kinds of information. For example, sprites (character images) can be moved and turned, numbers can be added or subtracted, and pictures can be recolored or cropped.* ***Crosscutting Concept:*** *Abstraction* ***Connection Within Framework:*** *K–2.Data and Analysis.Storage*
CONTROL	Computers follow precise sequences of instructions that automate tasks. Program execution can also be nonsequential by repeating patterns of instructions and using events to initiate instructions. *Computers follow instructions literally. Examples of sequences of instructions include steps for drawing a shape or moving a character across the screen. An event, such as the press of a button, can trigger an action. Simple loops can be used to repeat instructions. At this level, distinguishing different types of loops is not expected.* ***Crosscutting Concept:*** *Abstraction* ***Connection Within Framework:*** *K–2.Data and Analysis.Storage*

By the end of Grade 2: continued from previous page

MODULARITY	Complex tasks can be broken down into simpler instructions, some of which can be broken down even further. Likewise, instructions can be combined to accomplish complex tasks. *Decomposition is the act of breaking down tasks into simpler tasks. An example of decomposition is preparing for a party: it involves inviting guests, making food, and setting the table. These tasks can be broken down further. For example, setting the table involves laying a tablecloth, folding napkins, and placing utensils and plates on the table. Another example is breaking down the steps to draw a polygon.* *Composition, on the other hand, is the combination of smaller tasks into more complex tasks. To build a city, people build several houses, a school, a store, etc. To create a group art project, people can paint or draw their favorite ocean animal, then combine them to create an ecosystem.* ***Crosscutting Concept:*** *System Relationships* ***Connection Within Framework:*** *K–2.Computing Systems.Devices*
PROGRAM DEVELOPMENT	People develop programs collaboratively and for a purpose, such as expressing ideas or addressing problems. *People work together to plan, create, and test programs within a context that is relevant to the programmer and users. Programming is used as a tool to create products that reflect a wide range of interests, such as video games, interactive art projects, and digital stories.* ***Crosscutting Concept:*** *Human–Computer Interaction* ***Connection Within Framework:*** *3–5.Impacts of Computing.Culture*

By the end of Grade 2: continued from previous page

Impacts of Computing	
CULTURE	Computing technology has positively and negatively changed the way people live and work. Computing devices can be used for entertainment and as productivity tools, and they can affect relationships and lifestyles. *Computing devices, such as fitness trackers, can motivate a more active lifestyle by monitoring physical activity. On the other hand, passively consuming media from computing devices may lead to a more sedentary lifestyle. In the past, the most popular form of communication was to send mail via the postal service. Now, more people send emails or text messages.* ***Crosscutting Concept:*** *Human–Computer Interaction* ***Connection Within Framework:*** *K–2.Data and Analysis.Inference and Models*
SOCIAL INTERACTIONS	Computing has positively and negatively changed the way people communicate. People can have access to information and each other instantly, anywhere, and at any time, but they are at the risk of cyberbullying and reduced privacy. *Online communication facilitates positive interactions, such as sharing ideas with many people, but the public and anonymous nature of online communication also allows intimidating and inappropriate behavior in the form of cyberbullying. Privacy should be considered when posting information online; such information can persist for a long time and be accessed by others, even unintended viewers.* ***Crosscutting Concepts:*** *Human–Computer Interaction; Privacy and Security* ***Connections Within Framework:*** *K–2.Data and Analysis.Storage; K–2.Data and Analysis.Visualization and Transformation*
SAFETY, LAW, AND ETHICS	People use computing technology in ways that can help or hurt themselves or others. Harmful behaviors, such as sharing private information and interacting with strangers, should be recognized and avoided. *Using computers comes with a level of responsibility, such as not sharing login information, keeping passwords private, and logging off when finished. Rules guiding interactions in the world, such as "stranger danger," apply to online environments as well.* ***Crosscutting Concept:*** *Privacy and Security* ***Connection Within Framework:*** *K–2.Networks and the Internet.Cybersecurity*

By the end of Grade 5

Computing Systems	
DEVICES	Computing devices may be connected to other devices or components to extend their capabilities, such as sensing and sending information. Connections can take many forms, such as physical or wireless. Together, devices and components form a system of interdependent parts that interact for a common purpose. *Computing devices often depend on other devices or components. For example, a robot depends on a physically attached light sensor to detect changes in brightness, whereas the light sensor depends on the robot for power. A smartphone can use wirelessly connected headphones to send audio information, and the headphones are useless without a music source.* ***Crosscutting Concepts:** Communication and Coordination; System Relationships* ***Connection Within Framework:** 3–5.Networks and the Internet.Network Communication and Organization*
HARDWARE AND SOFTWARE	Hardware and software work together as a system to accomplish tasks, such as sending, receiving, processing, and storing units of information as bits. Bits serve as the basic unit of data in computing systems and can represent a variety of information. *For example, a photo filter application (software) works with a camera (hardware) to produce a variety of effects that change the appearance of an image. This image is transmitted and stored as bits, or binary digits, which are commonly represented as 0s and 1s. All information, including instructions, is encoded as bits. Knowledge of the inner workings of hardware and software, number systems such as binary or hexadecimal, and how bits are represented in physical media are not priorities at this level.* ***Crosscutting Concepts:** Communication and Coordination; Abstraction* ***Connection Within Framework**: 3–5.Data and Analysis.Storage*
TROUBLESHOOTING	Computing systems share similarities, such as the use of power, data, and memory. Common troubleshooting strategies, such as checking that power is available, checking that physical and wireless connections are working, and clearing out the working memory by restarting programs or devices, are effective for many systems. *Although computing systems may vary, common troubleshooting strategies can be used on them, such as checking connections and power or swapping a working part in place of a potentially defective part. Rebooting a machine is commonly effective because it resets the computer. Because computing devices are composed of an interconnected system of hardware and software, troubleshooting strategies may need to address both.* ***Crosscutting Concepts:** System Relationships; Abstraction* ***Connection Within Framework:** 3–5.Networks and the Internet.Network Communication and Organization*

By the end of Grade 5: continued from previous page

Networks and the Internet	
NETWORK COMMUNICATION AND ORGANIZATION	Information needs a physical or wireless path to travel to be sent and received, and some paths are better than others. Information is broken into smaller pieces, called packets, that are sent independently and reassembled at the destination. Routers and switches are used to properly send packets across paths to their destinations. *There are physical paths for communicating information, such as ethernet cables, and wireless paths, such as Wi-Fi. Often, information travels on a combination of physical and wireless paths; for example, wireless paths originate from a physical connection point. The choice of device and type of connection will affect the path information travels and the potential bandwidth (the capacity to transmit data or bits in a given timeframe). Packets and packet switching (the method used to send packets) are the foundation for further understanding of Internet concepts. At this level, the priority is understanding the flow of information, rather than details of how routers and switches work and how to compare paths.* ***Crosscutting Concept:*** *Communication and Coordination* ***Connections Within Framework:*** *3–5.Computing Systems.Devices; 3–5.Computing Systems.Troubleshooting*
CYBERSECURITY	Information can be protected using various security measures. These measures can be physical and/or digital. *An offline backup of data is useful in case of an online security breach. A variety of software applications can monitor and address viruses and malware and alert users to their presence. Security measures can be applied to a network or individual devices on a network. Confidentiality refers to the protection of information from disclosure to unauthorized individuals, systems, or entities.* ***Crosscutting Concept:*** *Privacy and Security* ***Connection Within Framework:*** *3–5.Impacts of Computing.Safety, Law, and Ethics*

By the end of Grade 5: continued from previous page

Data and Analysis	
COLLECTION	People select digital tools for the collection of data based on what is being observed and how the data will be used. For example, a digital thermometer is used to measure temperature and a GPS sensor is used to track locations. *There is a wide array of digital data collection tools; however, only some are appropriate for certain types of data. Tools are chosen based upon the type of measurement they use as well as the type of data people wish to observe. Data scientists use the term observation to describe data collection, whether or not a human is involved in the collection.* ***Crosscutting Concept:** Abstraction* ***Connections Within Framework:** 3–5.Algorithms and Programming.Variables; 3–5.Algorithms and Programming.Algorithms*
STORAGE	Different software tools used to access data may store the data differently. The type of data being stored and the level of detail represented by that data affect the storage requirements. *Music, images, video, and text require different amounts of storage. Video will often require more storage than music or images alone because video combines both. For example, two pictures of the same object can require different amounts of storage based upon their resolution. Different software tools used to access and store data may add additional data about the data (metadata), which results in different storage requirements. An image file is a designed representation of a real-world image and can be opened by either an image editor or a text editor, but the text editor does not know how to translate the data into the image. Understanding binary or 8-bit versus 16-bit representations is not expected at this level.* ***Crosscutting Concept:** System Relationships* ***Connections Within Framework:** 3–5.Computing Systems.Hardware and Software; 3–5.Algorithms and Programming.Variables*
VISUALIZATION AND TRANSFORMATION	People select aspects and subsets of data to be transformed, organized, clustered, and categorized to provide different views and communicate insights gained from the data. *Data is often sorted or grouped to provide additional clarity. Data points can be clustered by a number of commonalities without a category label. For example, a series of days might be grouped by temperature, air pressure, and humidity and later categorized as fair, mild, or extreme weather. The same data could be manipulated in different ways to emphasize particular aspects or parts of the data set. For example, when working with a data set of popular songs, data could be shown by genre or artist. Simple data visualizations include graphs and charts, infographics, and ratios that represent statistical characteristics of the data.* ***Crosscutting Concepts:** Abstraction; Human–Computer Interaction* ***Connection Within Framework:** 6–8.Impacts of Computing.Social Interactions*

By the end of Grade 5: continued from previous page

INFERENCE AND MODELS	The accuracy of inferences and predictions is related to how realistically data is represented. Many factors influence the accuracy of inferences and predictions, such as the amount and relevance of data collected. *People use data to highlight or propose cause-and-effect relationships and predict outcomes. Basing inferences or predictions on data does not guarantee their accuracy; the data must be relevant and of sufficient quantity. An example of irrelevance is using eye color data when inferring someone's age. An example of insufficient quantity is predicting the outcome of an election by polling only a few people.* ***Crosscutting Concept:** System Relationships*

By the end of Grade 5: continued from previous page

Algorithms and Programming	
ALGORITHMS	Different algorithms can achieve the same result. Some algorithms are more appropriate for a specific context than others. *Different algorithms can be used to tie shoes or decide which path to take on the way home from school. While the end results may be similar, they may not be the same: in the example of going home, some paths could be faster, slower, or more direct, depending on varying factors, such as available time or the presence of obstacles (for example, a barking dog). Algorithms can be expressed in noncomputer languages, including natural language, flowcharts, and pseudocode.* ***Crosscutting Concept:*** *Abstraction* ***Connection Within Framework:*** *3–5.Data and Analysis.Collection*
VARIABLES	Programming languages provide variables, which are used to store and modify data. The data type determines the values and operations that can be performed on that data. *Variables are the vehicle through which computer programs store different types of data. At this level, understanding how to use variables is sufficient, without a fuller understanding of the technical aspects of variables (such as identifiers and memory locations). Data types vary by programming language, but many have types for numbers and text. Examples of operations associated with those types are multiplying numbers and combining text. Some visual, block-based languages do not have explicitly declared types but still have certain operations that apply only to particular types of data in a program.* ***Crosscutting Concept:*** *Abstraction* ***Connection Within Framework:*** *3–5.Data and Analysis.Storage*
CONTROL	Control structures, including loops, event handlers, and conditionals, are used to specify the flow of execution. Conditionals selectively execute or skip instructions under different conditions. *Different types of loops are used to repeat instructions in multiple ways depending on the situation. Examples of events include mouse clicks, typing on the keyboard, and collisions between objects. Event handlers are sets of commands that are tied to specific events. Conditionals represent decisions and are composed of a Boolean condition that specifies actions based on whether the condition evaluates to true or false. Boolean logic and operators (e.g., AND, OR, NOT) can be used to specify the appropriate groups of instructions to execute under various conditions.* ***Crosscutting Concepts:*** *Abstraction; Communication and Coordination* ***Connection Within Framework:*** *K–2.Computing Systems.Devices*

By the end of Grade 5: continued from previous page

MODULARITY	Programs can be broken down into smaller parts to facilitate their design, implementation, and review. Programs can also be created by incorporating smaller portions of programs that have already been created. *Decomposition facilitates aspects of program development, such as testing, by allowing people to focus on one piece at a time. Decomposition also enables different people to work on different parts at the same time. An example of decomposition at this level is creating an animation by separating a story into different scenes. For each scene, a background needs to be selected, characters placed, and actions programmed. The instructions required to program each scene may be similar to instructions in other programs.* ***Crosscutting Concepts:*** *System Relationships; Abstraction*
PROGRAM DEVELOPMENT	People develop programs using an iterative process involving design, implementation, and review. Design often involves reusing existing code or remixing other programs within a community. People continuously review whether programs work as expected, and they fix, or debug, parts that do not. Repeating these steps enables people to refine and improve programs. *Design, implementation, and review can be further broken down into additional stages and may have different labels. The design stage occurs before writing code. This is a planning stage in which the programmers gather information about the problem and sketch out a solution. During the implementation stage, the planned design is expressed in a programming language (code) that can be made to run on a computing device. During the review stage, the design and implementation are checked for adherence to program requirements, correctness, and usability. This review could lead to changes in implementation and possibly design, which demonstrates the iterative nature of the process. A community is created by people who share and provide feedback on one another's creations.* ***Crosscutting Concepts:*** *Human–Computer Interaction; System Relationships* ***Connection Within Framework:*** *K–2.Computing Systems.Troubleshooting*

By the end of Grade 5: continued from previous page

Impacts of Computing	
CULTURE	The development and modification of computing technology is driven by people's needs and wants and can affect groups differently. Computing technologies influence, and are influenced by, cultural practices. *New computing technology is created and existing technologies are modified to increase their benefits (for example, Internet search recommendations), decrease their risks (for example, autonomous cars), and meet societal demands (for example, smartphone apps). Increased Internet access and speed have allowed people to share cultural information but have also affected the practice of traditional cultural customs.* ***Crosscutting Concepts:** Human–Computer Interaction; System Relationships* ***Connections Within Framework:** K–2.Algorithms and Programming.Program Development; 6–8.Computing Systems.Devices; 6–8.Algorithms and Programming.Program Development*
SOCIAL INTERACTIONS	Computing technology allows for local and global collaboration. By facilitating communication and innovation, computing influences many social institutions such as family, education, religion, and the economy. *People can work in different places and at different times to collaborate and share ideas when they use technologies that reach across the globe. These social interactions affect how local and global groups interact with each other, and alternatively, these interactions can change the nature of groups. For example, a class can discuss ideas in the same school or in another nation through interactive webinars.* ***Crosscutting Concepts:** System Relationships; Human–Computer Interaction* ***Connection Within Framework:** K–2.Networks and the Internet.Network Communication and Organization*
SAFETY, LAW, AND ETHICS	Ethical complications arise from the opportunities provided by computing. The ease of sending and receiving copies of media on the Internet, such as video, photos, and music, creates the opportunity for unauthorized use, such as online piracy, and disregard of copyrights, such as lack of attribution. *Online piracy, the illegal copying of materials, is facilitated by the ability to make identical-quality copies of digital media with little effort. Other topics related to copyright are plagiarism, fair use, and properly citing online sources. Knowledge of specific copyright laws is not an expectation at this level.* ***Crosscutting Concepts:** System Relationships; Privacy and Security* ***Connection Within Framework:** 3–5.Networks and the Internet.Cybersecurity*

By the end of Grade 8

Computing Systems	
DEVICES	The interaction between humans and computing devices presents advantages, disadvantages, and unintended consequences. The study of human–computer interaction can improve the design of devices and extend the abilities of humans. *Accessibility is an important consideration in the design of any computing system. For example, assistive devices provide capabilities such as scanning written information and converting it to speech.The use of computing devices also has potential consequences, such as in the areas of privacy and security. For example, GPS-enabled smartphones can provide directions to a destination yet unintentionally allow a person to be tracked for malicious purposes. Also, the attention required to follow GPS directions can lead to accidents due to distracted driving.* ***Crosscutting Concepts:*** *Human–Computer Interaction; Privacy and Security* ***Connection Within Framework:*** *3–5.Impacts of Computing.Culture*
HARDWARE AND SOFTWARE	Hardware and software determine a computing system's capability to store and process information. The design or selection of a computing system involves multiple considerations and potential tradeoffs, such as functionality, cost, size, speed, accessibility, and aesthetics. *The capability of a computing system is determined by the processor speed, storage capacity, and data transmission speed, as well as other factors. Selecting one computing system over another involves balancing a number of tradeoffs. For example, selecting a faster computer with more memory involves the tradeoffs of speed and cost. Choosing one operating system over another involves the tradeoff of capability and compatibility, such as which apps can be installed or which devices can be connected. Designing a robot requires choosing both hardware and software and may involve a tradeoff between the potential for customization and ease of use. The use of a device that connects wirelessly through a Bluetooth connection versus a device that connects physically through a USB connection involves a tradeoff between mobility and the need for an additional power source for the wireless device.* ***Crosscutting Concepts:*** *System Relationships; Communication and Coordination* ***Connection Within Framework:*** *6–8.Data and Analysis.Collection*

By the end of Grade 8: continued from previous page

TROUBLESHOOTING

Comprehensive troubleshooting requires knowledge of how computing devices and components work and interact. A systematic process will identify the source of a problem, whether within a device or in a larger system of connected devices.

Just as pilots use checklists to troubleshoot problems with aircraft systems, people can use a similar, structured process to troubleshoot problems with computing systems and ensure that potential solutions are not overlooked. Because a computing device may interact with interconnected devices within a system, problems may not be due to the specific computing device itself but to devices connected to it. Examples of system components that may need troubleshooting are physical and wireless connections, peripheral equipment, and network hardware. Strategies for troubleshooting a computing system and debugging a program include some problem-solving steps that are similar.

Crosscutting Concepts: *System Relationships; Abstraction*

Connection Within Framework: *6–8.Algorithms and Programming.Algorithms*

By the end of Grade 8: continued from previous page

Networks and the Internet

NETWORK COMMUNICATION AND ORGANIZATION

Computers send and receive information based on a set of rules called protocols. Protocols define how messages between computers are structured and sent. Considerations of security, speed, and reliability are used to determine the best path to send and receive data.

Protocols allow devices with different hardware and software to communicate, in the way that people with different native languages may use a common language for business. Protocols describe established commands and responses between computers on a network, such as requesting data or sending an image. There are many examples of protocols including TCP/IP (Transmission Control Protocol/Internet Protocol) and HTTP (Hypertext Transfer Protocol), which serve as the foundation for formatting and transmitting messages and data, including pages on the World Wide Web. Routers also implement protocols to record the fastest and most reliable paths by sending small packets as tests. The priority at this grade level is understanding the purposeof protocols, while knowing details of how specific protocols work is not expected.

Crosscutting Concepts: *Communication and Coordination; Abstraction; Privacy and Security*

Connection Within Framework: *6–8.Data and Analysis.Storage*

CYBERSECURITY

The information sent and received across networks can be protected from unauthorized access and modification in a variety of ways, such as encryption to maintain its confidentiality and restricted access to maintain its integrity. Security measures to safeguard online information proactively address the threat of breaches to personal and private data.

The integrity of information involves ensuring its consistency, accuracy, and trustworthiness. For example, HTTPS (Hypertext Transfer Protocol Secure) is an example of a security measure to protect data transmissions. It provides a more secure browser connection than HTTP (Hypertext Transfer Protocol) because it encrypts data being sent between websites. At this level, understanding the difference between HTTP and HTTPS, but not how the technologies work, is important.

Crosscutting Concept: *Privacy and Security*

Connection Within Framework: *6–8.Impacts of Computing.Safety, Law, and Ethics*

By the end of Grade 8: continued from previous page

Data and Analysis	
COLLECTION	People design algorithms and tools to automate the collection of data by computers. When data collection is automated, data is sampled and converted into a form that a computer can process. For example, data from an analog sensor must be converted into a digital form. The method used to automate data collection is influenced by the availability of tools and the intended use of the data. *Data can be collected from either individual devices or systems. The method of data collection (for example, surveys versus sensor data) can affect the accuracy and precision of the data. Some types of data are more difficult to collect than others. For example, emotions must be subjectively evaluated on an individual basis and are thus difficult to measure across a population. Access to tools may be limited by factors including cost, training, and availability.* ***Crosscutting Concept:*** *Human–Computer Interaction* ***Connection Within Framework:*** *6–8.Computing Systems.Hardware and Software*
STORAGE	Applications store data as a representation. Representations occur at multiple levels, from the arrangement of information into organized formats (such as tables in software) to the physical storage of bits. The software tools used to access information translate the low-level representation of bits into a form understandable by people. *Computers can represent a variety of data using discrete values at many different levels, such as characters, numbers, and bits. Text is represented using character encoding standards like UNICODE, which represent text as numbers. All numbers and other types of data are encoded and stored as bits on a physical medium. Lossy and lossless data formats are used to store different levels of detail, but whenever digital data is used to represent analog measurements, such as temperature or sound, information is lost. Representations, or file formats, can contain metadata that is not always visible to the average user. There are privacy implications when files contain metadata, such as the location where a photograph was taken.* ***Crosscutting Concept:*** *Abstraction* ***Connections Within Framework:*** *6–8.Algorithms and Programming.Variables; 6–8.Networks and the Internet.Network Communication and Organization*

By the end of Grade 8: continued from previous page

VISUALIZATION AND TRANSFORMATION	Data can be transformed to remove errors, highlight or expose relationships, and/or make it easier for computers to process. *The cleaning of data is an important transformation for reducing noise and errors. An example of noise would be the first few seconds of a sample in which an audio sensor collects extraneous sound created by the user positioning the sensor. Errors in survey data are cleaned up to remove spurious or inappropriate responses. An example of a transformation that highlights a relationship is representing two groups (such as males and females) as percentages of a whole instead of as individual counts. Computational biologists use compression algorithms to make extremely large data sets of genetic information more manageable and the analysis more efficient.* ***Crosscutting Concept:*** *Abstraction* ***Connection Within Framework:*** *6–8.Algorithms and Programming.Algorithms*
INFERENCE AND MODELS	Computer models can be used to simulate events, examine theories and inferences, or make predictions with either few or millions of data points. Computer models are abstractions that represent phenomena and use data and algorithms to emphasize key features and relationships within a system. As more data is automatically collected, models can be refined. *Very large data sets require a model for analysis; they are too large to be analyzed by examining all of the records. While individual users are online, shopping websites and online advertisements use personal data they generate, compared to millions of other users, to predict what they would like and make recommendations. A video-streaming website may recommend videos based on models generated from other users and based upon their personal habits and preferences. The data that is collected about an individual and potential inferences made from that data can have implications for privacy.* ***Crosscutting Concepts:*** *Privacy and Security; Abstraction* ***Connections Within Framework:*** *6–8.Algorithms and Programming.Algorithms; 6–8.Impacts of Computing.Culture*

By the end of Grade 8: continued from previous page

Algorithms and Programming	
ALGORITHMS	Algorithms affect how people interact with computers and the way computers respond. People design algorithms that are generalizable to many situations. Algorithms that are readable are easier to follow, test, and debug. *Algorithms control what recommendations a user may get on a music-streaming website, how a game responds to finger presses on a touchscreen, and how information is sent across the Internet. An algorithm that is generalizable to many situations can produce different outputs, based on a wide range of inputs. For example, an algorithm for a smart thermostat may control the temperature based on the time of day, how many people are at home, and current electricity consumption. The testing of an algorithm requires the use of inputs that reflect all possible conditions to evaluate its accuracy and robustness.* ***Crosscutting Concepts:** Human–Computer Interaction; Abstraction* ***Connections Within Framework:** 6–8.Data and Analysis.Inference and Models; 6–8.Computing Systems.Troubleshooting; 6–8.Data and Analysis.Visualization and Transformation*
VARIABLES	Programmers create variables to store data values of selected types. A meaningful identifier is assigned to each variable to access and perform operations on the value by name. Variables enable the flexibility to represent different situations, process different sets of data, and produce varying outputs. *At this level, students deepen their understanding of variables, including when and how to declare and name new variables. A variable is like a container with a name, in which the contents may change, but the name (identifier) does not. The identifier makes keeping track of the data that is stored easier, especially if the data changes. Naming conventions for identifiers, and thoughtful choices of identifiers, improve program readability.* *The term variable is used differently in programming than the way it is commonly used in mathematics: a program variable refers to a location in which a value is stored, and the name used to access the value is called the identifier. A program variable is assigned a value, and that value may change throughout the execution of the program. Mathematicians typically do not make a distinction between a variable and the variable name. A mathematics variable often represents a set of values for which the statement containing the variable is true.* ***Crosscutting Concept:** Abstraction* ***Connection Within Framework:** 6–8.Data and Analysis.Storage*

By the end of Grade 8: continued from previous page

CONTROL	Programmers select and combine control structures, such as loops, event handlers, and conditionals, to create more complex program behavior. *Conditional statements can have varying levels of complexity, including compound and nested conditionals. Compound conditionals combine two or more conditions in a logical relationship, and nesting conditionals within one another allows the result of one conditional to lead to another being evaluated. An example of a nested conditional structure is deciding what to do based on the weather outside. If it is sunny outside, I will further decide if I want to ride my bike or go running, but if it is not sunny outside, I will decide whether to read a book or watch TV. Different types of control structures can be combined with one another, such as loops and conditionals. Different types of programming languages implement control structures in different ways. For example, functional programming languagesimplement repetition using recursive function calls instead of loops. At this level, understanding implementation in multiple languages is not essential.* ***Crosscutting Concept:** Abstraction*
MODULARITY	Programs use procedures to organize code, hide implementation details, and make code easier to reuse. Procedures can be repurposed in new programs. Defining parameters for procedures can generalize behavior and increase reusability. *A procedure is a module (a group of instructions within a program) that performs a particular task. In this framework, procedure is used as a general term that may refer to an actual procedure or a method, function, or similar concept in other programming languages. Procedures are invoked to repeat groups of instructions. For example, a procedure, such as one to draw a circle, involves many instructions, but all of them can be invoked with one instruction, such as "drawCircle." Procedures that are defined with parameters are generalizable to many situations and will produce different outputs based on a wide range of inputs (arguments).* ***Crosscutting Concepts:** Abstraction; System Relationships*
PROGRAM DEVELOPMENT	People design meaningful solutions for others by defining a problem's criteria and constraints, carefully considering the diverse needs and wants of the community, and testing whether criteria and constraints were met. *Development teams that employ user-centered design create solutions that can have a large societal impact, such as an app that allows people with speech difficulties to translate hard-to-understand pronunciation into understandable language. Use cases and test cases are created and analyzed to better meet the needs of users and to evaluate whether criteria and constraints are met. An example of a design constraint is that mobile applications must be optimized for small screens and limited battery life.* ***Crosscutting Concepts:** Human–Computer Interaction; Abstraction* ***Connection Within Framework:** 3–5.Impacts of Computing.Culture*

By the end of Grade 8: continued from previous page

Impacts of Computing	
CULTURE	Advancements in computing technology change people's everyday activities. Society is faced with tradeoffs due to the increasing globalization and automation that computing brings. *The effects of globalization, such as the sharing of information and cultural practices and the resulting cultural homogeneity, are increasingly possible because of computing. Globalization, coupled with the automation of the production of goods, allows access to labor that is less expensive and creates jobs that can easily move across national boundaries. Online piracy has increased because of information access that traverses national boundaries and varying legal systems.* ***Crosscutting Concepts:*** *Human–Computer Interaction; System Relationships* ***Connection Within Framework:*** *6–8.Data and Analysis.Inference and Models*
SOCIAL INTERACTIONS	People can organize and engage around issues and topics of interest through various communication platforms enabled by computing, such as social networks and media outlets. These interactions allow issues to be examined using multiple viewpoints from a diverse audience. *Social networks can play a large role in social and political movements by allowing individuals to share ideas and opinions about common issues while engaging with those who have different opinions. Computing provides a rich environment for discourse but may result in people considering very limited viewpoints from a limited audience.* ***Crosscutting Concepts:*** *System Relationships; Human–Computer Interaction* ***Connections Within Framework:*** *3–5.Data and Analysis.Visualization and Transformation; 9–12.Data and Analysis.Visualization and Transformation*
SAFETY, LAW, AND ETHICS	There are tradeoffs between allowing information to be public and keeping information private and secure. People can be tricked into revealing personal information when more public information is available about them online. *Social engineering is based on tricking people into breaking security procedures and can be thwarted by being aware of various kinds of attacks, such as emails with false information and phishing. Security attacks often start with personal information that is publicly available online. All users should be aware of the personal information, especially financial information, that is stored on the websites they use. Protecting personal online information requires authentication measures that can often make it harder for authorized users to access information.* ***Crosscutting Concepts:*** *Privacy and Security; Communication and Coordination* ***Connection Within Framework:*** *6–8.Networks and the Internet.Cybersecurity*

By the end of Grade 12

Computing Systems	
DEVICES	Computing devices are often integrated with other systems, including biological, mechanical, and social systems. These devices can share data with one another. The usability, dependability, security, and accessibility of these devices, and the systems they are integrated with, are important considerations in their design as they evolve. *A medical device can be embedded inside a person to monitor and regulate his or her health, a hearing aid (a type of assistive device) can filter out certain frequencies and magnify others, a monitoring device installed in a motor vehicle can track a person's driving patterns and habits, and a facial recognition device can be integrated into a security system to identify a person. The devices embedded in everyday objects, vehicles, and buildings allow them to collect and exchange data, creating a network (e.g., Internet of Things). The creation of integrated or embedded systems is not an expectation at this level.* ***Crosscutting Concepts:*** *System Relationships; Human–Computer Interaction; Privacy and Security* ***Connections Within Framework:*** *9–12.Networks and the Internet.Network Communication and Organization; 9–12.Data and Analysis.Collection; 9–12. Impacts of Computing.Culture*
HARDWARE AND SOFTWARE	Levels of interaction exist between the hardware, software, and user of a computing system. The most common levels of software that a user interacts with include system software and applications. System software controls the flow of information between hardware components used for input, output, storage, and processing. *At its most basic level, a computer is composed of physical hardware and electrical impulses. Multiple layers of software are built upon the hardware and interact with the layers above and below them to reduce complexity. System software manages a computing device's resources so that software can interact with hardware. For example, text editing software interacts with the operating system to receive input from the keyboard, convert the input to bits for storage, and interpret the bits as readable text to display on the monitor. System software is used on many different types of devices, such as smart TVs, assistive devices, virtual components, cloud components, and drones. Knowledge of specific, advanced terms for computer architecture, such as BIOS, kernel, or bus, is not expected at this level.* ***Crosscutting Concepts:*** *Abstraction; Communication and Coordination; System Relationships* ***Connections Within Framework:*** *9–12.Networks and the Internet.Network Communication and Organization; 9–12.Algorithms and Programming.Variables; 9–12. Algorithms and Programming.Modularity*

By the end of Grade 12: continued from previous page

TROUBLESHOOTING	Troubleshooting complex problems involves the use of multiple sources when researching, evaluating, and implementing potential solutions. Troubleshooting also relies on experience, such as when people recognize that a problem is similar to one they have seen before or adapt solutions that have worked in the past. *Troubleshooting information may come from external sources, such as user manuals, online technical forums, or manufacturer websites. Distinguishing between reliable and unreliable sources is important. Examples of complex troubleshooting strategies include resolving connectivity problems, adjusting system configurations and settings, ensuring hardware and software compatibility, and transferring data from one device to another.* ***Crosscutting Concepts:*** *Abstraction; System Relationships* ***Connection Within Framework:*** *9–12.Algorithms and Programming.Program Development*

By the end of Grade 12: continued from previous page

Networks and the Internet	
NETWORK COMMUNICATION AND ORGANIZATION	Network topology is determined, in part, by how many devices can be supported. Each device is assigned an address that uniquely identifies it on the network. The scalability and reliability of the Internet are enabled by the hierarchy and redundancy in networks. *Large-scale coordination occurs among many different machines across multiple paths every time a web page is opened or an image is viewed online. Devices on the Internet are assigned an Internet Protocol (IP) address to allow them to communicate. The design decisions that directed the coordination among systems composing the Internet also allowed for scalability and reliability. Scalability is the capability of a network to handle a growing amount of work or its potential to be enlarged to accommodate that growth.* ***Crosscutting Concepts:*** *Communication and Coordination; Abstraction; System Relationships* ***Connections Within Framework:*** *9–12.Computing Systems.Devices; 9–12. Computing Systems.Hardware and Software; 9–12.Impacts of Computing.Social Interactions*
CYBERSECURITY	Network security depends on a combination of hardware, software, and practices that control access to data and systems. The needs of users and the sensitivity of data determine the level of security implemented. *Security measures may include physical security tokens, two-factor authentication, and biometric verification, but every security measure involves tradeoffs between the accessibility and security of the system. Potential security problems, such as denial-of-service attacks, ransomware, viruses, worms, spyware, and phishing, exemplify why sensitive data should be securely stored and transmitted. The timely and reliable access to data and information services by authorized users, referred to as availability, is ensured through adequate bandwidth, backups, and other measures.* ***Crosscutting Concepts:*** *Privacy and Security; System Relationships; Human–Computer Interaction* ***Connection Within Framework:*** *9–12.Algorithms and Programming.Algorithms*

By the end of Grade 12: continued from previous page

Data and Analysis	
COLLECTION	Data is constantly collected or generated through automated processes that are not always evident, raising privacy concerns. The different collection methods and tools that are used influence the amount and quality of the data that is observed and recorded. *Data can be collected and aggregated across millions of people, even when they are not actively engaging with or physically near the data collection devices. This automated and nonevident collection can raise privacy concerns, such as social media sites mining an account even when the user is not online. Other examples include surveillance video used in a store to track customers for security or information about purchase habits or the monitoring of road traffic to change signals in real time to improve road efficiency without drivers being aware. Methods and devices for collecting data can differ by the amount of storage required, level of detail collected, and sampling rates. For example, ultrasonic range finders are good at long distances and are very accurate, as compared to infrared range finders, which are better for short distances. Computer models and simulations produce large amounts of data used in analysis.* ***Crosscutting Concept:** Privacy and Security* ***Connections Within Framework:** 9–12.Computing Systems.Devices; 9–12. Impacts of Computing.Safety, Law, and Ethics*
STORAGE	Data can be composed of multiple data elements that relate to one another. For example, population data may contain information about age, gender, and height. People make choices about how data elements are organized and where data is stored. These choices affect cost, speed, reliability, accessibility, privacy, and integrity. *A data model combines data elements and describes the relationships among the elements. Data models represent choices made about which data elements are available and feasible to collect. Storing data locally may increase security but decrease accessibility. Storing data on a cloud-based, redundant storage system may increase accessibility but reduce security, as it can be accessed online easily, even by unauthorized users. Data redundancies and backups are useful for restoring data when integrity is compromised.* ***Crosscutting Concepts:** System Relationships; Privacy and Security; Communication and Coordination* ***Connection Within Framework:** 9–12.Algorithms and Programming.Algorithms*

By the end of Grade 12: continued from previous page

VISUALIZATION AND TRANSFORMATION	People transform, generalize, simplify, and present large data sets in different ways to influence how other people interpret and understand the underlying information. Examples include visualization, aggregation, rearrangement, and application of mathematical operations. *Visualizations, such as infographics, can obscure data and positively or negatively influence people's views of the data. People use software tools or programming to create powerful, interactive data visualizations and perform a range of mathematical operations to transform and analyze data. Examples of mathematical operations include those related to aggregation, such as summing and averaging. The same data set can be visualized or transformed to support multiple sides of an issue.* ***Crosscutting Concepts:*** *Abstraction; Human–Computer Interaction* ***Connection Within Framework:*** *6–8.Impacts of Computing.Social Interactions*
INFERENCE AND MODELS	The accuracy of predictions or inferences depends upon the limitations of the computer model and the data the model is built upon. The amount, quality, and diversity of data and the features chosen can affect the quality of a model and ability to understand a system. Predictions or inferences are tested to validate models. *Large data sets are used to make models used for inference or predictions, such as forecasting earthquakes, traffic patterns, or the results of car crashes. Larger quantities and higher quality of collected data will tend to improve the accuracy of models. For example, using data from 1,000 car crashes would generally yield a more accurate model than using data from 100 crashes. Additionally, car crashes provide a wide variety of data points, such as impact speed, car make and model, and passenger type, and this data is useful in the development of injury prevention measures.* ***Crosscutting Concepts:*** *Abstraction; Privacy and Security*

By the end of Grade 12: continued from previous page

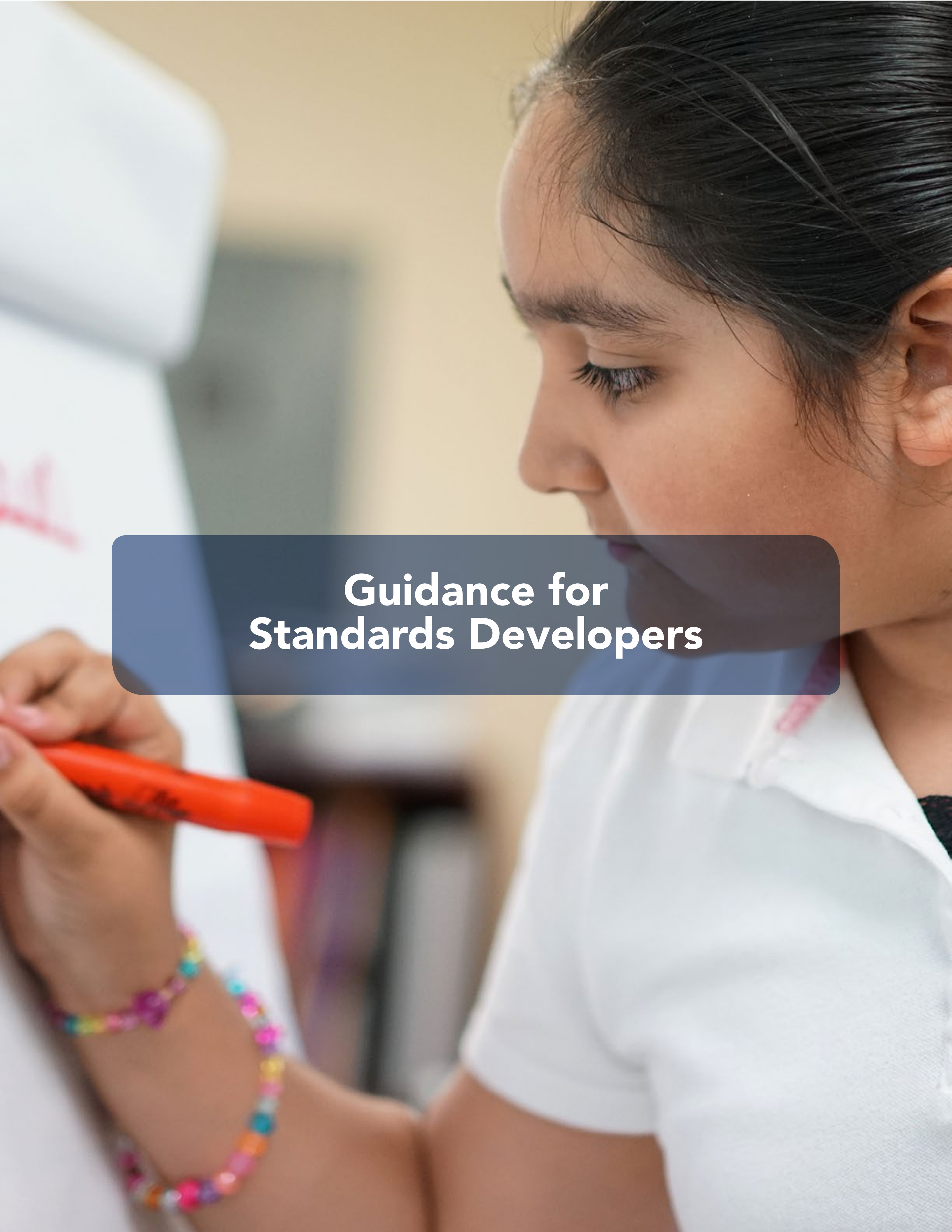
Algorithms and Programming	
ALGORITHMS	People evaluate and select algorithms based on performance, reusability, and ease of implementation. Knowledge of common algorithms improves how people develop software, secure data, and store information. *Some algorithms may be easier to implement in a particular programming language, work faster, require less memory to store data, and be applicable in a wider variety of situations than other algorithms. Algorithms used to search and sort data are common in a variety of software applications. Encryption algorithms are used to secure data, and compression algorithms make data storage more efficient. At this level, analysis may involve simple calculations of steps. Analysis using sophisticated mathematical notation to classify algorithm performance, such as Big-O notation, is not expected.* ***Crosscutting Concepts:*** *Abstraction; Privacy and Security* ***Connections Within Framework:*** *9–12.Data and Analysis.Storage; 9–12.Networks and the Internet.Cybersecurity*
VARIABLES	Data structures are used to manage program complexity. Programmers choose data structures based on functionality, storage, and performance tradeoffs. *A list is a common type of data structure that is used to facilitate the efficient storage, ordering, and retrieval of values and various other operations on its contents. Tradeoffs are associated with choosing different types of lists. Knowledge of advanced data structures, such as stacks, queues, trees, and hash tables, is not expected. User-defined types and object-oriented programming are optional concepts at this level.* ***Crosscutting Concepts:*** *Abstraction; System Relationships* ***Connection Within Framework:*** *6–8.Computing Systems.Hardware and Software*
CONTROL	Programmers consider tradeoffs related to implementation, readability, and program performance when selecting and combining control structures. *Implementation includes the choice of programming language, which affects the time and effort required to create a program. Readability refers to how clear the program is to other programmers and can be improved through documentation. The discussion of performance is limited to a theoretical understanding of execution time and storage requirements; a quantitative analysis is not expected. Control structures at this level may include conditional statements, loops, event handlers, and recursion. Recursion is a control technique in which a procedure calls itself and is appropriate when problems can be expressed in terms of smaller versions of themselves. Recursion is an optional concept at this level.* ***Crosscutting Concepts:*** *Abstraction; System Relationships*

By the end of Grade 12: continued from previous page

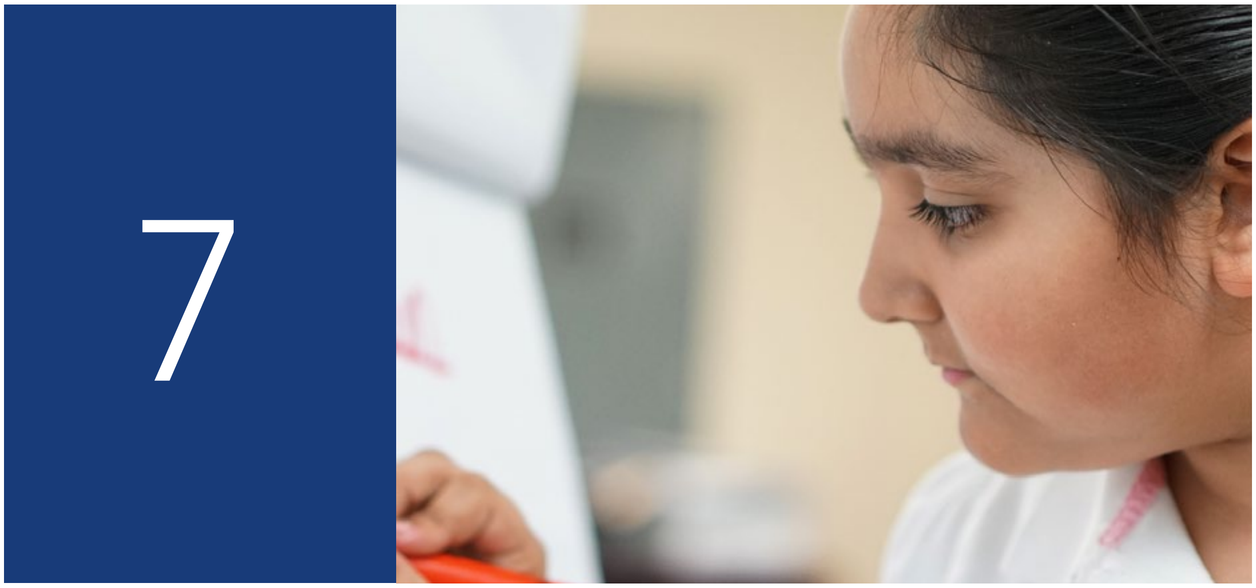
MODULARITY	Complex programs are designed as systems of interacting modules, each with a specific role, coordinating for a common overall purpose. These modules can be procedures within a program; combinations of data and procedures; or independent, but interrelated, programs. Modules allow for better management of complex tasks. *Software applications require a sophisticated approach to design that uses a systems perspective. For example, object-oriented programming decomposes programs into modules called objects that pair data with methods (variables with procedures). The focus at this level is understanding a program as a system with relationships between modules. The choice of implementation, such as programming language or paradigm, may vary.* ***Crosscutting Concepts:*** *System Relationships; Abstraction* ***Connection Within Framework:*** *9–12.Computing Systems.Hardware and Software*
PROGRAM DEVEL-OPMENT	Diverse teams can develop programs with a broad impact through careful review and by drawing on the strengths of members in different roles. Design decisions often involve tradeoffs. The development of complex programs is aided by resources such as libraries and tools to edit and manage parts of the program. Systematic analysis is critical for identifying the effects of lingering bugs. *As programs grow more complex, the choice of resources that aid program development becomes increasingly important. These resources include libraries, integrated development environments, and debugging tools. Systematic analysis includes the testing of program performance and functionality, followed by end-user testing. A common tradeoff in program development is sometimes referred to as "Fast/Good/Cheap: Pick Two": one can develop software quickly, with high quality, or with little use of resources (for example, money or number of people), but the project manager may choose only two of the three criteria.* ***Crosscutting Concepts:*** *Human–Computer Interaction; System Relationships; Abstraction* ***Connection Within Framework:*** *9–12.Computing Systems.Troubleshooting*

By the end of Grade 12: continued from previous page

Impacts of Computing	
CULTURE	The design and use of computing technologies and artifacts can improve, worsen, or maintain inequitable access to information and opportunities. *While many people have direct access to computing throughout their day, many others are still underserved or simply do not have access. Some of these challenges are related to the design of computing technologies, as in the case of technologies that are difficult for senior citizens and people with physical disabilities to use. Other equity deficits, such as minimal exposure to computing, access to education, and training opportunities, are related to larger, systemic problems in society.* ***Crosscutting Concepts:*** *Human–Computer Interaction; System Relationships* ***Connection Within Framework:*** *9–12.Computing Systems.Devices*
SOCIAL INTERACTIONS	Many aspects of society, especially careers, have been affected by the degree of communication afforded by computing. The increased connectivity between people in different cultures and in different career fields has changed the nature and content of many careers. *Careers have evolved, and new careers have emerged. For example, social media managers take advantage of social media platforms to guide the presence of a product or company and increase interaction with their audience. Global connectivity has also changed how teams in different fields, such as computer science and biology, work together. For example, the online genetic database made available by the Human Genome Project, the algorithms required to analyze the data, and the ability for scientists around the world to share information have accelerated the pace of medical discoveries and led to the new field of computational biology.* ***Crosscutting Concepts:*** *System Relationships; Human–Computer Interaction* ***Connection Within Framework:*** *9–12.Networks and the Internet.Network Communication and Organization*
SAFETY, LAW, AND ETHICS	Laws govern many aspects of computing, such as privacy, data, property, information, and identity. These laws can have beneficial and harmful effects, such as expediting or delaying advancements in computing and protecting or infringing upon people's rights. International differences in laws and ethics have implications for computing. *Legal issues in computing, such as those related to the use of the Internet, cover many areas of law, reflect an evolving technological field, and can involve tradeoffs. For examples, laws that mandate the blocking of some file-sharing websites may reduce online piracy but can restrict the right to freedom of information. Firewalls can be used to block harmful viruses and malware but can also be used for media censorship. Access to certain websites, like social networking sites, may vary depending on a nation's laws and may be blocked for political purposes.* ***Crosscutting Concepts:*** *System Relationships; Privacy and Security; Abstraction* ***Connection Within Framework:*** *9–12.Data and Analysis.Collection*



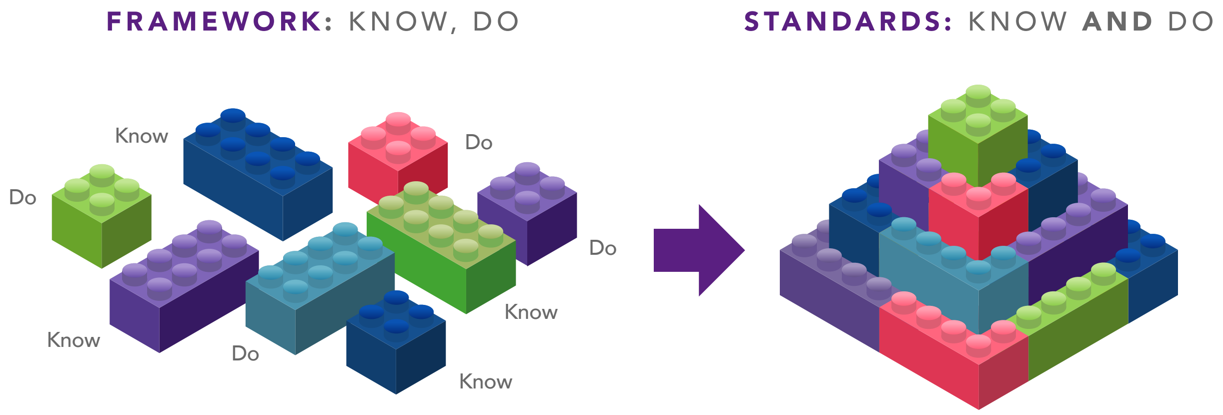
Guidance for Standards Developers



Guidance for Standards Developers

The K–12 Computer Science Framework is designed to serve as a foundation from which all states, districts, and organizations can develop computer science education standards for K–12 students. Standards play a vital role in achieving the vision of computer science for all students. They democratize computer science by setting learning goals for all students and the expectation that all schools will provide opportunities to achieve those goals so that all children, regardless of their age, race, gender, disability, socioeconomic level, or what school they attend, will be able to have engaging and rigorous computer science experiences. As illustrated in Figure 7.1, the framework provides the building blocks by which states can develop their own standards.

Figure 7.1: Building blocks for standards



Standards are an essential component of a larger education plan and can provide a foundation with which to align the other components, such as curriculum, instruction, professional development, and assessment, to better prepare students for success in college and the workplace. They also communicate core learning goals to policymakers, administrators, teachers, parents, and students. Computer science standards provide insight into a discipline that will be new to many teachers and offer inspirational starting points to create projects, lessons, and activities. Standards facilitate the sharing of content, such as lessons, among teachers and are a useful way to categorize that content for easy search and retrieval. Consistent standards promote alignment and connections among different districts within a state so that if a student moves to a different school, he or she will not end up with different expectations.

The purpose of this guide is to provide information and recommendations for the development of K–12 computer science education standards

- at the beginning to set the criteria and prepare standards writers,
- during the writing process with examples and exercises, and
- afterward to help evaluate the outcome.

This guide was developed in partnership with the nonprofit education organization Achieve based on recommendations for standards developers from the National Research Council (NRC, 2012). It also uses criteria and procedures Achieve has established and refined based on aspects of quality academic content standards.

These categories are described in Table 7.1.

Table 7.1: Guidance for Standards Developers summary

CRITERIA	SUMMARY
Rigor: What is the intellectual demand of the standards?	Rigor is the quintessential hallmark of exemplary standards. It is the measure of how closely a set of standards represents the content and cognitive demand necessary for students to succeed in credit-bearing college courses without remediation and in entry-level, high-quality, high-growth jobs. We recommend that standards writers establish and articulate the appropriate level of rigor in computer science to prepare all students for success in college and careers.
Focus/Manageability: Have choices been made about what is most important for students to learn and what is a manageable amount of content?	High-quality standards establish priorities about the concepts and skills that should be acquired by graduation from high school. Choices should be based on the knowledge and skills essential for students to succeed in postsecondary education and the world of work. A sharpened focus also helps ensure that the cumulative knowledge and skills students are expected to learn is manageable. We recommend grade-level standards that clearly communicate student expectations at each stage. In the case of grade-banded standards, we recommend that guidance be provided for users in creating their own grade-level standards or mapping standards to specific courses.

Table continues on next page

Table continued from previous page

Specificity: Are the standards specific enough to convey the level of performance expected of students?	High-quality standards are precise and provide sufficient detail to convey the level of performance expected without being overly prescriptive. Standards that maintain a relatively consistent level of precision ("grain size") are easier to understand and use. Those that are overly broad or vague leave too much open to interpretation, increasing the likelihood that students will be held to different levels of performance, while standards that are too prescriptive encourage a checklist approach to teaching and learning that undermines students' opportunities to demonstrate their understanding in equitable ways. We recommend that standards developers write standards that are neither too broad nor too specific and that the grain size is consistent across the standards.
Equity/Diversity: Were the standards written for all students by a diverse set of writers and reviewers? Are students able to demonstrate performance in multiple ways?	Standards, just like other aspects of education infrastructure, play a role in creating an equitable environment for all students. We recommend that diversity and equity be attended to not only in the makeup of the groups writing, advising, and reviewing the standards but also in the standards content by designing standards that can be engaged in by ALL students and are flexible enough to allow them to demonstrate proficiency in multiple ways.
Clarity/Accessibility: Are the standards clearly written and presented in an error-free, legible, easy-to-use format that is accessible to the general public?	Clarity requires more than just plain and jargon-free prose that is free of errors. Standards also must be communicated in language that can gain widespread acceptance not only by postsecondary faculty but also by employers, teachers, parents, school boards, legislators, and others who have a stake in schooling. A straightforward, functional format facilitates user access. We recommend that standards writers consider the knowledge level of users of the standards by clarifying terms and providing examples.
Coherence/Progression: Do the standards convey a unified vision of the discipline, do they establish connections among the major areas of study, and do they show a meaningful progression of content across the grades?	The way in which standards are categorized and broken out into supporting strands should reflect a coherent structure of the discipline and/or reveal significant relationships among the strands and how the study of one complements the study of another. If standards suggest a progression, that progression should be meaningful and appropriate across the grades or grade spans. We recommend that standards writers clearly communicate progressions of content and practices in the standards.
Measurability: Is each standard measurable, observable, or verifiable in some way?	In general, standards should focus on the results, rather than the processes of teaching and learning. Standards should make use of performance verbs that call for students to demonstrate knowledge and skills and should avoid using those that refer to learning activities, such as examine, investigate, and explore, or to cognitive processes that are hard to verify, such as appreciate. We recommend ensuring that each standard is measurable.
Integration of Practices and Concepts: Does each standard reflect at least one practice and one concept?	To ensure that instruction reflects both knowing and doing computer science, the core concepts of computer science should be taught alongside the practices by fully integrating them at the standards level. We recommend that standards integrate the computer science practices with the concept statements.
Connections to Other Disciplines: Are there explicit ways in which computer science is shown to be relevant in other subjects?	There are many possible areas of overlap between computer science and subject areas such as math, science, and engineering as well as humanities, including languages, social studies, art, and music. Making intentional connections between computer science standards and academic standards in other disciplines will promote a more coherent education experience. We recommend that computer science standards be written to align with and connect to (possibly via clarifying examples) state math and science standards, as well as standards from other disciplines.

Recommendations

Recommendation 1: Rigor. We recommend that standards establish and articulate the appropriate level of rigor in computer science to prepare all students for success in college and careers.

High-quality standards create foundational expectations for all students, rather than just those interested in advanced study, and prepare students for a variety of postsecondary experiences.

Standards aim to prepare students for the demands of the world they will encounter after graduation. That preparation is even more difficult when the job market changes rapidly as the influence of technology in the workforce grows steadily. It is therefore critical that standards describe rigorous expectations in computer science for all students. In addition, some students will want to specialize in computer science fields and require an even higher level of intellectual demand than is necessary for all students.

To facilitate appropriate use of the standards, differentiating between technical career standards for advanced courses and core academic standards for all students is crucial. The former may be equivalent to the expectations for specialized computer science courses—in particular, career and technical education pathways. In contrast, standards for all students describe expectations that will be important for every student to meet to help ensure their future success in any chosen field.

For example, the different standards in Figure 7.2 are based on the same practice and concept in the 9–12 grade band of the K–12 Computer Science Framework, and they compare a standard for an advanced course with a standard for all students.

Figure 7.2: Differentiating rigor for all students

Practice: Testing and Refining Computational Artifacts	Concept: Systematic analysis is critical for identifying the effects of lingering bugs. (9–12.Algorithms and Programming.Program Development)

Example 1: Test and refine software components by using unit tests to identify lingering bugs during an agile programming development cycle.

Example 2: Test and refine a program using a systematic debugging process as part of a larger iterative development process.

The first standard would be more appropriate for high school students in a specialized career and technical education course, as it calls for a product (software components) and methodologies (unit tests and agile development) that are specific to the software industry. The second standard sets a goal for all students that reflects a more general product (any computer program) yet still maintains rigor through the expectation of a systematic and iterative process.

Standards meant for all students should have sufficient rigor to help prepare students to enter and succeed in entry-level postsecondary courses that require skills such as critical thinking, problem solving, and computational literacy. Rigor applies equally to practices and concepts.

The K–12 Computer Science Framework was written to describe a vision of computer science education for all students, so most standards based on the framework could be written at a level of rigor intended for all students, rather than for students in advanced courses. Care should be taken to align the standards with grade-appropriate student abilities. It is possible that feedback or current system constraints could influence standards writers to try to limit the rigor of the standards, particularly at the elementary grade levels. However, research into students' use of sequence and iteration and practice of other aspects of computational thinking indicates that students can learn computer science at young ages (Flannery et al., 2013) when they have the support and opportunities to do so. Standards writers should be careful to keep rigor at a high enough level for younger students to ensure that all students have access to high-quality computer science education. The concept statements in the K–2 and 3–5 grade bands of the framework have been reviewed by early childhood computer science education experts and provide a blueprint for the appropriate expectations for elementary-age students. See Figure 7.3 for criteria to determine if a standard has the right amount of rigor.

Computer Science Applies to Many College Majors and Careers

Business:
Business professionals can apply processes learned in computer science to expand a business and optimize for efficiency.

Music:
Musicians can design sounds, effects, and filters. They can create a system to control music using gestures to manipulate sounds and visuals for a live show.

Biology:
Researchers can analyze a database of genetic sequences for genes similar to a known cancer gene.

Sports:
Coaches can create algorithms to analyze the performance of athletes as a training tool or develop strategies using real-time data on the field.

Figure 7.3: Determining the right amount of rigor for a standard

A standard should meet all three of these criteria:

- Does the standard require an appropriate level of conceptual understanding?
- Does it require application of that concept?
- Does it require engagement with a practice?

To help ensure that standards set expectations that prepare students for success in entry-level postsecondary courses and careers, feedback from employers and faculty members, including from two-year institutions, is crucial. The involvement of reviewers with a perspective on student preparation for postsecondary courses and careers will provide valuable information about the rigor necessary in the standards.

Recommendation 2: Focus/Manageability. We recommend that standards be limited in number, focus on the content and practices described in the framework, and be written for individual grade levels or courses.

High-quality standards prioritize the concepts and skills that should be acquired by students. A sharpened focus helps ensure that the knowledge and skills students are expected to learn are important and manageable in any given grade or course.

A clear focus within standards helps teachers see and prioritize learning experiences for students. Therefore the framework was developed to describe a core set of concepts and practices, which were selected using criteria developed by the writing team and vetted by the computer science community during review periods. Standards based on the framework should focus on the set of concepts and practices described here, rather than incorporating additional topics that could be included in advanced computer science courses.[1] See Table 7.2 for examples of important topics that are essential or not essential for all students to learn.

1 Additional topics would be appropriate for standards for advanced courses, if they are clearly designated as such and not as standards for all students (see Recommendation 1).

Table 7.2: Examples of essential and non-essential topics

IMPORTANT AND ESSENTIAL FOR ALL STUDENTS	IMPORTANT BUT NOT ESSENTIAL FOR ALL STUDENTS
• Troubleshooting strategies	• Operating systems
• Searching and sorting	• Algorithmic efficiency
• Digital data representations	• Relational databases
• Basic online security measures	• Cryptography methods

This focus will help ensure that the limited time available for computer science education throughout K–12 is concentrated on those areas that are priorities for all students. Additional standards could be added for elective computer science courses, but those should be noted as elective and not for all students. Figure 7.4 provides an example of a standard appropriately focused on the concept.

Figure 7.4: Focusing on the concept

Practice: Recognizing and Defining Computational Problems
Concept: Different software tools used to access data may store the data differently. The type of data being stored and the level of detail represented by that data affect the storage requirements. (3–5.Data and Analysis.Storage)

Standard that focuses on the concept: Evaluate the appropriateness of different ways to store data based on the type of data and the level of detail.

Standard that includes extraneous concepts: Evaluate the appropriateness of binary, octal, and hexadecimal representations of data and convert between bits and bytes.

Another aspect of appropriate focus is that standards are developed for either specific grade levels or courses. Although the framework's statements are written for grade bands (i.e., K–2, 3–5, 6–8, 9–12) and more accurately, grade-band endpoints, standards developed from the framework should be written for individual grade levels. For example, the framework's expectations by the end of 5th grade (Grades 3–5) may inform standards in all three grade levels—Grades 3, 4, and 5—or in Grade 5 only. If grade level standards are not possible, guidance should be provided about how users of the standards can create their own grade level or course-specific student expectations. Narrowing the focus of student goals at each grade level or course—either by standards writers or by district and state administrators—will enable alignment across the education system and help ensure that teachers have the support they need to focus on particular standards during a course or grade level.

Recommendation 3: Specificity. We recommend that standards writers attend to the specificity of the standards to ensure that they are neither too broad nor too specific and that the grain size, when possible, is consistent across the standards.

High-quality standards are precise and provide sufficient detail to convey the level of performance expected without being overly prescriptive. Those that maintain a relatively consistent level of precision tend to have consistent interpretation and use. Conversely, those that are overly broad or vague leave too much open to interpretation and are implemented inconsistently, and overly specific standards reduce students' opportunities to demonstrate their understanding in flexible ways.

Writing standards to a useful level of specificity requires a balance between being too vague and too specific (see Figure 7.5). A consistent and appropriate level of specificity will help ensure that teachers have the understanding and support they need to help students reach the standards. When standards are too broad, a teacher must interpret the intent of the standards—to decide what types of connections are to be understood and what depth of complexities of problems are to be solved. Useful specificity can often be added with boundary statements, which specify what content is not expected, clarifying the scope of material to be taught. For example, students by the end of eighth grade should know that network protocols exist to allow different computers to communicate with one another but not the structure of messages sent using a specific protocol, such as HTTP (Hypertext Transfer Protocol).

Figure 7.5: A spectrum of specificity in standards

	Standard	**Comments**
Too vague	Use conditionals in a program.	This standard lacks context and is too vague to be assessed.
Balanced	Design an algorithm that efficiently uses conditional statements to represent multiple branches of execution.	This standard specifies the type of product and a level of rigor yet allows for multiple contexts in which to demonstrate performance.
Too specific	Create an app to help friends decide between watching a comedy, action, or science fiction movie by using three if-statements.	The context for this standard is too specific and does not allow for a range of demonstrations of performance.

Consistency in the level of specificity across the standards is also important (see Figure 7.6). In practice, standards within the same document may be interpreted to have equal levels of specificity and may thus be allotted equal amounts of instructional time. It is more difficult for educators, curriculum designers, and assessment developers to use standards that vary in scope across grade levels.

Figure 7.6: Calibrating specificity across standards writers

Create a set of three to five standards that vary in specificity and have different standards writers (as small groups or individuals) put them in order and compare. Discuss the differences and characteristics of each standard, then select the one or two examples of specificity that the group should be aiming for when writing standards.

Recommendation 4: Equity/Diversity. We recommend that diversity and equity be attended to by developing standards that allow for engagement by ALL students and allow for flexibility in how students may demonstrate proficiency. The makeup of the groups of stakeholders writing and reviewing the standards should be diverse.

The framework is based on the belief that all students, regardless of race, gender, socioeconomic class, or disability, when given appropriate support, can learn all of the concepts and practices described in the framework.

Equitable standards create expectations for students with a variety of college and career interests, allow for flexible demonstrations of performance, do not assume out-of-school preparation, and are written by stakeholders with diverse perspectives.

Standards that are created for all students focus on the core aspects of computer science that are applicable to a wide range of college and career choices, rather than extraneous content with narrow application. The concepts and practices of the K–12 Computer Science Framework represent literacy in computer science for all students, not just students interested in majoring in the field or pursuing technical careers.

If computer science education is expected of all students, it must also be equitable and allow students to demonstrate their knowledge and skill in multiple ways. When a standard is particularly prescriptive, such as when it resembles the scope ("grain size") of an assessment item, it prescribes a particular way that students should demonstrate their understanding, creating the potential for an inequitable classroom environment. Equitable standards are not biased for or against students from a particular background. This includes making standards accessible to students with special needs or English language learners.

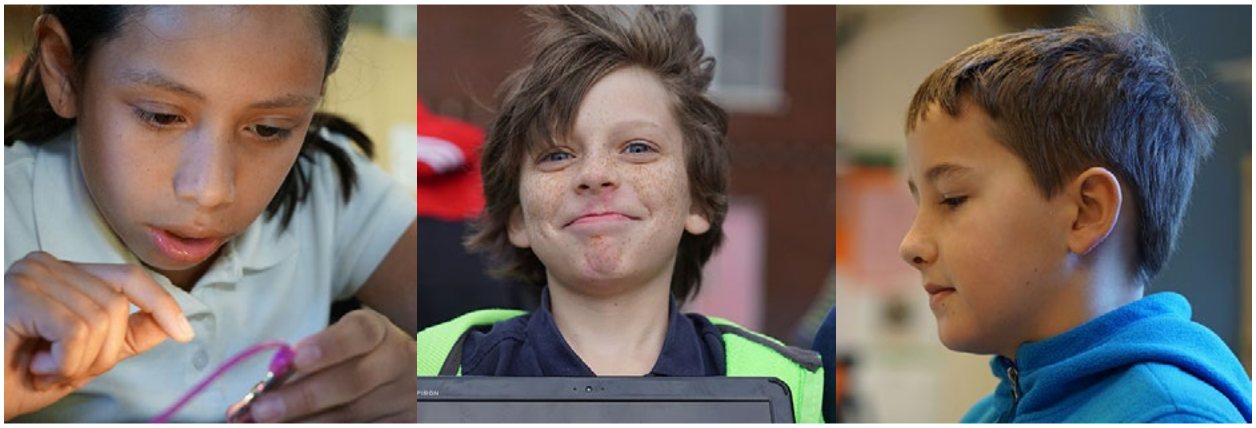
Equitable standards do not presuppose content knowledge, and therefore a level of preparation, in computer science but instead include all key stages in a learning progression. Incomplete learning progressions require out-of-school opportunities to fill in gaps in knowledge, putting students without these experiences at a disadvantage.

Developing equitable standards requires diverse stakeholders. The writers and reviewers involved in developing the standards should include diverse representation from two- and four-year institutions; the research community; industry; and most important, K–12 education, including expertise in early childhood, English language learners, and students with disabilities. This diversity will help ensure that different perspectives and areas of expertise are involved in each standard's development decision and that writers and reviewers can review each statement and example for possible bias. For example, creating standards that require specific equipment or software that is not readily accessible will disadvantage certain groups, such as rural or poor communities.

Recommendation 5: Clarity/Accessibility. We recommend that standards writers clarify standards for the average user of the standards, including defining terms and providing examples.

High-quality standards are clearly written and presented in an error-free, legible, easy-to-use format that is accessible to both the targeted instructors and the general public.

Writing clear and accessible standards is challenging. As content experts, writers may tend to drift into technical language. Additionally, computer scientists may use terms in different ways than many of the users of the standards. Precision in meaning is important but so is an awareness of the audience that will be reading and implementing the standards. In all cases, standards writers must attend to the technical understanding of the user as well as the actual content of the standard.



Computer science standards writers should consider the potential technical understanding of the average user, given the current scenarios in which computer science is taught. Rather than decreasing rigor, writers should consider how to frame standards language so that it is accessible to educators who are teaching computer science outside of their primary area of certification and may not have a computer science background. In most elementary schools, teachers are generalists, with no special training in computer science. Policymakers and community members also need to understand the educational priorities communicated by the standards. It is therefore critical for computer science standards to be accessible to many different audiences.

Precise use of language is very helpful in creating a common understanding of student outcomes among varied users, such as educators, curriculum developers, and assessment designers. Clarifications could come via boundary statements that describe the limits of standards; parenthetical notes within the standards themselves; or separate, nonassessable statements that accompany the standards. This is particularly true when words like abstraction, parallelization, and even algorithms may be used differently in different disciplines. Technical terms should be defined and, as often as possible, plain language restatements added so that the readers, particularly teachers, will be able to understand and apply the standards consistently for both curriculum and assessment. Explanations, simpler language, and/or detailed descriptions would be helpful to ensure consistent application of the standards (see Figure 7.7).

Figure 7.7: Example of technical terms versus simple language in standards

Standard 1: Use an API by calling a procedure and supplying arguments with appropriate data types to efficiently employ high-level functionality.

Standard 2: Select and use a procedure from a library of procedures (API) and provide appropriate input as arguments to replace repetitive code.

The second standard retains "API" (application programming interface), adds more accessible wording such as "library of procedures," and prefaces the specific programming term "argument" with the more general "appropriate input." The second standard continues to use the terms "procedure" and "arguments" as these are necessary terms that provide clarity. Accessible standards use key terminology to provide clarity and avoid extraneous terms and technical jargon.

Examples are very useful to communicate the intent of the standards to users. However, when examples are used, we recommend that multiple examples always be present. The use of single examples can often seem to be a limiting factor or inadvertent prescription of curriculum (Achieve, 2010).

Recommendation 6: Coherence/Progression. We recommend that standards be organized as progressions that support student learning of content and practices over multiple grades.

Coherence refers to how well a set of standards conveys a unified vision of the discipline, establishing connections among the major areas of study and showing a meaningful progression of content across grade levels and grade spans.

Research on student learning indicates that students need explicit help to connect new ideas to ideas that have been learned previously (Marzano, 2004). To support teachers as they help students make these connections, standards should describe developmentally appropriate levels of a learning progression, and the learning progressions embedded in standards must be made apparent to users. This is true for both the content and the practices, as students' facilities with each of the practices change and deepen over time when they are provided adequate instructional opportunities. Separate displays that show the progression of each dimension through K–12 have been very useful to educators in implementing standards.

The framework writers were careful to describe coherent progressions of content and skills across grade bands. Standards based on the framework, however, may be written for individual grade levels. In that case, care should be taken to ensure that the progression from grade level to grade level is coherent and research-based as much as possible and that student knowledge and practice will build on the foundation of content and skills learned previously. The progressions in the framework revolve around a central subconcept in each core concept area and reflect developmentally appropriate milestones that grow in sophistication from kindergarten to Grade 12 (see Figure 7.8).

Figure 7.8: Example learning progression

Computing Systems.Hardware and Software

By the end of Grade 2: A computing system is composed of hardware and software. Hardware consists of physical components, while software provides instructions for the system. These instructions are represented in a form that a computer can understand.

By the end of Grade 5: Hardware and software work together as a system to accomplish tasks, such as sending, receiving, processing, and storing units of information as bits. Bits serve as the basic unit of data in computing systems and can represent a variety of information.

By the end of Grade 8: Hardware and software determine a computing system's capability to store and process information. The design or selection of a computing system involves multiple considerations and potential tradeoffs, such as functionality, cost, size, speed, accessibility, and aesthetics.

By the end of Grade 12: Levels of interaction exist between the hardware, software, and user of a computing system. The most common levels of software that a user interacts with include system software and applications. System software controls the flow of information between hardware components used for input, output, storage, and processing.

Recommendation 7: Measurability. We recommend ensuring that each standard is objective and measurable.

Standards should focus on the results, rather than the processes of teaching and learning. They should make use of performance verbs that call for students to demonstrate knowledge and skills, with each standard being measurable, observable, or verifiable in some way.

To be effective for teaching and learning, standards must be observable and measurable. What the standard intends a student to understand or be able to do should be clear. Accordingly, teachers need to be able to clearly determine if the expectation has been met to know whether students need further help with these concepts.

However, standards do not necessarily need to be written such that they could be tested on a large-scale summative assessment. They simply need to be observable by some measure, including by a classroom teacher. Careful selection of the verbs used in each standard, along with specificity of content, will help ensure that the standard is observable and measurable (see Table 7.3).

Table 7.3: Examples of verbs that assist with measurability

VERBS THAT REFER TO OBSERVABLE PERFORMANCE OR RESULTS	VERBS THAT REFER TO LEARNING ACTIVITIES	VERBS THAT REFER TO COGNITIVE PROCESSES
Create	Examine	Know
Develop	Explore	Understand
Test	Observe	Appreciate
Refine	Discover	
Communicate		

Recommendation 8: Integration of Practices and Concepts. We recommend that standards integrate the computer science practices with the concept statements.

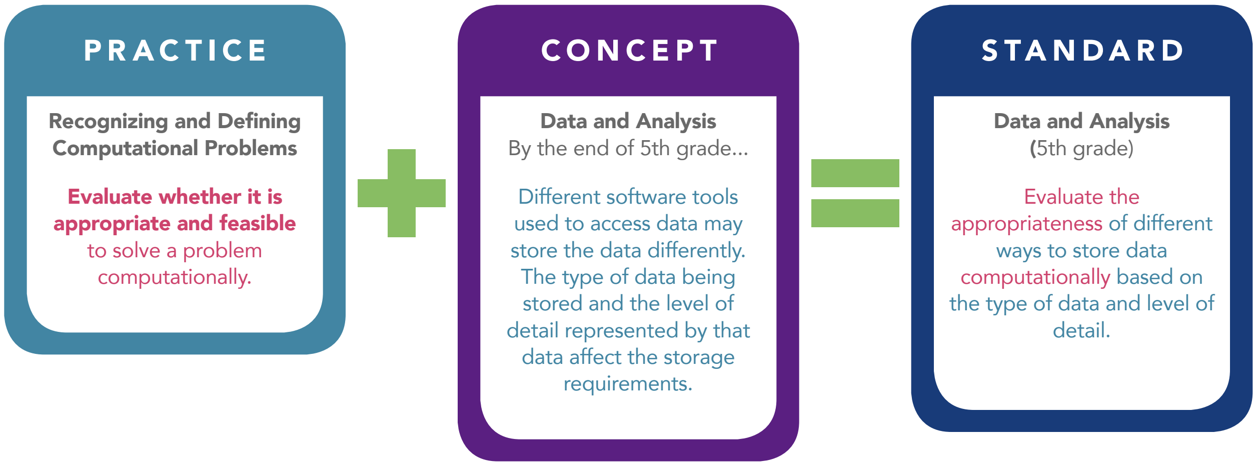
To realize the vision described in this framework and to ensure that all students can become proficient users of computer science knowledge and practice, the practices and concepts should be integrated in the standards, as well as in curriculum and instruction.

Previous sets of education standards in many different disciplines included separate practice and content standards. However, because teachers and curriculum designers were more familiar and comfortable with the content standards, the practice standards were very rarely implemented. They were separate, so they were typically left out or "covered" in the first week of school and then forgotten, or they were used irregularly. One efficient way to help ensure that practices are included throughout instruction is to integrate them completely with the content standards.

More important, part of the vision for computer science education is that students will become proficient at using and applying knowledge—not just memorizing it. If application and deep understanding is indeed the goal, education standards should be written to reflect that goal. By combining a practice with each concept statement to create a standard, the resulting standards more closely describe the behavior, abilities, and deep knowledge we want students to have.

Figure 7.9 below shows an example of how to integrate a computer science practice with a concept statement from the framework.

Figure 7.9: Example of integrating a practice and concept to create a standard



PRACTICE

Recognizing and Defining Computational Problems

Evaluate whether it is appropriate and feasible to solve a problem computationally.

CONCEPT

Data and Analysis
By the end of 5th grade...

Different software tools used to access data may store the data differently. The type of data being stored and the level of detail represented by that data affect the storage requirements.

STANDARD

Data and Analysis
(5th grade)

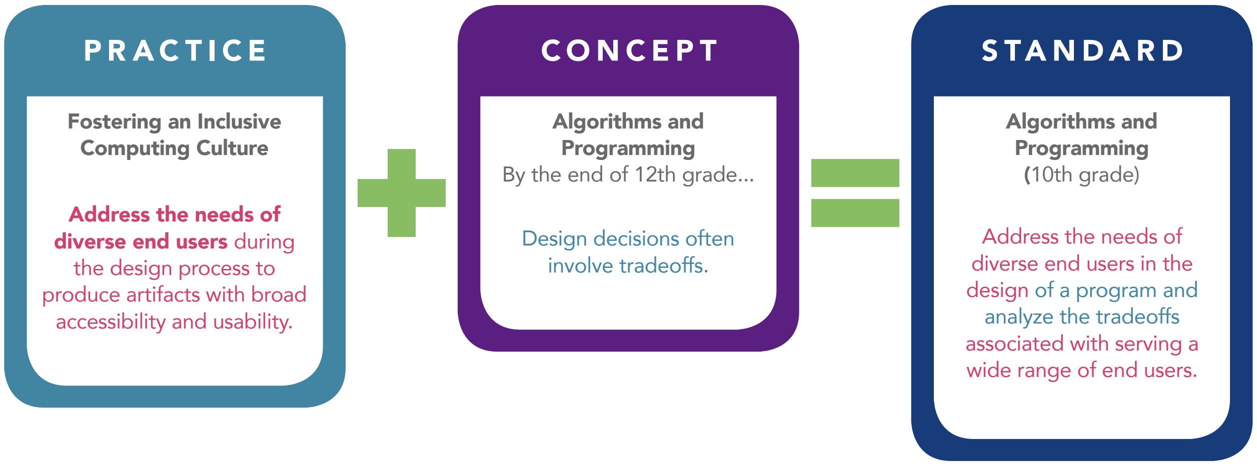
Evaluate the appropriateness of different ways to store data computationally based on the type of data and level of detail.

The following steps were taken to create this example.

1. The writer chose a specific practice statement within Practice 3: *Recognizing and Defining Computational Problems.*
2. The writer selected a portion of the *Data and Analysis* concept statement as a context for applying the practice.
3. The practice and concept were combined to create an observable performance expectation that calls for the application of the practice within the context of the concept. The bolded verb stem in the practice statement helped to focus the action in the standard.

Figure 7.10 provides another example of integrating a practice and concept to create a standard. By using the checklist provided in Recommendation 1: Rigor, we see that this standard requires an appropriate level of content understanding, as reflected in the concept portion [highlighted in blue] and engagement with a practice [highlighted in magenta], which facilitates the application of the content [the standard as a whole].

Figure 7.10: Second example of integrating a practice and concept to create a standard


PRACTICE
Fostering an Inclusive Computing Culture
Address the needs of diverse end users during the design process to produce artifacts with broad accessibility and usability.
CONCEPT
Algorithms and Programming
By the end of 12th grade…
Design decisions often involve tradeoffs.
STANDARD
Algorithms and Programming
(10th grade)
Address the needs of diverse end users in the design of a program and analyze the tradeoffs associated with serving a wide range of end users.

It is not expected, or recommended, that each concept statement be combined with statements from all of the practices to form multiple standards. For example, although there are a total of 68 concept statements and seven practices (each of which has multiple statements), a K–12 standard set should not expect to have 476 standards (i.e., 68 multiplied by 7). Only the practice statements that are most relevant to a concept statement should be considered. In addition, remember that integrating practices with concept statements often introduces more rigor to the student performance expectation than would be seen in the concept statement on its own because students now will have to do something with that conceptual knowledge. Care should be taken to ensure that the particular combination of practices and concepts does not introduce a higher level of rigor than is appropriate for the grade band. Figure 7.11 provides an exercise for standards developers using these considerations.

Figure 7.11: Exercise in standards creation

1. As a group, pick the same concept and practice and create a standard from the pairing.
2. Compare each other's proposed standard.
3. Ask:
   a. Is the rigor appropriate for the grade band?
   b. Is the performance expectation clear?
   c. Does it accurately reflect components of the concept and practice?
   d. Is this an appropriate standard for all students or just those going on to extended study?

Recommendation 9: Connections to Other Disciplines. We recommend that computer science standards be written to align with and connect to other academic standards, such as mathematics and science.

There are many possible ways computer science can connect with other subjects, like math, science, and engineering, as well as humanities, such as languages, social studies, art, and music. Making intentional connections between computer science standards and academic standards in other disciplines will help teachers understand how computer science can connect with their implementation of standards in other subjects and promote more coherent education experiences for students. While related, technology/digital literacy and computer science are distinct subjects.

With limited time in the classroom, students' education should be as coherent as possible. When content in different disciplines is related or connecting, it is important to point out those connections to educators and to facilitate them through standards. When potential alignments are not recognized in standards, extra instructional time may be required to cover everything. For example, if a core math concept is required for third grade computer science standards but is not included in math standards until fifth grade, third grade teachers would need to add that concept into their computer science curriculum, or they might end up ignoring the computer science content due to an impression that it is too overwhelming.

In addition to aligning grade-level expectations, it can also be helpful to include clarifying examples that align and connect to math, science, and engineering standards (see Figure 7.12).

Figure 7.12: Example of a computer science standard that connects with a science standard

Standard: Test and refine a program using a wide range of inputs until criteria and constraints are met.

This standard connects with Next Generation Science Standard MS-ETS1-2 Engineering Design: Evaluate competing design solutions using a systematic process to determine how well they meet the criteria and constraints of the problem.

Attention should also be given to connections to subjects outside of science, technology, engineering, and math, such as language arts literacy standards for technical subjects. Since computer science is not currently required or assessed in most states, illustrating how the standards connect to and help meet existing standards in other subjects will be very useful. These connections can be made through ancillary materials like crosswalks and examples and can be used as a tool to integrate content from other subjects into computer science or embed computer science content into other subjects. This is particularly true for Grades K–8, as budget constraints may not allow for separate computer science teachers in elementary and middle schools.

In 2010, the Association for Computing Machinery's *Running on Empty* reported that "there is deep and widespread confusion within the states as to what should constitute and how to differentiate technology education, literacy and fluency; information technology education; and computer science as an academic subject" (p. 9). While it is plausible to combine digital/technology literacy standards with academic computer science standards, care should be taken so as not to confuse addressing one with addressing the other. For example, while a digital presentation can be used to communicate a team's software development process, the creation of the digital presentation, or the general use of office productivity software, is not a computer science activity. Again, *Running on Empty* reported that "consistent with efforts to improve 'technology literacy,' states are focused almost exclusively on skill-based aspects of computing (such as using a computer in other learning activities) and have few standards on the conceptual aspects of computer science that lay the foundation for innovation and deeper study in the field (for example, develop an understanding of an algorithm)" (p. 7). If combining digital literacy and computer science into one set of standards, it is important that the distinction be kept clear through separately identifiable strands.

References

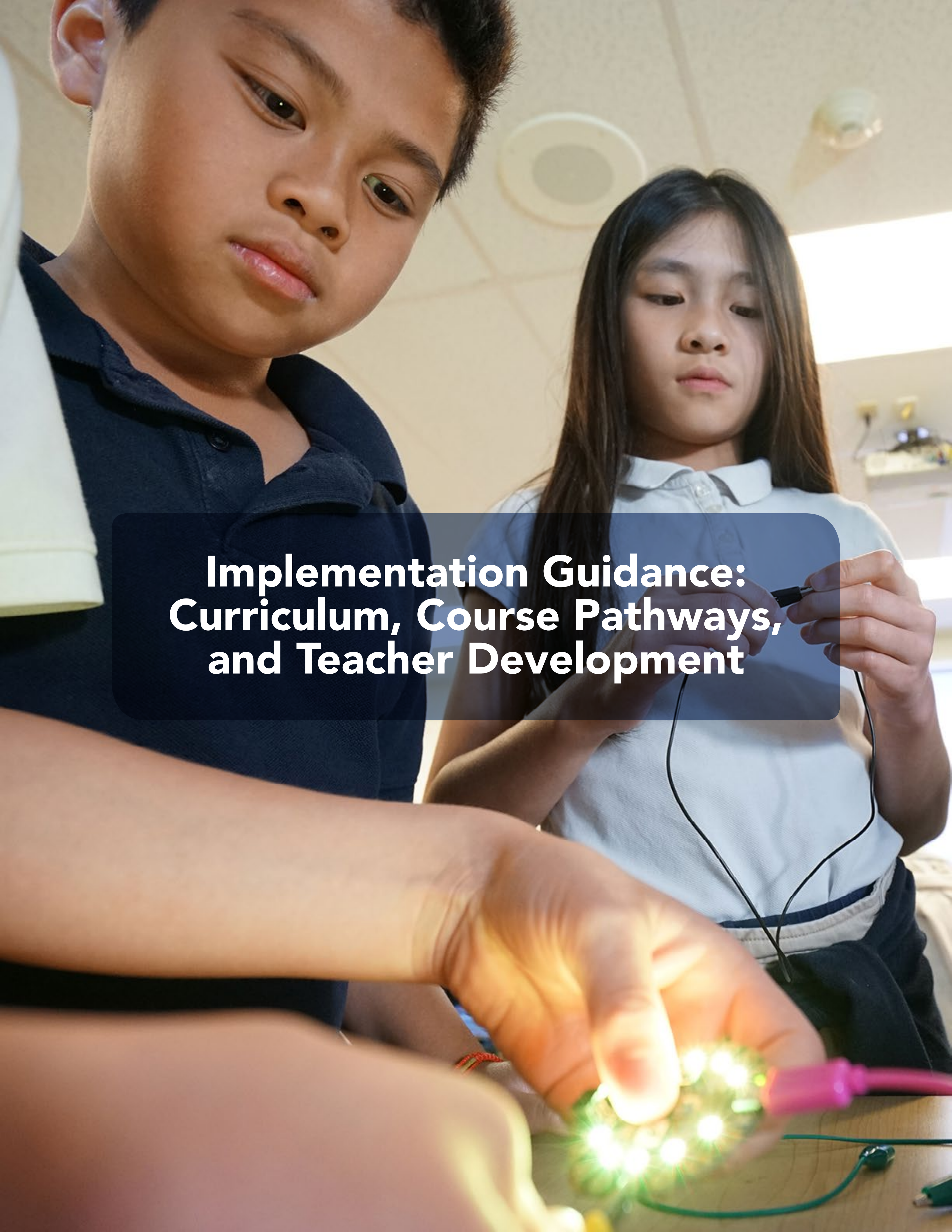
Achieve. (2010). *International science benchmarking report*. http://www.achieve.org/files/InternationalScience BenchmarkingReport.pdf

Association for Computing Machinery & Computer Science Teachers Association. (2010). *Running on empty: The failure to teach K–12 computer science in the digital age.* Retrieved from http://runningonempty.acm.org/fullreport2.pdf

Flannery, L. P., Kazakoff, E. R., Bontá, P., Silverman, B., Bers, M. U., & Resnick, M. (2013, June). Designing ScratchJr: Support for early childhood learning through computer programming. In *Proceedings of the 12th International Conference on Interaction Design and Children* (pp. 1–10).

Marzano, R. J. (2004). *Building background knowledge for academic achievement: Research on what works in schools.* Alexandria, VA: Association for Supervision and Curriculum Development.

National Research Council. (2012). *A Framework for K–12 science education: Practices, crosscutting concepts, and core ideas.* Committee on a Conceptual Framework for New K–12 Science Education Standards. Board on Science Education, Division of Behavioral and Social Sciences and Education. Washington, DC: The National Academies Press.



Implementation Guidance: Curriculum, Course Pathways, and Teacher Development

8

Implementation Guidance: Curriculum, Course Pathways, and Teacher Development

A surge of interest across the nation has led many schools, districts, and states to start figuring out how to increase student opportunities for learning computer science. They are tackling tough questions, such as: How does learning build over time? What technical infrastructure is required to offer computer science? How does computer science fit in the school schedule? How will enough teachers be prepared? Although grassroots efforts to integrate computer science at the classroom level have existed for decades,[1] current efforts seek to also address computer science at the state and district levels, at large scale, and for all students. At any grade level, the implementation of computer science is accompanied by the unique opportunities and challenges of adding a new discipline to the scope of K–12 education.

Large-scale computer science education reform is still a new frontier, and states and districts are trying a variety of approaches. As of September 2016, governors of six states pledged to work towards three policy goals: offer at least one computer science course in all of their high schools, fund professional development opportunities to build teacher capacity, and create comprehensive K–12 computer science standards (Governors for Computer Science, 2016). Large school districts in cities including New York City, San Francisco, and Oakland have launched multi-year initiatives that will lead to every school offering computer science, with some also aiming to provide instruction to every single student. Chicago Public Schools has gone one step further by becoming the largest district in the nation to create a computer science graduation requirement for all students (Chicago Public Schools, 2016).

1 The use of the Logo programming language in K–12 schools peaked in the early to mid-1980s. Logo's goals went beyond introducing computing fundamentals. Logo included the opportunity to create with technology, and in the process, develop (computational) thinking skills.

This chapter suggests topics for state boards and departments of education to consider as they develop and adopt policies to support the expansion of K–12 computer science, and it provides guidance to district- and school-level administrators, teachers, and informal educators who are planning to implement computer science. K–12 education is a complex system composed of many interacting parts that must coordinate to achieve a common purpose: computer science for all students. This chapter specifically addresses curriculum, assessment, course pathways, technical infrastructure, stakeholder involvement, preservice programs, certification, and professional development. The vision, concepts, and practices of the framework play a key role in each of these pieces. Standards development also plays a vital role in the implementation of computer science education and is discussed in the **Guidance for Standards Developers** chapter.

States, districts, and schools should initiate an intentional and reflective process that maintains review and reassessment at key stages with the goal of improving execution and outcomes.

The topics discussed in this chapter are meant to spark further conversations that lead to a sustainable infrastructure for computer science education (see Figure 8.1 for recommended state policies). Due to the increasing demand for computer science and genuine concern from parents, educators, and industry that students are being left behind, reform efforts can easily focus on access and scale at the expense of quality and sustainability. These goals are not mutually exclusive. States, districts, and schools should initiate an intentional and reflective process that maintains review and reassessment at key stages with the goal of improving execution and outcomes.

At a time when only a handful of states have have begun to initiate statewide efforts to implement computer science education, there are no clear models that demonstrate long-term success.

Figure 8.1: Recommended policies that promote and support computer science education

The following recommendations are excerpted from *Making Computer Science Fundamental to K–12 Education: Eight Policy Ideas* (Code.org, 2015).

Define Computer Science and Establish K–12 Computer Science Standards: Standards provide a foundation for aligning all other policies under a coherent vision of computer science for all students.

Allocate Funding for Rigorous Professional Development and Course Support: States and districts can dedicate funding for developing the capacity to teach computer science, including course materials and technical infrastructure.

Implement Clear Certification Pathways: In addition to the development of traditional certification pathways, incentives and expedited, alternative pathways will help address the short- and long-term need for computer science teachers.

Create Incentives at Institutions of Higher Education to Offer Computer Science to Preservice Teachers: States can create competitive grants for schools of education to increase the number of preservice teachers who can teach computer science and incentivize partnership opportunities between local school districts and schools of education to create direct pathways for teachers into high-need school districts.

Establish Dedicated Computer Science Positions in State and Local Education Authorities: Any sustained effort to scale computer science education will require leadership positions at the state and district levels.

Require that All Secondary Schools Offer Computer Science: While kindergarten through 12th grade computer science education is the long-term vision, states and districts can act in the short term by requiring all high schools to offer at least one computer science course.

Allow Computer Science to Count as a Core Graduation Requirement: States that allow computer science to count as a graduation requirement, rather than an elective, see increases in the number of students taking advanced computer science courses and increases in participation from underrepresented minorities.*

Allow Computer Science to Count as an Admission Requirement for Institutions of Higher Education: Policies that do not allow computer science to satisfy an admission requirement can reduce the incentive for students to take computer science, even when it fulfills a high school graduation requirement.

* Review of 2012 Advanced Placement® (AP) data on a per-state basis for AP Computer Science and AP Calculus provided by the College Board.

The conversations started in this chapter must be revisited frequently and informed by lessons learned within each state and across the nation. Rather than thinking about sustainability as establishing reforms that "last and *stay the same*," policymakers and decisionmakers must think of sustainability as establishing reforms that "last and *change*" (Century, 2009, para. 7). Due to the evolving nature of computer science education, implementation plans must be flexible and adaptable, while heeding the public's demand for high-quality computer science experiences for all students.

Incorporating Computer Science into K–12 Systems

Adding computer science to K–12 education requires more than just adding or revising content. A thoughtful approach to including computer science in K–12 instruction will also encompass course and instructional pathways; technical infrastructure; and buy-in from important stakeholder groups, such as administrators, parents, teachers, and support staff. It is important to consider implementation for all students, regardless of race, gender, disability, socioeconomic status, or English language proficiency, as inequities can be propagated when implementing large-scale reforms. Recruitment, expansion, and equity should be actively monitored during the entire implementation process (Margolis, Goode, & Chapman, 2015).

Recruitment, expansion, and equity should be actively monitored during the entire implementation process (Margolis, Goode, & Chapman, 2015).

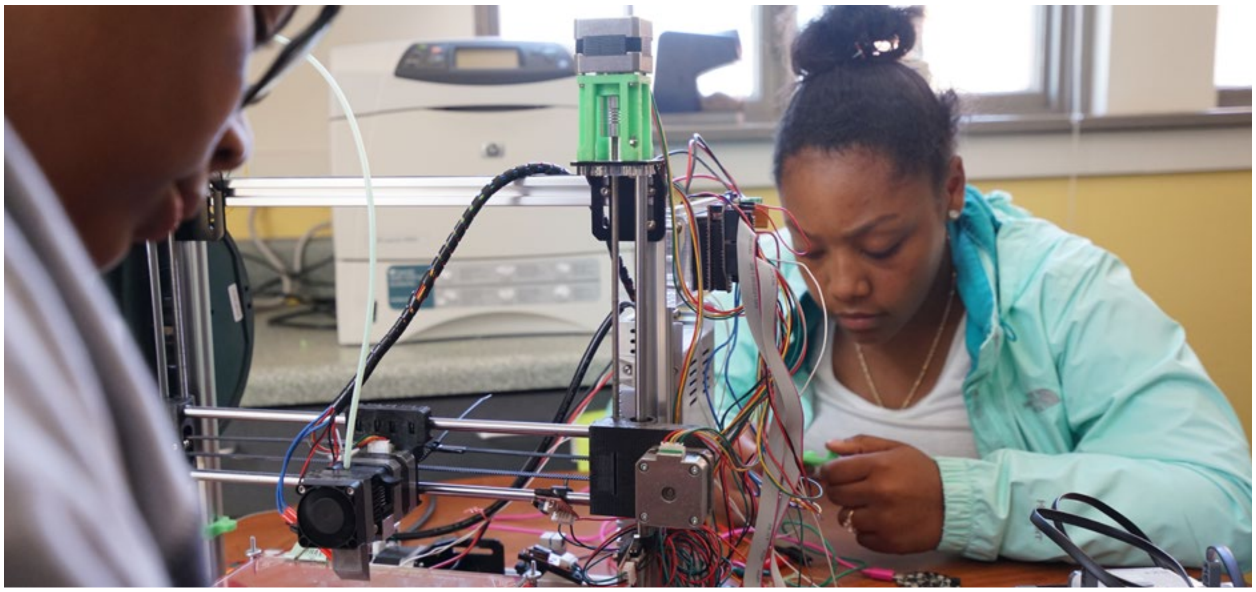
Curriculum

As computer science opportunities increase across the nation, both in school and out of school, students will be entering classrooms with a wide range of experiences in computer science. Some students will have had a great deal of early exposure, while other students will have had none. To meet the needs of all learners, teachers need to be prepared to teach this wide range of students equitably. This section describes some of the many considerations educators will have to take into account as they select and/or develop meaningful curriculum experiences for all students. Although the framework can guide curriculum considerations at a high level, performance expectations (i.e., standards) that integrate the framework's concepts and practices should ultimately guide what happens in the classroom. Formal curriculum developers and content providers can support classroom teachers, especially those new to computer science, by developing high-quality curriculum materials aligned to a coherent K–12 vision.

Uses and limitations for curriculum development

When laying out a comprehensive, K–12 computer science curriculum based on the framework, understanding the breadth of approaches that the framework promotes as well as its limitations is important.

The framework provides broad expectations for K–12 computer science that are designed to be inclusive of diverse curriculum approaches. Schools, districts, and states should look to the framework as a document that provides clarity of content (the "what") but does not dictate implementation (the "how"). For example, different elementary school curricula that teach young students to program, one with physical robots and one within an online, virtual environment, are both able to instill a knowledge of algorithms (K–2.Algorithms and Programming.Algorithms) and help students understand how the computer responds to commands (K–2.Algorithms and Programming.Control).



The concept and practice statements are big ideas that can be used to inform lessons, a series of lessons, or curriculum units. Organizations that use the Understanding by Design curriculum design model can use the framework to inform the "enduring understandings" and "essential questions" for lessons and units (Wiggins & McTighe, 2005). Enduring understandings, similar to the framework's concept statements, illuminate the major, recurring ideas that lend significance and meaning to individual curriculum elements, such as facts and skills, and essential questions reflect the key inquiries that focus curriculum around deeper understanding.

Further, the framework lays out the big ideas that students should understand by the end of a grade band but does not encompass all of the learning that can take place within that grade band. Although the framework focuses on an essential foundation in computer science, schools and teachers are encouraged to create additional supporting material and extend instructional experiences beyond these baseline competencies, while giving special consideration to developmental appropriateness. Schools and classrooms that are already exceeding the expectations in the framework should be encouraged to continue doing so while others can use the framework as an aspiration and starting point.

Figure 8.2: Concepts and practices of the K–12 Computer Science Framework

Core Concepts

1. Computing Systems
2. Networks and the Internet
3. Data and Analysis
4. Algorithms and Programming
5. Impacts of Computing

Core Practices

1. Fostering an Inclusive Computing Culture
2. Collaborating Around Computing
3. Recognizing and Defining Computational Problems
4. Developing and Using Abstractions
5. Creating Computational Artifacts
6. Testing and Refining Computational Artifacts
7. Communicating About Computing

Considerations for curriculum alignment

When selecting or developing curriculum, instructional materials, and computing tools (such as programming environments) that align to the framework, reflecting on the following pedagogical considerations is important.

First, computer science curricula should provide a comprehensive view of computer science informed by the five core concepts and seven core practices of computer science as delineated in the framework (see Figure 8.2). Unfortunately, computer science and programming (or coding) are often considered synonymous in K–12 education. This belief leads to courses that focus only on programming and leave out other areas of computer science that influence our world, such as the Internet, data, and cultural and societal perspectives on computing.

Second, curricula aligned to the framework should be developmentally appropriate per the grade-band progressions in the framework. Ascertaining the extent to which the content aligns to the concepts and practices in the framework, including grade-band placement, is important. Further, because the concepts and practices are framed in K–12 learning progressions, it is important that grade-level curricula fit into a coherent K–12 experience. The framework supports pathways of courses that are offered in every grade or only in particular grades within grade bands; states and districts will decide based on their local context.

Curricula should integrate the concepts and practices into meaningful experiences for students, rather than solely focus on the concepts.

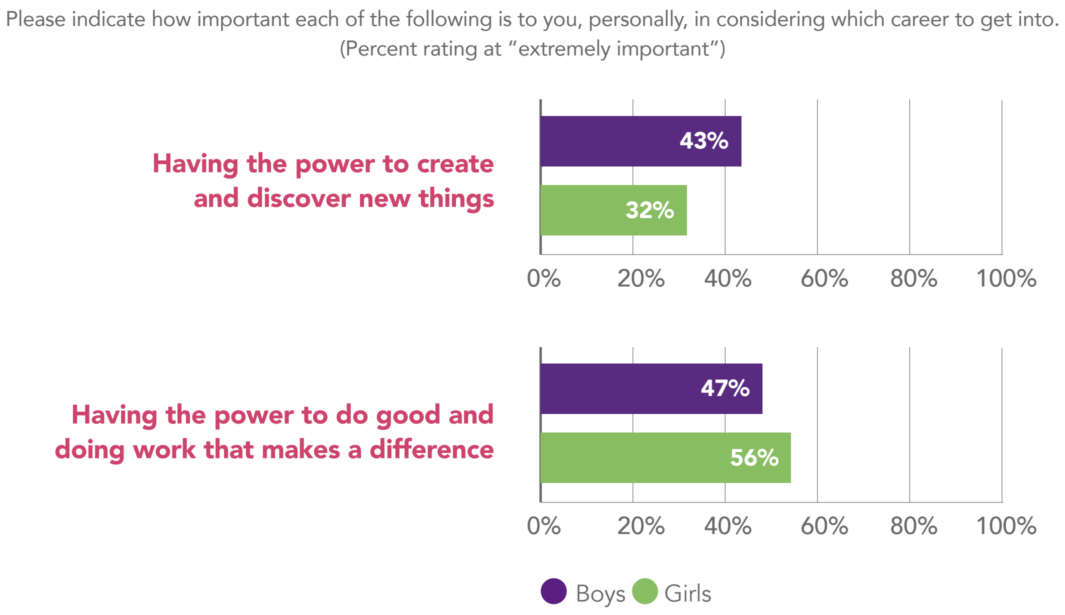
Third, curricula should integrate the concepts and practices into meaningful experiences for students, rather than solely focus on the concepts. Lessons and activities should provide not only content-rich opportunities for students to learn about computer science concepts but also meaningful opportunities for students to engage in computer science practices. Daily instruction and student activities should integrate the computer science practices with one another and with the concepts. Additionally, some concepts may be easily addressed in the same lesson or activity; the descriptive material for each concept statement provides guidance around which ideas connect to one another.

Finally, it is important to consider whether learning a programming environment is the main focus of a curriculum or if the use of programming tools is driven by addressing the concepts and practices. A "tool-first" approach may not provide all of the experiences needed for a student to engage fully in the concepts and practices of the framework. Instead, curriculum should be designed from a "content-first" perspective, in which programming tools, equipment (e.g., robots), and even languages, are a vehicle for learning the concepts and practices, rather than becoming the focus themselves.

Socially relevant and culturally situated

Curriculum should feature projects that offer opportunities to create innovative technologies within socially relevant and culturally situated contexts. In a survey of the career preferences of college-bound students ages 13–17 (see Figure 8.3), students who value having the power to create and discover new things and working in a cutting-edge field also show a high interest in computer science (WGBH Educational Foundation & ACM, 2009). The concepts and practices of the framework provide opportunities for students to engage in these types of activities, such as creating a variety of computational artifacts across multiple areas of computer science.

Figure 8.3: Characteristics of careers that students deem important



Source: WGBH Educational Foundation and the Association for Computing Machinery, 2009

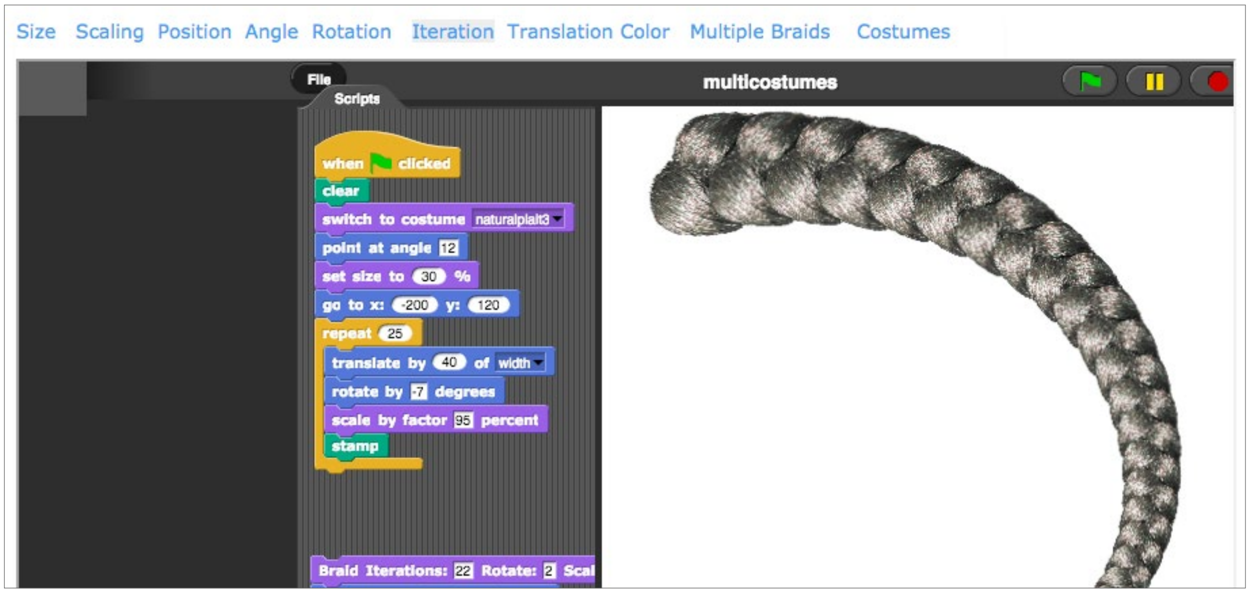
The survey also found that the students who are the least likely to show interest in computer science are those who like working with people in an interconnected, social, and innovative way and those who find making a difference in other people's lives important for a career (WGBH Educational Foundation & ACM, 2009). Further, female students were more likely than males to rate making a difference as "extremely important" in a career. These results may speak more to the stereotypes that students have of computer science as being incompatible with these desires, rather than the reality of how computer science is practiced in careers and the influence it can have on others. Whatever the reason for these differences, curriculum plays a role in addressing misperceptions around careers in computer science, and the framework's concepts and practices can guide experiences that reach all students. For example, the core concept *Impacts of Computing* highlights the influence computing has on people's social and cultural interactions at the community and societal level, addressing students' desires to engage in fields that make a difference in others' lives. The framework's practice of *Creating Computational Artifacts* emphasizes the creation of a "computational artifact . . . to address a societal issue" (P5.Creating Computational Artifacts.2). Additional practices, such as *Fostering an Inclusive Computing Culture*, *Collaborating Around Computing*, and *Communi-*

Students who value having the power to create and discover new things and working in a cutting-edge field also show a high interest in computer science.

cating About Computing, emphasize the social nature of engaging in computer science. Computer science is a project-based, transformational discipline, and students will be more engaged with projects that focus on real-world and community problems for social good (Goldweber et al., 2013).

Projects should acknowledge and build on the rich cultural backgrounds, or "funds of knowledge," students bring to the classroom. Funds of knowledge refers to the "historically accumulated and culturally developed bodies of knowledge and skills essential for household or individual functioning and well-being" (Moll, Amanti, Neff, & Gonzalez, 1992, p. 133). By learning about students and their families, teachers can develop the perspective that "the households of their students contain rich cultural and cognitive resources and that these resources can and should be used in their classroom in order to provide culturally responsive and meaningful lessons that tap students' prior knowledge" (Lopez, n.d., para. 3). The Exploring Computer Science curriculum, also discussed in the **Equity in Computer Science Education** chapter, uses culturally situated design tools (Eglash, 2003) to "encourage students to artistically express computing design concepts from Latino/a, African American, and Native American history as well as cultural activities in dance, skateboarding, graffiti art, and more" (Margolis et al., 2012, p. 76). Through culturally situated lessons (see Figure 8.4 for an example), students build personal relationships with computer science concepts and practices and ultimately feel like computer science is more relevant to their lives. The use of culturally situated computing contexts has additional benefits, such as the possibility of counteracting barriers to increasing minority participation in computing. Seeing how aspects of a student's culture are based in computing concepts and practices can reduce identity conflict, in which students feel like their personal or cultural identity is incompatible with participation or academic success in a subject (e.g., "my people don't do computing") (Eglash, Bennett, O'Donnell, Jennings, & Cintorino, 2006).

Figure 8.4: Example of a culturally situated computing activity



```
when [flag] clicked
clear
switch to costume naturalplait3
point at angle 12
set size to 30 %
go to x: -200 y: 120
repeat 25
    translate by 40 of width
    rotate by -7 degrees
    scale by factor 95 percent
    stamp

Braid Iterations: 22 Rotate: 2 Scal
```

Socially relevant and culturally situated contexts offer opportunities for integrating computer science with other subjects. Teachers should look to their students' communities for examples of projects and applications of computer science education that can be aligned to the framework. However, these projects should be carefully crafted and scaffolded for beginners. Research in introductory undergraduate courses suggests that developing authentic, humanitarian-focused projects that are motivating to novices is hard, as humanitarian problems are complex by their very nature (Rader, Hakkarinen, Moskal, & Hellman, 2011). On the other hand, some researchers have noted that novices tend to favor assignments that they perceive as easy and fun and shy away from problems that are too open-ended (Cliburn & Miller, 2008). Thus, with consideration toward students' experiences with computing and their ability levels, socially relevant and culturally situated curricula hold promise for engaging all students.

Assessment

Assessments are used in multiple ways in K–12 education. This section focuses on assessment at the classroom level, rather than the high-stakes testing that can dominate the conversation of reform in other subject areas. Generally, classroom assessment can be formative or summative. Formative assessment is used during classroom activities to modify instruction or provide students with immediate feedback about their learning or progress, whereas summative assessment is used to evaluate or measure student learning at the end of a period of instruction (Black, Harrison, Lee, Marshall, & William, 2003). End-of-course summative assessments and programmatic exams are currently rare in computer science outside of Advanced Placement® (AP), International Baccalaureate, or industry-recognized certification exams in career and technical education (CTE) programs. There are several important facets of classroom computer science assessments to consider: the use of authentic tasks, the breadth of concepts being assessed, and the special role computers can play in delivering instruction and measuring performance.

Project-based and portfolio-based assessment methods are critical for authentically measuring performance in the computer science classroom. Performance tasks are typically more flexible than traditional assessments that seek one solution or answer to an assessment question. These tasks allow students to demonstrate their understanding in multiple ways that highlight their creativity, interests, and understanding. Consequently, these assessments can provide an educator with a richer understanding of students' knowledge and reasoning. These nontraditional assessments can be useful in measuring students' use of algorithms, computational thinking, and problem solving—which are generally hard to measure with multiple-choice questions. Practices identified in the framework, such as *Communicating About Computing, Collaborating Around Computing*, and *Creating Computational Artifacts*, are key to emphasize in any computer science performance task. The AP Computer Science Principles course employs performance tasks and accompanying rubrics that can be freely accessed, which teachers may find useful as a starting point for designing their own assessments.

Assessment should reflect multiple aspects of computer science as defined by the five core concepts and seven practices of the framework. Most computer science assessments focus primarily on programming (Yadav et al., 2015) and ignore other aspects of computer science, such as data analysis or the impact of computing on society. Multiple concepts can be addressed simultaneously. For example, teachers can assess students' ability to analyze the advantages and disadvantages of different encryption algorithms, which addresses the idea of algorithmic performance (*Algorithms and Programming*) as well as cybersecurity (*Networks and the Internet*). Even when programming is the focus, students should be assessed on not only their ability to write the program but also their ability to communicate the product's significance and development process (*Communicating About Computing*), including the collaboration among members (*Collaborating Around Computing*). For example, students can submit planning documents used to produce the program, do a presentation on the impact that their program will have on a target audience, and write a reflection on how the team worked to put the program together.

Compared to other subjects, computer science provides a unique opportunity for taking advantage of online learning and computerized assessment while maintaining an authentic experience for demonstrating performance. Platforms dedicated to computer science allow students to create programs such as games, apps, and simulations within an environment that also collects data, analyzes achievement, and communicates progress to both students and teachers. These platforms have the ability to naturally integrate instruction, practice, and assessment. Online learning and assessment platforms also have the potential for reaching students in rural school districts, which are often at a disadvantage in finding teachers in high-need subject areas, such as computer science.

Course and Instructional Pathways

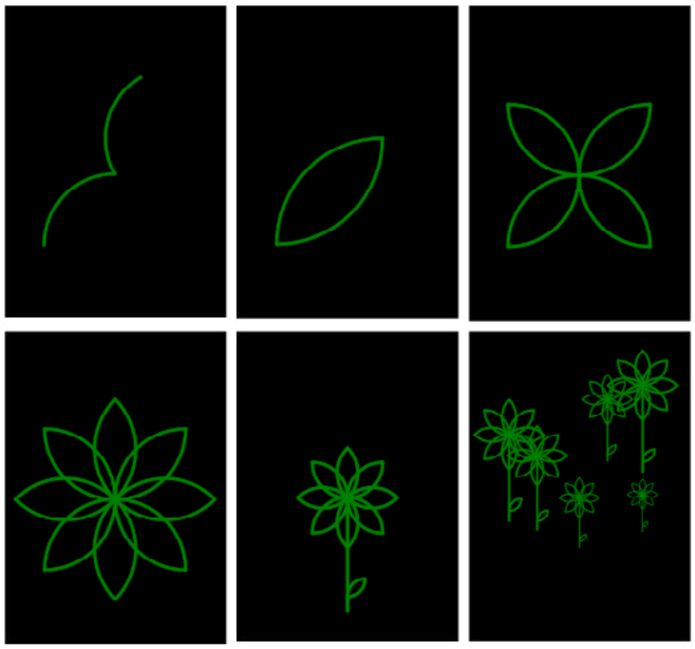
The framework describes a K–12 experience in computer science that builds sophistication over time. Course and instructional pathways that treat computer science as a discipline, as opposed to an individual elective, are required to implement this vision. Courses and curriculum do not exist in an instructional vacuum and should take into account the concepts and practices as laid out across the four grade bands in the framework. This section explores options for building a K–12 instructional pathway in computer science.

Integrated computer science courses

Rather than adding to educators' already full plates, computer science can aid the current movement toward interdisciplinary education. Classrooms can infuse computer science into practically every other subject area, including mathematics, science, English language arts, world languages, social sciences, fine arts, service learning, health and physical education, and CTE programs. Throughout the framework, opportunities are provided for integration and application within other content areas. For example, concepts within *Computing Systems*, when combined with the framework's practices, could be incorporated into fine arts as students produce works of art that include digital music, animation, and lighting systems. The organization of networks in *Networks and the Internet* can be used to reinforce ideas of networks in other content areas, such as the connections between characters in a story in English language arts or the ways that contagious diseases are spread through populations in health and physical education. Programming can assist in the collection or representation of data in mathematics or science classrooms, tying together the *Data and Analysis* and *Algorithms and Programming* core concepts. And the *Impacts of Computing* concept statements can be reinforced as students consider the effects of computing in world languages (such as translation software), service learning (how technology can allow people in different locations to connect, communicate, and collaborate), or social sciences (such as the interaction of social media and political movements or the use of technology to monitor communications). The concepts and practices of the K–12 Computer Science Framework can be integrated into other content areas in numerous other ways as well.

There is a history of rich examples of age-appropriate, integrated computing experiences at the K–8 level (e.g., Papert, 1980). In his classic book on children and computing, *Mindstorms*, Seymour Papert describes a situation that reflects common interactions between two children who are working and playing with the Logo programming language. The experience begins with a student wanting to draw a flower on the computer screen. One student asks the other whether she has any pre-existing programs they can use to draw the petals. They modify a program that draws an arc multiple times until they realize how to use their understanding of angles to create a full petal and then multiple petals. Then, the students proceed to create a garden of flowers by repeating the procedure for drawing a single flower and using variables to randomize the size and location. This example, shown in Figure 8.5, illustrates the integration of mathematics, programming, and play.

Figure 8.5: An example of the iterative process students could use to create a garden of flowers



Some recent efforts have incorporated computer science into mathematics and science classrooms. Bootstrap and Project GUTS (Growing Up Thinking Scientifically), two programs sponsored by the National Science Foundation, demonstrate that students can learn computer science concepts and practices within the overarching goals of their algebra and science classes. Bootstrap (2016) uses video game programming as an approach to teach topics such as the Pythagorean theorem, distance formula, and linear equations. Project GUTS (2016) incorporates modeling and simulation projects into Earth, life, and physical science to explore topics such as water pollution, ecosystems, and chemical reactions. The experiences within these programs are modular and can range from a couple lessons to several weeks of content. While the effects on instructional focus and subsequent performance on assessments remain under exploration, many similar programs exist around the nation (e.g., Algorithmic Geometry, 2016; UC Davis C-STEM Center, 2016), reinforcing this approach as a model that can be used to expose students to computer science by using pre-existing resources.

It should be noted that integration is a matter of implementation at the school or district level, not within standards at the state level. Computer science can be embedded within pre-existing subjects, but state-level standards for computer science, which clarify specific disciplinary expectations, should remain a discrete set of standards or strand, rather than be mixed with the performance expectations of other subjects. A few states have successfully created computer science standards as discrete strands within a larger set of standards. For example, Indiana has created a computer science strand within its overall science standards (Indiana Department of Education, 2016), and Massachusetts has combined Digital Literacy and Computer Science Standards with discrete strands for each that build off one another (Massachusetts Department of Elementary and Secondary Education, 2016). The Massachusetts standards delineate the differences between computer literacy, digital citizenship, and computer science. These approaches allow schools and districts to decide how (curriculum), when (grade level), and/or where (subject) computer science is integrated. Additionally, this standards-level integration does not necessarily mean computer science is embedded into other subjects, as it is equally viable that computer science instruction exists as independent courses that integrate content from other disciplines.

Independent computer science courses

Offering computer science as independent, standalone courses has the benefit of not affecting instructional time in other subjects. In elementary school, computer science can exist as a special class, similar to music, art, and physical education, through which students rotate during their weekly schedule. In middle school, computer science can be a dedicated semester- or year-long experience at a particular grade level or available at all grade levels. In high school, computer science can be taught in introductory courses; AP courses; and specialized courses such as cybersecurity, game design, or robotics. Unfortunately, when computer science is offered as an independent course, it is often as an elective. Computer science as an elective presents a disadvantage compared to integrating computer science into a subject that all students take, such as mathematics or science, because fewer students will be exposed to computer science. Furthermore, students who have preconceived misperceptions about computer science may self-select out of computer science before they even attempt a course. It should be noted that standalone computer science courses and interdisciplinary integration are not mutually exclusive—a computer science course can feature projects that are couched within the context of other disciplines, such as math, science, and art.

Revising technology and computer literacy courses

One of the most natural places to increase computer science participation is through pre-existing technology education credits and courses. Many states and districts have some type of technology or computer literacy graduation credit, which can be modified to allow computer science to count. For example, in 2015, Maryland revised its one-credit technology graduation requirement and allowed computer science courses to satisfy this requirement. Previously, only courses that fit a perspective on technology that included engineering, manufacturing, transportation, agriculture, or medicine, were allowed to count (Maryland State Department of Education, 2005). Before 2015, some Maryland school districts had already taken the initiative to integrate computer science into their general technology education courses, leading to an increase in the percentage of underrepresented minorities and females taking computer science compared to districts that still considered computer science an elective (Wilson & Yongpradit, 2015). Since the change to the technology graduation requirement, Maryland has reported a dramatic increase in computer science enrollment (CTE Maryland, 2016). Whether or not technology education courses count as a graduation requirement, revising outdated courses to focus on computer science is practical and appropriate.

Building the Pathway

With the introduction of a new subject into the K–12 space, schools will need to develop plans for gradual implementation. These plans should account for early years of implementation in which students in the upper grades will not have had the basic fundamentals and may not learn the full progression of concepts and practices before graduation.

Schools may choose to convene working groups with individual disciplines (e.g., math, science, humanities) to determine if any of the concepts or practices are already covered or strongly aligned with current curriculum. The framework provides opportunities for schools to think critically about how to implement computer science, such as via collaborative lessons between disciplines or integration into other disciplines.

Considerations for Staffing Classes

Districts and schools must consider how to develop and staff new computer science courses with limited staffing resources; what to look for when hiring computer science teachers; and how to increase the pool of computer science teachers at a time when there is a shortage across science, technology, engineering, and math subjects (Barth, Dillon, Hull, & Higgins, 2016).

Much the same way they bring together vertical teams for traditional subjects, such as math and science, schools or districts may want to bring together multigrade vertical teams for computer science. These teams can communicate positive approaches, lessons, or examples that are particularly engaging to students and areas of strength or weakness in student learning. As designers of a new subject area, these vertical teams can become professional learning communities, exploring the computer science content or resources from higher or lower grades to develop better understanding of expected student outcomes.

School systems that have been working toward district-wide computer science provide an example of the level of collaboration required to make such a reform possible. From the beginning of their effort, Broward County Public Schools in Florida, the seventh largest school district in the nation, formed a computer science implementation team consisting of representatives responsible for certification, professional development, media communications, school leadership, career technical education, and science, technology, engineering, and math (STEM). This team set up a structure for interdepartmental collaboration and the development of a comprehensive district strategy, resulting in a sharp increase in student participation. Before implementation, in school year 2013–14, nine high schools and no middle or elementary schools offered computer science, serving a total of 240 computer science students. After two years, 33 high schools, 34 middle schools, and 113 elementary schools were offering computer science. More than 38,000 students took computer science in 2015–16 (Broward County Public Schools, 2016). This dramatic change required training close to 1,000 teachers through a partnership with professional development providers, higher education institutions, and local community stakeholders (White House, 2016). Despite increased access, districts must continually temper growth with quality, evidence-based implementation. For example, Broward County Public Schools is currently researching and evaluating a model for integrating computer science and STEM into the elementary schedule.

Overall, the development of a K–12 sequence will be an ongoing and iterative process informed by an emerging computer science pedagogy that is inclusive of all learners (Snodgrass, Israel, & Reese, 2016). Schools and districts will need to be flexible in their implementation and potentially explore multiple options before deciding upon the right fit for their students and community.

The following sections highlight different models at the K–8 and high school levels (see Figure 8.6). These different models can be mixed and matched to create custom K–12 pathways (see Figure 8.7).

Elementary and middle school models

Implementing computer science instruction at the K–8 level is typically more flexible than in high school. Potential models include instructional units dedicated to computer science within general technology and media arts classes, dedicated weekly computing classes offered as electives, and integration of computer science instruction into other content areas. These courses could be taught by a variety of teachers, such as elementary classroom teachers, subject area teachers in school (e.g., mathematics, science, technology, music, art, library/media arts), or dedicated computer science teachers. Regardless of the instructional delivery model, attention should be given to aligning those instructional experiences with the framework's progressions.

High school models

The content of the framework is intended for all students, and although some districts have been motivated to create a computer science graduation requirement, all high schools should offer at least one rigorous computer science course. More than half of all high school seniors do not attend a high school that offers any computer science courses (Change the Equation, 2016). According to analyses of data from the National Center for Education Statistics, the percentage of high school seniors taking computer science in 2015 (22%) (Change the Equation, 2016) was less than in 1990 (25%), with a low of 19% in 2009 (Nord et al., 2011). States and districts that are looking to take a first step toward computer science for all students should consider starting by offering computer science in every high school.

All high schools should offer at least one rigorous computer science course.

Over the course of the rollout of a K–12 pathway in computer science for all students, high school programs will have to adapt as more students enter high school with prior experiences from elementary and middle school. High school courses that are designed for students who have had no prior computer science experience will continue to play an important role in catching up students who, for example, opted out of elective courses in earlier grades or transferred between school districts.

As students who gained introductory experiences choose to continue their study of computer science, schools and districts may choose to include more advanced course offerings and pathways, such as courses that can provide college entrance credit, career and technical preparation, or AP courses. The content of these courses will be more advanced than that of the framework, but the framework's organization of core concepts and practices can provide a guiding structure. For example, AP, by definition, is considered college-level coursework, and the framework provides the fundamental understandings that precede these experiences. Conversely, some existing courses, such as AP Computer Science A, focus primarily on algorithms and programming, and schools will need to implement supplemental courses or curricula to address the full set of concepts in the framework's 9–12 grade band.

States and districts must consider whether computer science lives within an academic pathway, a CTE pathway, or both, as the choice will affect access, funding, and course content. In 2010, the Association for Computing Machinery's report *Running on Empty* observed,

> A major contributing factor to the confusion about computer science education is that computer science or "computing" courses are organized into various departments within schools. For example, some are placed in the mathematics or science departments and some are within the vocational education departments. When computer science courses are placed within vocational education, they are rarely part of the "core" curriculum a student must take. Further, the curriculum for these courses tends to be focused on broader IT or technology skills rather than deeper computer science concepts. (Wilson, Sudol, Stephenson, & Stehlik, 2010, p. 15)

CTE courses, with their career focus, provide great opportunities for students who want to explore specializations within computer science, such as cybersecurity, database administration, and software engineering. There is room for both academic and CTE classification of computer science. For example, the early, foundational courses in a CTE program of study can be dual-coded as part of the CTE pathway as well as a math, science, or technology credit. CTE systems differ from state to state, and it is recommended that states provide clear guidance to districts so that as many students as possible can have foundational computer science experiences and participate in CTE pathways.

Foundational courses in a CTE program of study can be dual-coded as part of the CTE pathway as well as a math, science, or technology credit.

Example K–12 pathways

Ultimately, course pathways will be driven by the framework and/or standards that are developed based on the framework. The number of standards, whether they are grade-specific or grade-banded and voluntary or mandatory, will affect the choice of pathway. In the examples pathways in Figure 8.7, a computer science experience can range from a few hours a week to a semester- or year-long course. For the purpose of the example pathways, it is assumed that elementary school includes Grades K–5, middle school covers Grades 6–8, and high school covers Grades 9–12. The different models in each grade band are organized by an estimate of the total amount of focused computer science instructional time that the model may allow, from least to greatest. The examples within each grade band are not mutually exclusive; many options can be combined to create additional avenues for computer science instruction.

Figure 8.6: Options for implementing computer science

ELEMENTARY SCHOOL IMPLEMENTATION EXAMPLES

- Integrated into the general classroom
- Integrated into an existing special (e.g., media arts, computer lab)
- Independent special (similar to Science, Music, Art; kindergarten to Grade 5)

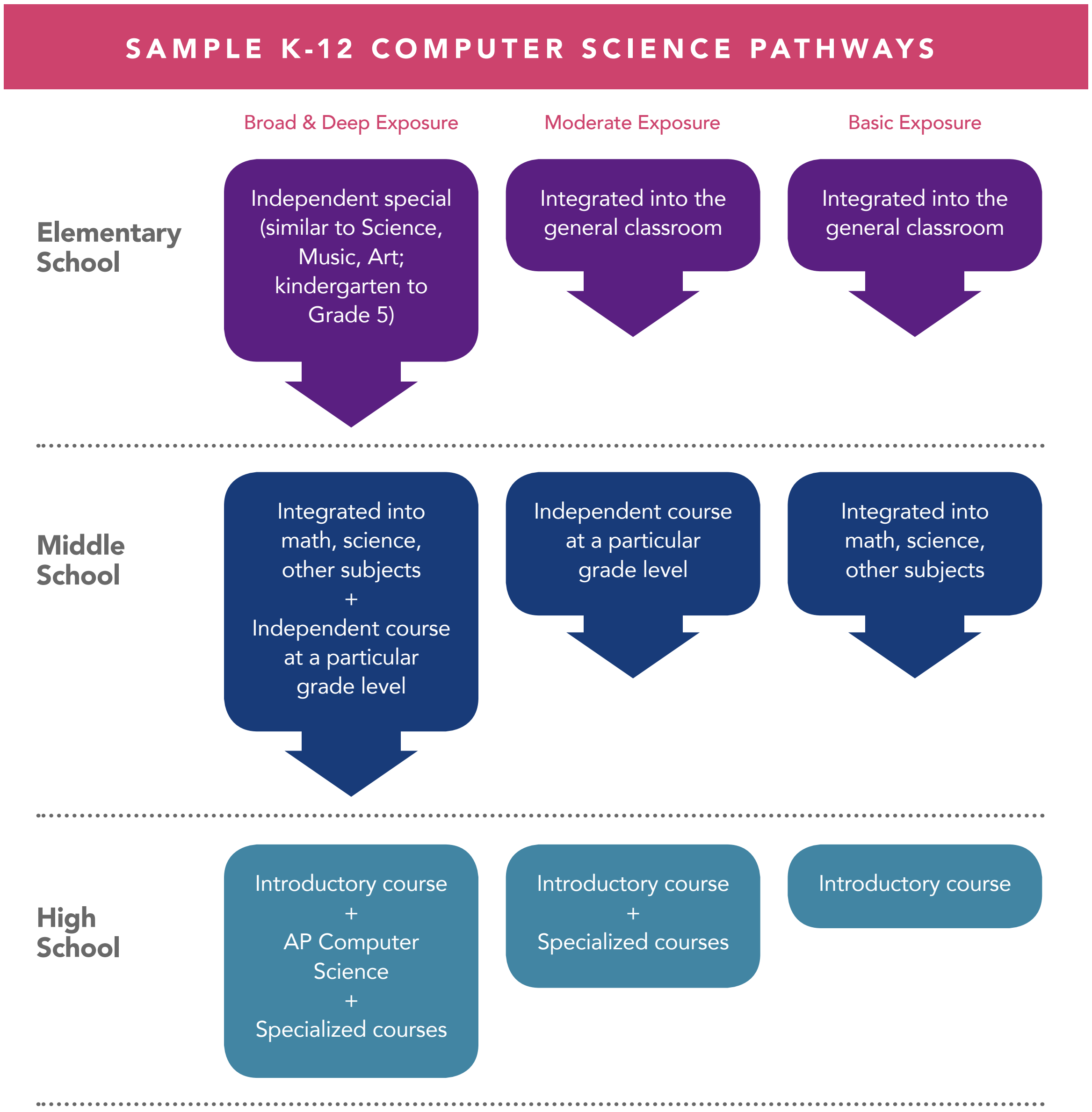
MIDDLE SCHOOL IMPLEMENTATION EXAMPLES

- Integrated into math, science, or other subjects
- Independent course at a particular grade level or at all grade levels

HIGH SCHOOL IMPLEMENTATION EXAMPLES

- Integrated into math, science, or other subjects
- Introductory course
- Advanced course (e.g., honors, AP)
- Specialized courses (e.g., game design, cybersecurity, networking, robotics)

Figure 8.7: Multiple pathways for implementing K–12 computer science


SAMPLE K-12 COMPUTER SCIENCE PATHWAYS
Broad & Deep Exposure
Moderate Exposure
Basic Exposure
Elementary School
Independent special (similar to Science, Music, Art; kindergarten to Grade 5)
Integrated into the general classroom
Integrated into the general classroom
Middle School
Integrated into math, science, other subjects + Independent course at a particular grade level
Independent course at a particular grade level
Integrated into math, science, other subjects
High School
Introductory course + AP Computer Science + Specialized courses
Introductory course + Specialized courses
Introductory course

Technical Infrastructure

Modern computer science education does not often require complex hardware setups or software configurations, although some basic technical conditions must be in place. The availability of computers is the greater concern and should be addressed in a district's or state's technology plan by involving a school system's office of technology as a key stakeholder in computer science education initiatives. Some common strategies at the classroom level for dealing with a lack of computers are having students work collaboratively on devices, having a center for computing (particularly at the elementary level), or rotating students through a computing experience within a weekly schedule.

While some computer science education curricula and programs require locally installed resources, many computer science education programs and curricula now exist online. Using programs and curricula that are online allows schools to cover much of the curriculum with browser-capable devices, including netbooks and, in some cases, tablets. Using a web-based solution to support computer science curricula does require a robust network connection, but unfortunately, 23% of school districts still lack fast and stable Internet connectivity (Education Superhighway, 2015). Schools will want to weigh their options based upon their own current technological resources.

Stakeholders

Although physical infrastructure is important, equally important is cultivation of community stakeholders who can support computer science implementation. Computer science is a relatively new discipline in K–12 education, so policymakers and administrators at educational institutions may be unclear about appropriate content for either standalone computer science classes or integrated experiences within other content areas. They may also have misconceptions about the target audience for computer science education. For example, some stakeholders may believe that computer science should be available only as enrichment opportunities for typically high-achieving students, rather than for the entire student population. However, just like other subject areas, all students are capable of learning the basics of computer science, and it is widely agreed in the computer science education community that the fundamentals of computer science are essential for developing critical thinking skills and understanding the technology that people interact with daily.

"There is deep and widespread confusion within the states as to what should constitute and how to differentiate technology education, literacy and fluency; information technology education; and computer science as an academic subject." (Wilson, Sudol, Stephenson, & Stehlik, 2010, p. 9)

Community and Business Partners

The school community should be educated both to inform and to build support for computer science implementation. School events such as back-to-school nights, parent-teacher conferences, school board meetings, or academic showcases like science fairs can be useful for communicating the specific nature of computer science education in a particular school or district and for engaging local elected officials. Highlighting and featuring student work is an excellent opportunity to spark discussion and address misconceptions about what computer science is, who it is for, and when students should be learning it. Many computer science education organizations have resources and ideas for outreach and community events.[2]

2 For example, the National Center for Women & Information Technology (NCWIT) has Outreach-in-a-Box kits, and Code.org has Hour of Code event ideas.

A school leader interviewed on LeadCS.org (2015) explains the importance of involving the business partners when implementing new computer science programs:

> The whole idea is the neighborhood—to get the entire buy-in of the neighborhood where the school is in. At some point you are going to need help in funding, whether that's through grants or local businesses here or there. [Get] that buy-in ahead of time so that if you do have to go ask for funding, they'll be at the table and know ahead of time that the request doesn't just come out from nowhere. (p. 2)

Other than funding, community and business partners can provide guest speakers, teaching assistants, field trip and camp opportunities, and even internships.

Informal Education

Informal education organizations are essential to the computer science education ecosystem and should be included as critical stakeholders in state and district implementation efforts. Organizations participating in informal education networks, such as the Statewide Afterschool Networks, can play a large role in supporting partnerships with industry and higher education institutions, including organizing large-scale professional development opportunities.

Informal education is essential to the computer science education ecosystem.

Informal education offers young people opportunities to increase their interest in computer science, develop relationships with role models, and build capacity for engaging in a variety of computer science activities. Informal education is used here to refer to organizations that provide extracurricular, out-of-school, afterschool, camp, or other learning environments beyond the scope of the school day, including the developers of games and apps on the Internet and mobile devices. Informal education can provide increased opportunities for project-based learning, as demonstrated by the maker movement in afterschool programs. Informal education provides a natural setting for socially relevant and culturally situated activities, such as community impact projects. Informal education also allows students to amplify the foundation taught in school and explore a vast array of novel, specialized topics beyond the scope of formal education. These activities can be guided by the concepts and practices of the framework to provide a link to in-school learning.

Informal education programs also have the potential to significantly increase the number of students and the diversity of students who are exposed to computer science. By one estimate, 7 million students have access to afterschool STEM learning opportunities, based on a survey of parents (Afterschool Alliance, 2015, p. 7). Further, parents from African American, Hispanic, Asian, and Caucasian populations report that children participate in afterschool STEM activities at similar rates (approximately 80%), and parents of males and females report similar levels of access (Afterschool Alliance, 2015). Some organizations focus primarily on serving female (e.g., Girls Who Code, 2016; National

Center for Women & Information Technology, 2016; National Girls Collaborative Project, 2016) and minority (e.g., Black Girls Code, 2016; Level Playing Field Institute, 2016) students and a large percentage of participants attending these programs express interest or plans to study computer science at the postsecondary level (Girls Who Code, 2016). The framework has potential for increasing coherence between formal and informal computer science education, uniting both under a common vision.

Teacher Development

Teacher development is a critical part of the computer science education infrastructure. Teacher development is used here as a broad term that includes preservice teacher preparation, certification, licensure, and ongoing professional development. It concerns stakeholders in higher education, state agencies, school districts, and organizations that provide professional development.

The K–12 Computer Science Framework can inform the design of teacher development programs. The concepts and practices help program designers and novice teachers organize and understand the breadth of knowledge in computer science. The framework was intentionally written with the understanding that it would be used by both teachers who are familiar with computer science and teachers who are new to computer science. This section details the role that the framework plays in teacher development and describes policies that support and promote the development of computer science teachers.

Preservice Teacher Preparation

There is a nationwide lack of preservice teacher preparation programs in computer science. Most states do not have a single university teacher preparation program in computer science. For example, in 2014–15, only 51 computer science teachers, across all 50 states, graduated from teacher

preparation programs with explicit certification in computer science (Title II, 2016). UTeach, a STEM teacher preparation program that operates out of 44 universities, has noted that of all of its subjects, increasing the pool of computer science teachers has proven to be the most difficult (Heitin, 2016). Colleges and universities looking to build capacity to offer special courses for computer science education and create new programs face a difficult task, with challenges such as funding, time, staffing, enrollment, and expertise. The current landscape of computer science teacher preparation has reflected a paradoxical challenge: secondary schools cannot offer computer science classes because they cannot find prepared teachers, and preservice programs do not prepare teachers because there are not enough computer science classes for potential graduates to teach.

Using the framework to guide teacher preparation content

The framework can be used in a number of ways to guide teacher preparation programs as they help preservice teachers develop the content and pedagogical knowledge necessary to meet the needs of a diverse student population. This section describes how the framework informs teacher preparation programs, including the organization of courses, pedagogical content knowledge, and pedagogical practice.

The concepts and practices provide an organizing structure for framing preservice teachers' content knowledge, but the depth of coursework should not be limited by the student expectations in the framework. The five core concepts can be used to determine which courses are required, potentially necessitating coursework outside of the traditional computer science pathway (which often focuses on programming). For example, computer science courses may need to integrate content from data science or ethics courses. These hybrid courses provide an opportunity for collaboration among different departments in higher education institutions to fulfill requirements in multiple majors. This approach can decrease the need for specific computer science faculty and increase exposure to computer science for students in other majors.

In addition to gaining subject matter knowledge and general pedagogical skills, preservice computer science teachers need to develop pedagogical content knowledge specific to the teaching of computer science (Tucker et al., 2006). At the intersection between subject matter knowledge and pedagogy, pedagogical content knowledge "includes an understanding of what makes the learning of specific topics easy or difficult: the conceptions and preconceptions that students of different ages and backgrounds bring with them to the learning of those most frequently taught topics and lessons" (Shulman, 1986, p. 9). Teaching methods courses offer the environment in which pedagogy and pedagogical content knowledge can be developed. Explicit modeling of computer science pedagogy and opportunities for exercising pedagogical content knowledge will provide teachers strategies for working with all students, such as those who may struggle with programming. The framework's learning progressions can be used to support teachers' development of pedagogical content knowledge because the progressions describe key conceptual milestones and show how students' understanding can build over time. Focusing preservice course activities around the framework learning progressions provides a context in which preservice teachers can identify potential misconceptions and ideas that students may find difficult to comprehend.

Preservice preparation programs can be inspired by the framework's vision to develop students who understand the world through the lens of computer science and can apply computer science to a variety of interests and disciplines. To align to this vision, teacher preparation programs should encourage preservice teachers to connect computer science to a variety of personal, practical, and social contexts. For example, course activities could include critical discussion of current events that demonstrate the pervasiveness of computing and the ways these events connect to the relevant concepts and practices in the framework. Preservice teachers could also gain experience in producing a wide range of real-world computational artifacts that are personally relevant and meaningful—opportunities they will eventually provide for students.

Realizing this vision of the computer science classroom also requires teachers who are able to integrate the concepts and practices of the framework into meaningful learning experiences. Preservice teachers should develop facility in combining practices—such as promoting the needs of diverse end users, soliciting and incorporating feedback into the design process, and defending design decisions—with concepts across the five core concepts in the framework. The integration of the concepts and practices can also serve as focal points for projects, lessons, and activities and can provide contexts for examining different pedagogical approaches.

Structuring teacher preparation programs

Increasingly, teacher preparation programs at universities are creating innovative programs to expose preservice teachers to aspects of computer science so they can integrate it into their instruction or add computer science as another certification (e.g., DeLyser, 2016). This section describes how teacher preparation programs can be structured to prepare teachers to teach multiple content areas or to integrate computer science into other content areas. Partnerships between these programs, school districts, and state departments of education are critical for increasing the number of computer science teachers.

An option for preparing preservice teachers in computer science without creating full preparation pathways is to add computer science to teacher preparation programs for other subjects. For example, Illinois State University (2016) has a computer science education program that can be added onto a mathematics education major, resulting in dual certification. UTeach programs at The University of Texas at Austin (UTeach College of Natural Sciences, 2016) and its 43 partner universities are designed for computer science and other STEM majors to obtain teaching licensure without adding more time to their undergraduate degree plans. These types of programs prepare their graduates to teach two subjects, which can make those graduates attractive to school districts who need teachers who can teach one or two computer science courses. Depending on the program and with proper planning, students can graduate with a computer science certification with few to no additional credits.

Incorporating computer science content into a required course for all education majors, such as a learning theory or an educational technology course, can expose all preservice teachers to computer science. For example, the education department at Purdue University incorporated a one-week module on computational thinking into a required course for elementary and secondary majors (Yadav, Zhou, Mayfield, Hambrusch, & Korb, 2011). The computational thinking content replaced a module on problem solving and critical thinking by addressing similar objectives within a computer science context. By learning computational thinking concepts, teachers can be better prepared to integrate it into their teaching and better able to articulate broader uses of it as a problem-solving tool in other disciplines (Yadav, Mayfield, Zhou, Hambrusch, & Korb, 2014).

Incorporating computer science into a required course for all education majors can expose all preservice teachers to computer science.

State and district departments responsible for teacher hiring are influential in increasing the pool of qualified computer science teachers. As suggested earlier, states and districts should communicate their computer science implementation plan to engage a variety of education stakeholders. This collaboration is particularly relevant when hiring more computer science teachers, as partnerships with higher education institutions can lead to joint strategies targeting computer science, ongoing professional development opportunities, placements for professional internships, and a candidate pool for new teaching positions in computer science (Barth et al., 2016). Partnerships between states or districts and higher education can facilitate communication about projected vacancies in computer science and the demand for graduates with experience in computer science education. For example, districts that are integrating computer science into pre-existing courses should make preservice programs aware that teacher candidates who have had experience with integrating computer science will be favored in the hiring process (see Figure 8.8 for a sample interview activity). These partnerships can also benefit districts via the matching Teacher Quality Partnerships Grants provided through Title II of the Higher Education Act (2008) that are awarded to colleges of education who work with high-need school districts to improve teacher preparation.

Closely related to preservice teacher preparation, the next section discusses the computer science teacher certification landscape and offers recommendations for developing certification pathways.

Figure 8.8: Sample interview activity based on the framework

The framework can be used to initiate conversations during hiring or reassignment to assess teacher readiness.

Choose one of the concept statements from the framework and ask candidate teachers how they would integrate it with one of the seven practices in the classroom. Ask them to describe a project that they have facilitated in a classroom that would demonstrate mastery of one of the concept statements.

Certification

Practical, straightforward, and clearly communicated computer science certification pathways that are also supported by teacher preparation programs are a key contributor to the sustainability of computer science implementation. Unfortunately, based on an analysis of state data reported by Code.org (2016a), only 27 states offer a computer science certification option. The Computer Science Teachers Association has reported that of the states that offered computer science certification in 2013, 12 did not require it to teach (CSTA Certification Committee, 2013). The lack of certification pathways means that many teachers currently teaching computer science are certified in another subject. Similar to the preservice program paradox, the certification paradox is that "[s]tates are hesitant to require certification when they have no programs to train the teachers, and teacher training programs are hesitant to create programs for which there is no clear certification pathway" (Stephenson, 2015, para. 2).

The 2013 report *Bugs in the System* summarizes a certification landscape in which prospective computer science teachers are frustrated by unclear processes, preparation programs are few, and administrators are confused about what computer science even is (CSTA Certification Committee, 2013). The report states:

> This report on computer science teacher certification in the 50 states and the District of Columbia makes it clear that the certification/licensure processes for computer science are deeply flawed. In Florida, for example, would-be computer science teachers have to take a K–8 computer science methods course that is not offered in any teacher preparation program in the state. Prospective computer science teachers often meet difficulty in determining what the certification/licensure requirements are in their own states because no one seems to know. Add to that frustration the confusion that persists around what computer science is and isn't and where it fits in K–12 academics, and it's astounding that professionals with such valued expertise persevere to become computer science teachers. But they do. (Executive Summary)

Recent reforms at the national level, coupled with demand for computer science education from students, parents, and business leaders, provide opportunities and motivation for states to institute certification pathways. The STEM Education Act of 2015 expanded the definition of STEM to include computer science. This was followed by the Every Student Succeeds Act (2015) including computer science as part of a "well-rounded education" alongside subjects such as English language arts, mathematics, and science. As a result, STEM endorsement programs in the state of Maryland have used the clarification as an opportunity to revamp their programs to include computer science content. This type of modification to the STEM endorsement could be considered by other states that support and promote STEM education via certification or teacher development.

The Every Student Succeeds Act (2015) includes computer science as part of a "well-rounded education" alongside English language arts, mathematics, and science.

States implement computer science teacher certification in different ways. Some states have a full computer science teacher certification, and other states have a computer science endorsement that certified teachers can obtain in addition to their primary certification. For states seeking to develop or expand their computer science teacher certification pathways, multiple options are available. Potential ideas under development by Code.org (2016b) include suggestions for what can happen immediately, in the short term, and in the long term. Although creating a full certification pathway provides long-term sustainability, it also requires time, resources, and collaboration with teacher preparation institutions. Immediate and short-term solutions include

- using existing alternative CTE certification pathways,
- allowing teachers to teach computer science under a temporary license while obtaining professional education in computer science or while pursuing full certification,
- requiring computer science in existing pathways for technology education,
- creating add-on endorsements for teachers who are already certified in other content areas, and
- developing or adopting a computer science teacher licensure exam for endorsement.

These types of solutions can be instituted in parallel with the development of a full certification pathway (Code.org, 2016b). In each case, the framework's concepts and practices should be used to inform the selection or development of the coursework or examination necessary for certification, and the resources and requirements for certification should be publicly posted and readily accessible.

Inservice Professional Development

Professional learning in computer science builds off experiences in preservice programs to provide a coherent teacher development experience based on a foundation of the framework's concepts and practices. Professional development is currently being used as a way of preparing existing teachers to meet the demand for computer science courses. School districts should consider collaborating closely with universities, informal education programs, and organizations to offer large-scale professional development and ensure a consistent teacher development experience.

A lack of current computer science teachers has driven school systems to meet the immediate need by building capacity amongst existing teachers.

Efforts to build teacher capacity in computer science face a special challenge because the teachers attending professional development opportunities represent a wide range of experience. In 2013, a landscape study of computer science professional development, *Building an Operating System for Computer Science*, found that more than half of all teachers attending professional development in computer science education are novices to computer science, rather than current computer science teachers (Century et al., 2013). In addition, three-fourths of the computer science professional development providers surveyed reported that they work with teacher participants who are new to computer science. These findings are expected, as a lack of current computer science teachers has driven school systems to meet the immediate need for computer science courses by building capacity amongst existing teachers certified in other areas, such as math, science, and general education technology (Century et al., 2013).

The wide variety of computer science experiences in teachers' backgrounds necessitates professional development experiences that are differentiated to meet the needs of multiple populations: teachers who are already experienced and certified in computer science but are in need of continuing education, teachers from other disciplines who are new to teaching computer science, and teachers who are preparing to integrate computer science content into other content areas. Differentiation for teachers' comfort levels with computer science can affect whether that teacher continues teaching computer science. In one study of workshops designed for teachers with prior computer science experience, the teachers who did not have computer science backgrounds experienced frustration and ultimately quit teaching computer science (Ericson, Guzdial, & Biggers, 2007). Furthermore, the needs of secondary school educators who teach independent computer science courses may differ from the needs of elementary school teachers who want to incorporate computer science into their teaching.

Although there are effective practices that apply to all professional development experiences, the following recommendations address issues particular to computer science.

Customize professional development to meet teachers' varied backgrounds in computer science. Teachers' experience with computer science varies, as does their primary area of certification. When possible, the audience for a workshop should be homogenous based on computer science experience and area of primary certification. When this is not possible, a workshop could include sessions for teachers to break into groups based on experience or certification.

Professional development should attend to novice teachers’ anxiety over their lack of content knowledge.

Given the introduction of computer science into many education systems, it is natural that many teachers attending professional development may not already have a background in computer science. While not diminishing the importance of pedagogical content knowledge or general pedagogical practice for teaching computer science, professional development providers should attend to teachers’ anxiety about content knowledge by helping them see that many teachers are in the same situation. Professional development can instill a growth mindset in participants, in which learning builds over time, during a workshop as well as the school year while teachers deliver instruction. Professional development should be viewed as a safe space to try new or difficult things.

Providers should connect professional development experiences to a curricular context.

Disciplinary and pedagogical content should be learned within the context of a teacher’s instructional goals, curricular frameworks, and/or courses. Professional development that focuses mainly on a programming language or how to use a tool, without providing time for participants to make plans for using those tools or languages in their courses, is less practical and actionable as it does not prepare teachers to deliver a meaningful curriculum. Alternatively, professional development connected to the concepts and practices in the framework can provide opportunities to practice teaching content (i.e., microteaching) and can be contextualized to particular curricula that teachers will be using in their classrooms.

Professional development should include a focus on increasing access and equity.

Computer science courses often lack diversity and can be intimidating for many students. Teachers should have experience engaging in and reflecting on the same practices in the framework that are expected of students, particularly in terms of access and equity, such as incorporating diverse perspectives into a design, meeting the needs of diverse end users, and creating equitable workloads for teams. Computer science brings unique issues that require the emphasis of particular pedagogical practices, such as equitable practices that address the varied exposure students have in computer science and stereotypes that exist about the field (Ryoo, Goode, & Margolis, 2016). Professional development opportunities focused on equitable teaching strategies have shown success in recruiting and retaining females and underrepresented minorities (Cohoon, Cohoon, & Soffa, 2011). The issue of equity in computer science is addressed more fully in the **Equity in Computer Science Education** chapter.

Professional development should include a focus on increasing access and equity.

Professional development should address the management of a productive computer lab environment.

The computer plays a much larger role in the computer science classroom than others. Students will often move between computer work and classroom instruction, sometimes within the same period, and at other times work on the computer for days.

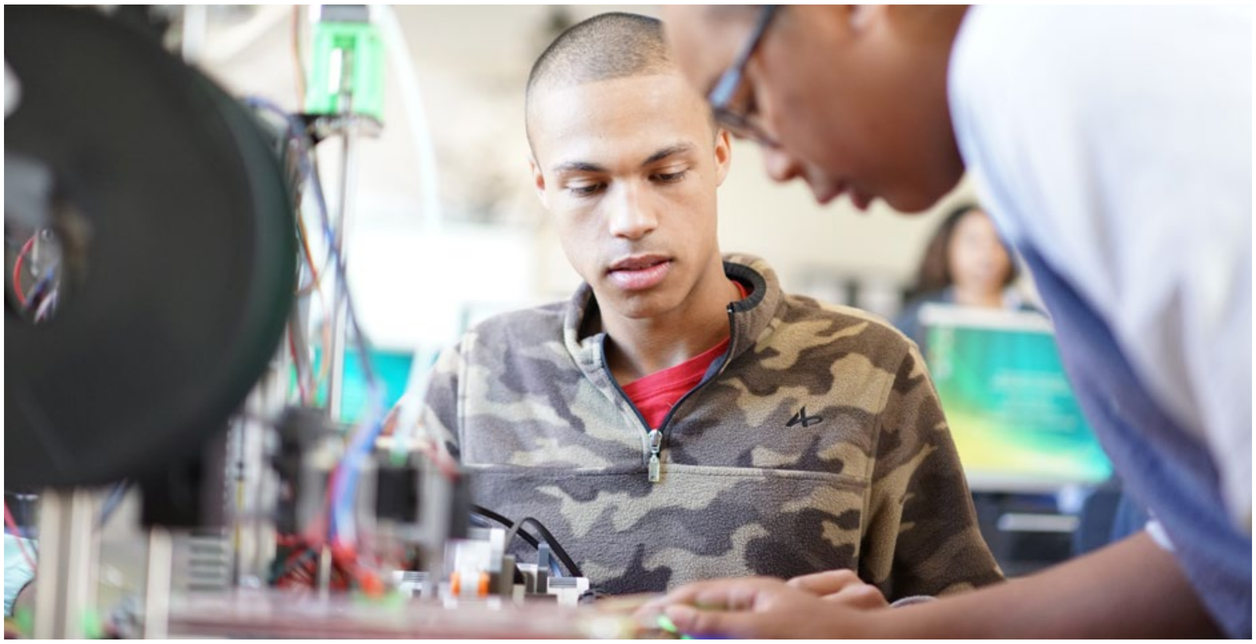
Sometimes an online environment can deliver instruction through videos and tutorials. Teachers must learn how to manage a classroom in which the computer serves as both the primary medium for demonstrating performance as well as an occasional teaching aid.

A number of these recommendations are inspired by the suggestions in *Building an Operating System for Computer Science* (Century et al., 2013), which offers additional guidance related to computer science teachers' needs, including a portrait of the computer science teaching population and the contexts in which they teach. The study is accessible at http://outlier.uchicago.edu/computerscience/OS4CS/.

Summary

The implementation of a K–12 computer science pathway is a long-term process, and the development of a robust framework of concepts and practices represents only a single step. Implementing computer science education requires engaging curriculum, evolving course pathways, technical infrastructure, and the involvement of the community and informal education organizations. Teachers from a variety of backgrounds must be prepared to teach the courses, and the overall pipeline of computer science teachers has to be built and filled for any reform to be sustainable.

The writers, advisors, and organizations who have developed this framework recognize that efforts to implement K–12 computer science exist in an education environment with multiple priorities including student engagement, high school graduation rates, high-stakes testing, teacher accountability, and budget shortfalls. Similar to a principle that helped guide the debates and discussions during the development of the framework, policymakers and educators must constantly make decisions based on what is best for students.



References

Afterschool Alliance. (2015). *America after 3 PM: Full STEM ahead.* Washington, DC. Retrieved from http://afterschoolalliance.org/AA3PM/STEM.pdf

Algorithmic Geometry. (2016). Retrieved from http://www.algogeom.org/

Barth, P., Dillon, N., Hull, J., & Higgins, B. H. (2016). *Fixing the holes in the teacher pipeline: An overview of teacher shortages.* Center for Public Education. Retrieved from http://www.centerforpubliceducation.org/Main-Menu/Staffingstudents/An-Overview-of-Teacher-Shortages-At-a-Glance/Overview-of-Teacher-Shortages-Full-Report-PDF.pdf

Black Girls Code. (2016). Retrieved from http://www.blackgirlscode.com

Black, P., Harrison, C., Lee, C., Marshall, B., & Wiliam, D. (2003). *Assessment for learning: Putting it into practice.* Maidenhead, Berkshire, UK: Open University Press.

Bootstrap. (2016). *From intro CS & Algebra to high-level courses in computer science, for all students.* Retrieved from http://www.bootstrapworld.org

Broward County Public Schools. (2016). *Broward codes.* Retrieved from http://browardschools.com/browardcodes

Century, J. (2009, September 25). The vanishing innovation: Why "sustaining change" must be as important as "scaling up." *Education Week.* Retrieved from http://www.edweek.org/ew/articles/2009/09/30/05century.h29.html

Century, J., Lach, M., King, H., Rand, S., Heppner, C., Franke, B., & Westrick, J. (2013). *Building an operating system for computer science.* Chicago, IL: CEMSE, University of Chicago with UEI, University of Chicago. Retrieved from http://outlier.uchicago.edu/computerscience/OS4CS/

Change the Equation (2016, August 9). New data: Bridging the computer science access gap [Blog post]. Retrieved from http://changetheequation.org/blog/new-data-bridging-computer-science-access-gap-0

Chicago Public Schools. (2016, February 24). New CPS computer science graduation requirement to prepare students for jobs of the future [Press release]. Retrieved from http://cps.edu/News/Press_releases/Pages/PR2_02_24_2016.aspx

Cliburn, D. C., & Miller, S. (2008). Games, stories, or something more traditional: The types of assignments college students prefer. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education* (pp. 138–142), Portland, OR.

Code.org. (2015). *Making computer science fundamental to K–12 education: Eight policy ideas.* Retrieved from https://code.org/files/Making_CS_Fundamental.pdf

Code.org. (2016a). *Promote computer science.* Retrieved from https://code.org/promote

Code.org. (2016b). *Teacher certification recommendations.* Unpublished paper.

Cohoon, J., Cohoon, J. M., & Soffa, M. (2011). Focusing high school teachers on attracting diverse students to computer science and engineering. In *Proceedings of the 41st ASEE/IEEE Frontiers in Education Conference* (pp. F2H-1–F2H-5). doi: 10.1109/FIE.2011.6143054

Computer Science Teachers Association Certification Committee. (2013). *Bugs in the system: Computer science teacher certification in the U.S.* Retrieved from the Computer Science Teachers Association and the Association for Computing Machinery website: https://csta.acm.org/ComputerScienceTeacherCertification/sub/CSTA_BugsInTheSystem.pdf

CTE Maryland. (2016). *Maryland public schools CTE enrollment.* Retrieved from https://www.mdctedata.org/dashboards/summary.php?c=IT&y=2016&l=25

DeLyser, L. A. (2016). *Building a computer science teacher pipeline for New York City.* Retrieved from the NYC Foundation for Computer Science Education website: http://pipeline.csnyc.org/#

Education Superhighway. (2015). *2015 state of the states: A report on the state of broadband connectivity in America's public schools.* Retrieved from http://stateofthestates.educationsuperhighway.org/assets/sos/full_report-55ba0a64dcae0611b15ba9960429d323e2eadbac5a67a0b369bedbb8cf15ddbb.pdf

Eglash, R. (2003). *Culturally situated design tools.* Retrieved from http://csdt.rpi.edu/

Eglash, R., Bennett, A., O'Donnell, C., Jennings, S., & Cintorino, M. (2006). Culturally situated design tools: Ethnocomputing from field site to classroom. *American Anthropologist (108)*2, 347–362.

Ericson, B., Guzdial, M., & Biggers, M. (2007). Improving secondary CS education: Progress and problems. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education* (pp. 298–301).

Every Student Succeeds Act of 2015, Pub. L. No. 114-95. 20 U.S.C.A. 6301 (2016).

Girls Who Code. (2016). *About us.* Retrieved from https://girlswhocode.com/about-us/

Goldweber, M., Barr, J., Clear, T., Davoli, R., Mann, S., Patitsas, E., & Portnoff, S. (2013). A framework for enhancing the social good in computing education: A values approach. *ACM Inroads, 4*(1), 58–79.Go

Governors for Computer Science. (2016). Retrieved from http://www.governorsforcs.org/about

Heitin, L. (2016, August). Physics not offered at 2 in 5 high schools, analysis finds. *Education Week*, *36*(1), 6.

Higher Education Act of 2008, title II U.S.C. § 205 *et seq.*

Illinois State University. (2016). *Teacher education in computer science (TECS).* Retrieved from http://tecs.illinoisstate.edu/resources/undergraduates

Indiana Department of Education. (2016). *Science & computer science.* Retrieved from http://www.doe.in.gov/standards/science-computer-science

LeadCS.org. (2015). *Advice from school leaders: Preparing for computer science in your school.* Chicago, IL: CEMSE, Outlier Research & Evaluation, University of Chicago. Retrieved from http://www.leadCS.org

Level Playing Field Institute. (2016). Retrieved from http://www.lpfi.org

Lopez, J. K. (n.d.). Funds of knowledge. In *Bridging Spanish language barriers in Southern schools.* Retrieved from http://www.learnnc.org/lp/editions/brdglangbarriers/939

Margolis, J., Ryoo, J., Sandoval, C., Lee, C., Goode, J., & Chapman, G. (2012). Beyond access: Broadening participation in high school computer science. *ACM Inroads*, *3*(4), 72–78.

Margolis, J., Goode, J., & Chapman, G. (2015). An equity lens for scaling: A critical juncture for Exploring Computer Science. *ACM Inroads, 6*(3), 58–66.

Maryland State Department of Education. (2005). *Maryland technology education state curriculum.* Retrieved from http://mdk12.msde.maryland.gov/instruction/curriculum/technology_education/vsc_technologyeducation_standards.pdf

Massachusetts Department of Elementary and Secondary Education. (2016, June). *2016 Massachusetts digital literacy and computer science (DLCS) curriculum framework*. Malden, MA: Author. Retrieved from http://www.doe.mass.edu/frameworks/dlcs.pdf

Moll, L., Amanti, C., Neff, D., & Gonzalez, N. (1992). Funds of knowledge for teaching: Using a qualitative approach to connect homes and classrooms. *Theory into Practice 31*(2), 132–141.

National Center for Women & Information Technology. (2016). Retrieved from https://www.ncwit.org

National Girls Collaborative Project. (2016). Retrieved from https://ngcproject.org

Nord, C., Roey, S., Perkins, R., Lyons, M., Lemanski, N., Brown, J., and Schuknecht, J. (2011). *The nation's report card: America's high school graduates (NCES 2011-462).* U.S. Department of Education, National Center for Education Statistics. Washington, DC: U.S. Government Printing Office.

Papert, S. (1980). *Mindstorms: Children, computers and powerful ideas.* New York, NY: Basic Books.

Project GUTS: Growing Up Thinking Scientifically. (2016). Retrieved from http://www.projectguts.org

Rader, C., Hakkarinen, D., Moskal, B. M., & Hellman, K. (2011) Exploring the appeal of socially relevant computing: Are students interested in socially relevant problems? In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (pp. 423–428). doi: 10.1145/1953163.1953288

Ryoo, J., Goode, J., & Margolis, J. (2016): It takes a village: Supporting inquiry- and equity-oriented computer science pedagogy through a professional learning community. *Computer Science Education*. doi: 10.1080/08993408.2015.1130952

Shulman, L. S. (1986). Those who understand: Knowledge growth in teaching. *Educational Researcher, 15*(2), 4–14. doi: 10.3102/0013189X015002004

Snodgrass, M. R., Israel, M., & Reese, G. (2016). Instructional supports for students with disabilities in K–5 computing: Findings from a cross-case analysis. *Computers & Education, 100,* 1–17.

STEM Education Act of 2015, Pub. L. No. 114-59. 129 Stat. 540 (2015).

Stephenson, C. (2015, July 16). The thorny issue of CS teacher certification [Blog post]. Retrieved from https://research.googleblog.com/2015/07/the-thorny-issue-of-cs-teacher.html

Title II. (2016). *2015 Title II Reports: National teacher preparation data* [Data file]. Retrieved from https://title2.ed.gov/Public/Home.aspx

Tucker, A., McCowan, D., Deek, F., Stephenson, C., Jones, J., & Verno, A. (2006). *A model curriculum for K–12 computer science: Report of the ACM K–12 task force curriculum committee* (2nd ed.). New York, NY: Association for Computing Machinery.

UC Davis C-STEM Center. (2016). *Transforming math education through computing.* Retrieved from http://c-stem.ucdavis.edu/

UTeach College of Natural Sciences. (2016). *Computer science*. Retrieved from https://austin.uteach.utexas.edu/certifications-and-degrees/certifications/high-school-certifications/computer-science

White House. (2016, September 13). *FACT SHEET: New progress and momentum in support of President Obama's Computer Science for All initiative*. Retrieved from https://www.whitehouse.gov/sites/default/files/microsites/ostp/csforall-fact-sheet-9-13-16-long.pdf

Wiggins, G., & McTighe, J. (2005). *Understanding by design* (Expanded 2nd ed.). Alexandria, VA: Association for Supervision and Curriculum Development.

Wilson, C., & Yongpradit, P. (2015, June 9). Maryland moves to increase diversity in computer science [Blog post]. Retrieved from http://blog.code.org/post/121123281798/md

Wilson, C., Sudol, L. A., Stephenson, C., & Stehlik, M. (2010). *Running on empty: The failure to teach K–12 computer science in the digital age.* Retrieved from the Association for Computing Machinery and the Computer Science Teachers Association website: http://runningonempty.acm.org/

WGBH Educational Foundation & the Association for Computing Machinery. (2009). *New image for computing: Report on market research (April 2009).* Retrieved from http://www.acm.org/membership/NIC.pdf

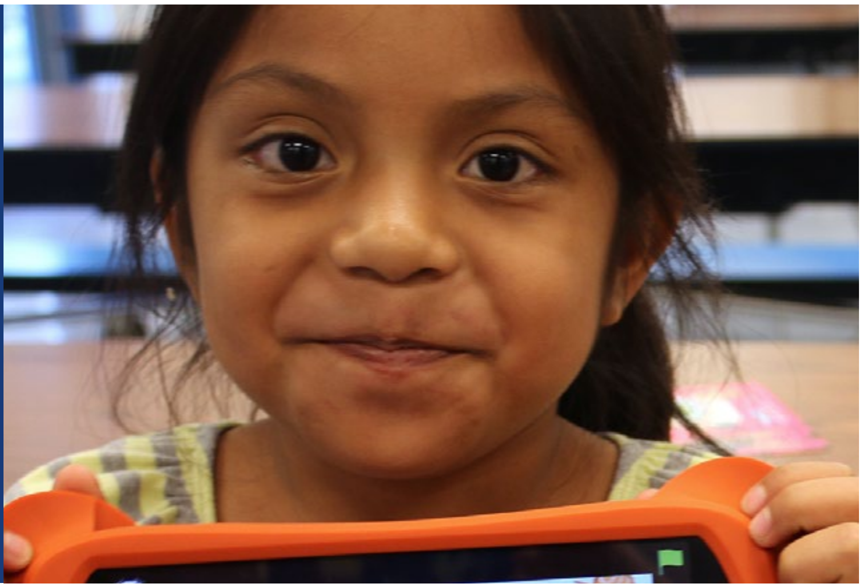
Yadav, A., Burkhart, D., Moix, D., Snow, E., Bandaru, P., & Clayborn, L. (2015). *Sowing the seeds: A landscape study on assessment in secondary computer science education.* Retrieved from the Computer Science Teachers Association website: https://csta.acm.org/Research/sub/Projects/ResearchFiles/AssessmentStudy2015.pdf

Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., & Korb, J. T. (2014). Computational thinking in elementary and secondary teacher education. *ACM Transactions on Computing Education (TOCE), 14*(1), Article 5, 1–16.

Yadav, A., Zhou, N., Mayfield, C., Hambrusch, S., and Korb, J. (2011) Introducing computational thinking in education courses. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (pp. 465–470), Dallas, TX.

Computer Science in Early Childhood Education

9



Computer Science in Early Childhood Education

Amidst concerns over an unprepared 21st century workforce, U.S. policymakers have placed increased emphasis on the subjects of science, technology, engineering, and math (STEM) to ensure that youth are sufficiently equipped to compete in the increasingly global economy (e.g., National Science Board, 2012; U.S. President's Council of Advisors on Science and Technology, 2010; U.S. Congress Joint Economic Committee, 2012). Computer science in particular has recently taken the national spotlight, with large-scale initiatives at the local, state, and federal levels aiming to ensure that students gain computational literacy skills viewed as "a 'new basic' skill necessary for economic opportunity and social mobility" (Smith, 2016, para. 1).

Many of these computer science initiatives focus on the K–12 and postsecondary education environments. For example, Maryland has established prekindergarten (pre-K) computer science standards (Maryland State Department of Education, 2015), while several school districts, including San Francisco Unified School District (Twarek, 2015) and Boston Public Schools (2016), have taken it upon themselves to initiate computer science education at the pre-K level.

In the same spirit of providing computer science learning opportunities for all K–12 students as a means to ensure a prepared and productive workforce for the 21st century, investing in early childhood education has been shown to be one of the best means for closing early achievement and development gaps, which subsequently aids the economic and social well-being of the broader community (Heckman, 2006; Heckman & Masterov, 2007; Magnuson, Meyers, Ruhm, & Waldfogel, 2004).

From landmark studies on the High/Scope Perry Preschool Program (Schweinhart et al., 2011) and the Carolina Abecedarian Project (Campbell, Ramey, Pungello, Sparling, & Miller-Johnson, 2002) to a recent review of 84 early childhood education interventions (Camilli, Vargas, Ryan, & Barnett, 2010), research consistently shows that high-quality early learning experiences have positive short- and long-term effects on children's learning and development. While specific "quality learning experiences" may look different from classroom to classroom, several common elements include instructionally and emotionally supportive interactions between teachers and students; developmentally appropriate curricular resources and materials; structured learning activities individualized to match students' different needs; and opportunities for exploration and play (e.g., Pianta, Barnett, Burchinal, & Thornberg, 2011; Yoshikawa et al., 2013).

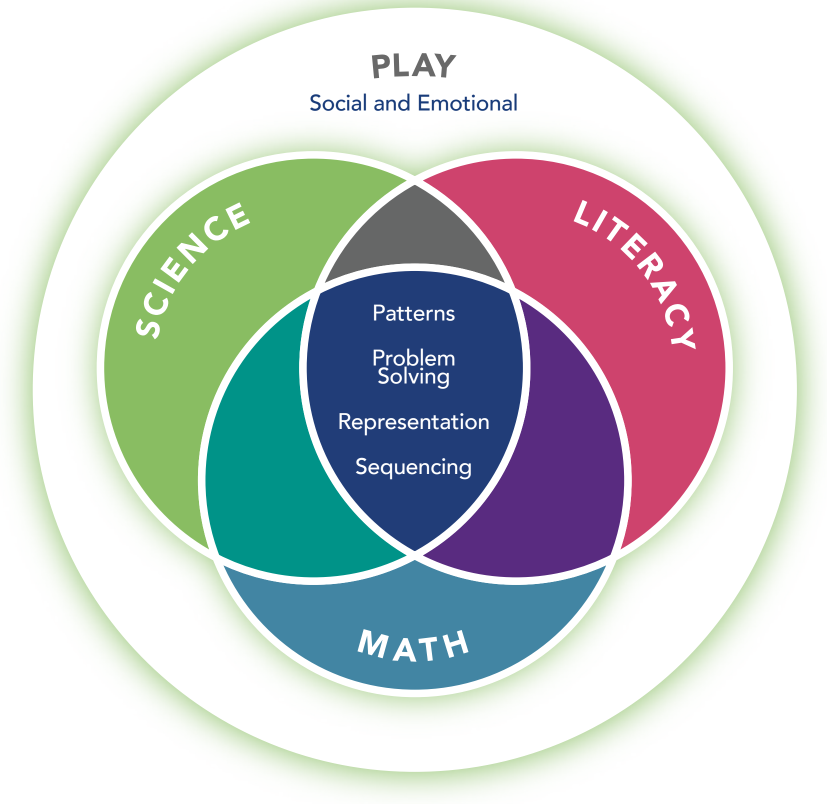
Given the large impacts that early childhood education can have on young children's learning and development, and the subsequent impacts this foundational learning has on the broader economic and social welfare, what, then, is the place of computer science in early childhood education? In response to this lack of clarity regarding what computer science looks like in the early childhood classroom, this chapter outlines practical applications of the K–12 Computer Science Framework's concepts and practices appropriate for the pre-K setting.

Powerful Ideas in Pre-K Computer Science

Researchers, educators, and policymakers alike have suggested a myriad of ideas as to the principles of computer science education. The proposed K–12 Computer Science Framework is itself an amalgamation of such principles and draws on Papert's (1980) "powerful ideas" to articulate specific computer science concepts and practices for the K–12 learning environment. However, instead of simply applying these core concepts and practices to the pre-K environment—and thus assuming that they are developmentally appropriate and necessary for all young children to learn—this chapter outlines a set of "powerful ideas" specific to early childhood education. These pre-K computer science concepts and practices build foundational knowledge and understanding for later engagement in computer science at the elementary school level. They are grounded in the research literature on computer science education in early childhood education settings and based on Papert's (1980) constructionist framework, which emphasizes children's active engagement in knowledge building through the construction of physical objects, where computing technologies are tools with which children can build and design to develop such knowledge. Teachers scaffold these active learning experiences by providing structured support that helps guide students to deeper engagement and higher-level thinking. Constructionism provides the foundation for much of the computer science education research and practice and aligns with traditional conceptions of early childhood education as a hands-on, interactive, and play-based learning environment (Bers, Ponte, Juelich, Viera, & Schenker, 2002).

As outlined in Figure 9.1, four powerful ideas are embedded within the core content areas of math, literacy, and science, and the fifth—social and emotional learning—is understood as a holistic frame for all early childhood educational practices. Further, these powerful ideas are encompassed by the pedagogical bedrock of early learning environments: **play**.

Figure 9.1: Integrating powerful ideas in computer science and early childhood education



In the following sections, these five powerful ideas are described in current, everyday pre-K contexts and then extended to a computer science context. In this way, computer science becomes a natural extension of children's everyday engagement with their environment and builds on what educators already do in their daily practice. Further, each powerful idea is connected to one or more of the framework's practices to provide insight on the progression from pre-K to elementary computer science.

1. Social and Emotional Learning: Strong Affective, Behavioral, and Cognitive Competencies Provide the Foundation for Successful Learning and Development.

As defined by the Collaborative for Academic, Social, and Emotional Learning (CASEL, 2012), social and emotional learning (SEL) "involves the processes through which children and adults acquire and effectively apply the knowledge, attitudes, and skills necessary to understand and manage emotions, set and achieve positive goals, feel and show empathy for others, establish and maintain positive relationships, and make responsible decisions (p. 4)." The five core competencies of SEL are self-awareness, self-management, social awareness, relationship skills, and responsible decision-making.

Children develop social and emotional skills through playful interactions with peers and adults, and research continually shows these interactions can have significant impacts on children's learning and development (e.g., Mashburn et al., 2008; Pianta et al., 2002). More than 500 studies demonstrate the positive benefits of SEL for children's interpersonal relationships, cognition, and academic learning in all content areas (e.g., Klem & Connell, 2004; Weissberg & Cascarino, 2013; Weissberg, Durkak, Doitrovich, & Gullota, 2015), and a strong SEL foundation developed in early childhood can have lasting impacts on children's future academic and professional success (e.g., Camilli et al., 2010; Chetty et al., 2011). Importantly, when teachers emotionally engage with students, show they care, listen to students' needs and desires, and take time to be mindful of momentary tones of the classroom climate, they can both model and elicit SEL among children.

Framework connections: P1.Fostering an Inclusive Computing Culture, P2.Collaborating Around Computing, and P7.Communicating About Computing

Within the context of the framework, practices 1, 2, and 7 encompass being able to work and communicate with teams with lots of differing perspectives. Teachers can foster an inclusive computing environment by presenting opportunities for students to share, collaborate, and support one another. They can also encourage children to be self-aware of their own engagement. These skills can be proactively addressed through conversations about differences in behaviors, opinions, and perspectives; advocating for self and friends; and struggles throughout playful engagements that were solved (or could be solved) through mediation and empathic problem solving.

In computer science, the best products are created by teams consisting of members with varied backgrounds who listen to and respect one another's ideas. Additionally, computer science is more than just creating products and involves effectively communicating (verbally and visually) processes and solutions to a broader audience. These principles can be developed in the pre-K classroom by fostering children's social and emotional development through play.

In computer science, the best products are created by teams consisting of members with varied backgrounds who listen to and respect one another's ideas.

Everyday example

Learning to play with a new playmate is a regular occurrence in an early learning environment and often involves a process of negotiation requiring teacher scaffolding. Educators can set up structures to help facilitate this process in three phases. First, child A chooses what game to play for five minutes. Then, child B chooses what game to play for five minutes. In the third iteration, the students practice compromising to find a game they will both enjoy. Teachers can help facilitate this third phase by encouraging child A to state an attribute of a game she wants to play without naming the actual game (e.g., "building something" instead of "playing with blocks"). Then child B refines this suggestion with something he wants. If the children have more verbal and reasoning skills, they could each describe several attributes and then problem solve together to come up with a game that satisfies all of the attributes. This allows each child to take a turn being the leader and a turn being a respectful listener and follower, as well as gives them a blueprint for having a conversation that takes into account multiple opinions.

Computer science example

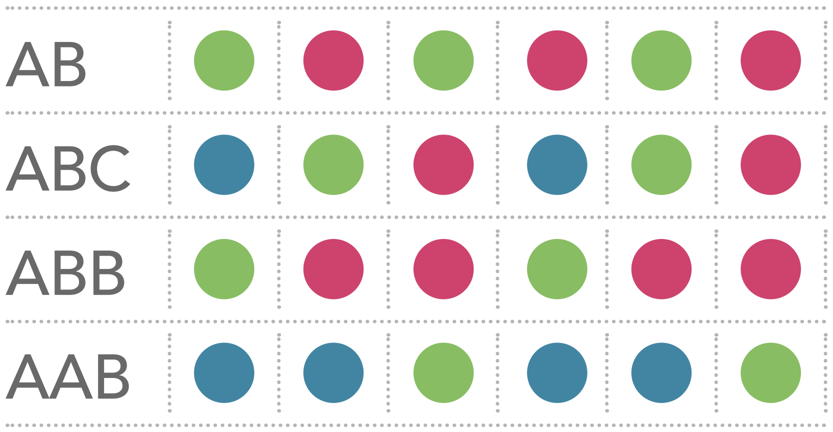
Often, early childhood education environments are not outfitted with one-to-one computing devices, such that children work in pairs or small groups by default. Educators can leverage this natural setup by facilitating pair programming experiences for children. At the simplest level, this facilitation could be helping students learn to share the device through the use of "My turn"/"Your turn" flashcards, with children passing the cards back and forth to designate whose turn it is to use the computer. Taking this a step further, educators can provide opportunities for computer-supported collaborative learning (Dillenbourg, 1999; Goodyear, Jones, & Thompson, 2014) in which children work together on the computer to solve a shared task or problem, rather than just taking separate turns using the device. Pair programming involves one person taking on the role of a "driver" while the other is the "navigator." The driver is the person who controls the actions of the computer and focuses on the details, while the navigator takes a bigger picture view of the problem and helps by answering questions and looking out for potential problems or mistakes. At the pre-K level, teachers can help facilitate pair programming among two children with the same "My turn"/"Your turn" flashcards to designate driver/navigator roles as well as encourage children to engage in collaboration and communication skills to foster peer-to-peer scaffolding. Educators can provide more support and scaffolding by engaging in child/teacher pair programming.

Pair programming involves one person taking on the role of a "driver" while the other is the "navigator."

2. Patterns: Patterns Help Us Make Sense of the World by Organizing Objects and Information Using Common Features (e.g., Color, Shape, Size).

In computer science, patterns allow people to reduce complexity by generalizing and applying solutions to multiple situations. Learning about patterns in the early years can build a foundation for developing and using abstractions (e.g., defining and calling procedures), solving computational problems more effectively (e.g., using loops instead of repeating commands), and making inferences (e.g., using models and simulations to draw conclusions). An example is shown in Figure 9.2.

Figure 9.2: Identifying patterns



Framework connections: P4.Developing and Using Abstractions

One aspect of developing and using abstractions is the ability to categorize items/objects/code and identify general attributes based on those categorizations (or "abstract" out more general patterns to describe the categorizations). Abstraction is one step beyond recognizing patterns, where the focus is on identifying and describing repeated features but not yet categorizing items/objects/code based on those features or abstracting more general attributes to define those categorizations.

Everyday example

Children learn to recognize patterns through routines that provide experiences with noticing and naming features of things in their worlds. For example, a child may notice and name the colors in a classroom or notice and name different features of Legos®. Over time, they can start to identify repeated features and, in turn, create images or move objects to show repeated features. For example, a child may place colored objects in an order—green, red, green, red, green, etc. Those patterns can become even more elaborate as additional colors or features are taken into consideration—yellow, green, red, yellow, green, red. Additionally, rhythm is a great example of a pattern because it presents a repeated sound and movement pattern. Children can show patterns in movement through dance, repeating a physical movement.

Extending this one step further into early abstraction skills, students can be given a specific category (e.g., dog or cat) and identify what they know (i.e., attributes) based on that categorization. For example, for the category "cat," some attributes children will know are the approximate size (significantly smaller than a horse), the color (not purple, green, or blue), the number of eyes (two) and legs (four), and the existence of a tail. Children also know that a cat likes to pounce and that it meows, but a cat cannot bark, nor can it fly.

Computer science example

In the digital world, computers use patterns to organize information. A teacher can show how repeated patterns are everywhere in the world. For example, barcode patterns that are everywhere in a grocery store are meant to give information about the item to the cash register. The barcode uses repeated features (thin lines, thick lines, and space) to give directions to computing devices. The "beep, beep, beep" of the scanner is also a pattern in the grocery store because it is a repeated feature. Patterns can be grouped. So, for example, we could group the pattern of barcode-beep, barcode-beep, barcode-beep.

Teachers can use a visual, block-based programming environment to present a pattern of commands and guide students in identifying the pattern. Many block-based programming languages use different colors for types of blocks. Teachers can demonstrate how a series of commands can be used to draw a simple shape, such as a square or triangle, and ask students to identify patterns by using different colors as visual cues. This activity can serve as a precursor to students independently creating their own programs.

3. Problem Solving: Children Construct Knowledge Through Problem Solving.

Young children naturally engage in problem-solving processes in their daily lives as they explore and interact with the world around them. Teachers can help make problem solving "visible" by asking questions to uncover children's reasoning and thought processes (e.g., How did you know that? What made you think that?) as well as offering structured methods to scaffold children's problem solving. One such method often used in computer science is an iterative development process. This process involves identifying a problem; devising and testing solutions; evaluating the results; and revising and redoing to find the best solution. Central to this process is making mistakes and learning from them to effectively solve new problems in different situations.

Framework connections: P3.Recognizing and Defining Computational Problems, P5.Creating Computational Artifacts, and P6.Testing and Refining Computational Artifacts

In one sense, computer science is the study of problems, problem-solving processes, and the solutions that result from such processes. Engaging in problem-solving activities early on can set the foundation for recognizing and defining computational problems, engaging in testing and refinement strategies, and developing and evaluating computational solutions to real-world problems.

Everyday example

Children regularly problem solve when they build with blocks. For example, while making a block bridge for toy cars to cross, a child may move two base blocks near each other and add a block on top. If the top block does not reach across the base blocks, then the child might move the base blocks closer together and try to balance the top block again. A teacher can make this problem-solving behavior visible for children by explaining how the child's action demonstrates revising and redoing. The teacher can talk about the student's thought process that went into this kind of problem solving, making comments such as, "Wow, I noticed that you figured out that the bridge wouldn't work. How did you know what to do next to solve your problem?"

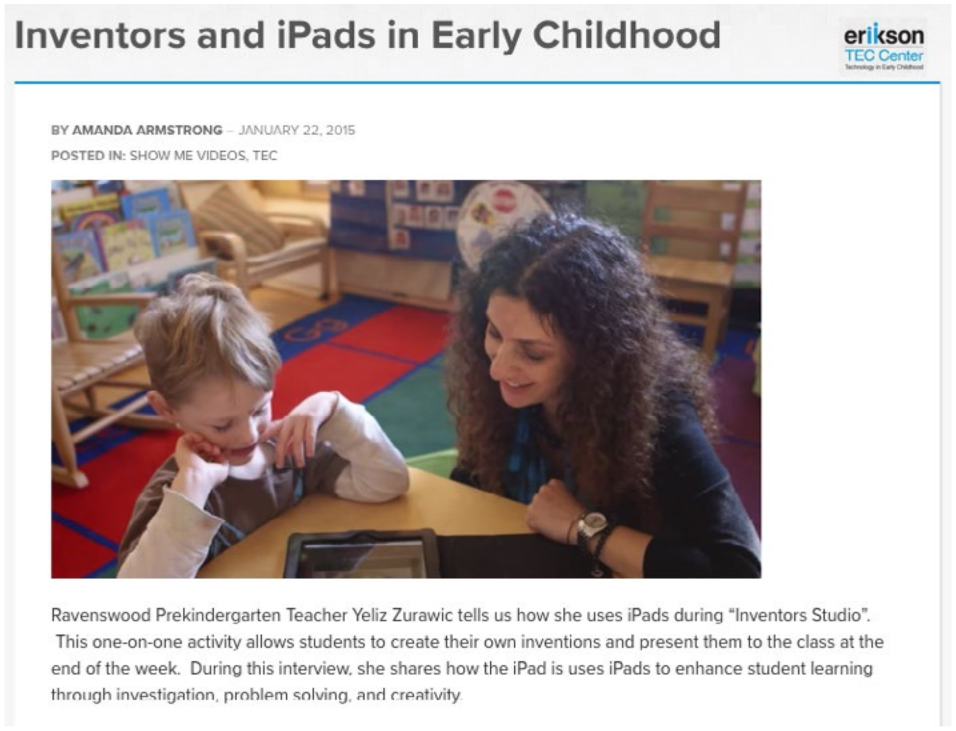
Additionally, teachers can engage children in an iterative development process by setting up an "Inventors Studio" in their classroom where children use problem-solving skills to create something new with the help of teacher scaffolding and technology resources technology resources (see Figure 9.3). For example, if a child wants to make a sock puppet, she might first draw images of her sock puppet on paper and write down all the materials she would need to make it. Then, with the help of a teacher, she could look at images of sock puppets online and search for instructions on how to make one. The child can then compare and revise her original drawing and materials list based on what she and her teacher find online before trying to build the actual sock puppet. Throughout the construction process, the teacher can also use technology to help document this problem-solving design process by taking pictures along the way so that the child can look back and reflect on the different steps that went into creating her sock puppet.

A teacher can also explicitly present problems and ask for children's creative solutions. For example, she may present a scenario: A mouse wants to hop onto a bed, but the nearby shoebox is too short to help the mouse reach its destination. How might the mouse solve this problem? What else could a mouse use to reach the top of the bed? In this example, the teacher can solicit many different ideas and emphasize that there is no one right answer to solving this problem but that some solutions may be more effective and efficient than others.

Computer science example

When developers create new technology, they often use an iterative design protocol that involves creating an early version of the technology, testing it out, evaluating the results, making revisions, and then testing it out again. In a classroom, sometimes technology does not work, and teachers can engage children in a similar iterative problem-solving process to figure out why. They might check to see whether the device is turned on, whether it is out of power, or whether it is physically broken. These different checks are important to figuring out why the technology is not working and how to get it to work again.

Figure 9.3: Student using technology resources during "Inventors Studio"



Inventors and iPads in Early Childhood

BY AMANDA ARMSTRONG – JANUARY 22, 2015
POSTED IN: SHOW ME VIDEOS, TEC

Ravenswood Prekindergarten Teacher Yeliz Zurawic tells us how she uses iPads during "Inventors Studio". This one-on-one activity allows students to create their own inventions and present them to the class at the end of the week. During this interview, she shares how the iPad is uses iPads to enhance student learning through investigation, problem solving, and creativity.

4. Representation: People Can Represent Concepts Using Symbols

Any language that has a print version is an example of how language can be represented. In the case of English, the language is represented by words or word parts, which denote sounds and meanings. Similarly, computational languages are represented by numbers, text, and symbols.

Framework connections: P4.Developing and Using Abstractions, P5.Creating Computational Artifacts, and P7.Communicating About Computing

Understanding representation in the early years can build a foundation for understanding how computers represent information and simulate the behavior of systems, both of which are important for developing and using abstractions. Additionally, the creation of computational artifacts involves developing simulations and visualizations that require an understanding of how computers represent data, and effective communication about computing involves presenting information through visual representations (e.g., storyboards, graphs).

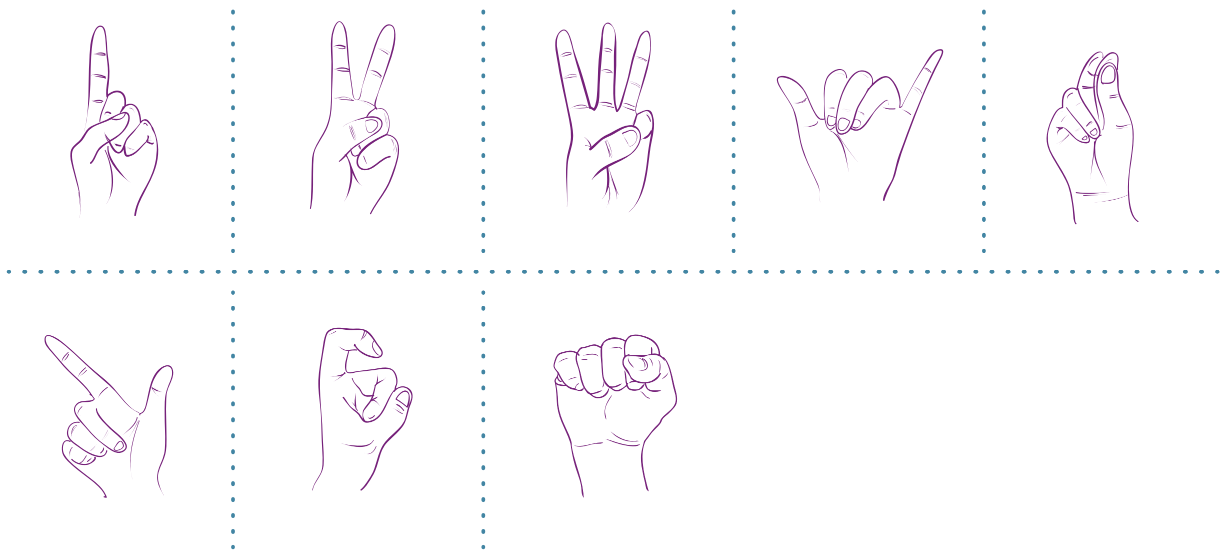
Everyday example

Children draw pictures of their family that often look like "potato people"—circles for the head and body and lines sticking out as arms and legs. Sometimes these "potato people" representations have deeper meanings and stories behind them. Teachers can extend the idea of representation by asking a child to share what her picture means and writing—in printed text—a sentence to describe the child's pictorial representation.

Alternatively, teachers can introduce the idea of representation by presenting a picture or symbol that represents an idea. For example, a teacher may represent directions with icons such as an arrow for moving forward, a spiral for turning around, or an octagon for stop. These iconic representations can be used on cards for playing games such as "Red Light, Green Light," in which children look at the pictorial representations of the directions and act accordingly.

In a similar vein, teachers can explore representation with children by comparing different cultural number systems. In America, children can count from one to five on a single hand, where each finger represents a value of one. In Chinese, however, they can count from one to ten on one hand. Teachers can teach children the Chinese system and then discuss the differences between the two. The American system relies only on the number of fingers that are being held up, but the Chinese system also uses which fingers are used, the way in which the finger is held (curled), and the angle (down versus up). Figure 9.4 shows examples of representations of numbers.

Figure 9.4: Example of representing numbers using fingers

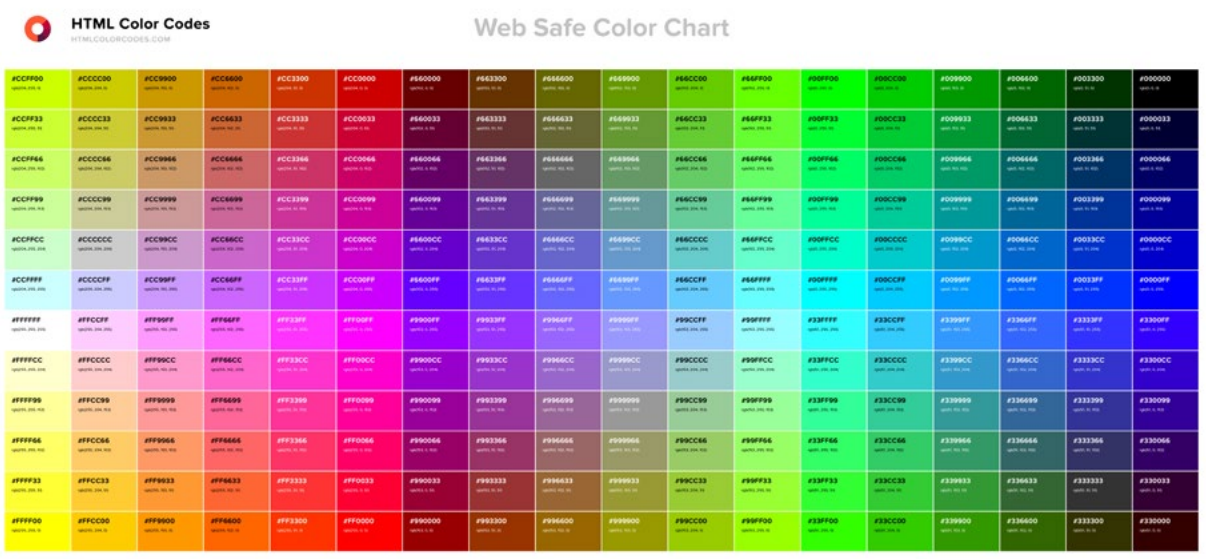


Computer science example

In the digital world, computers scientists use representations to "communicate" with computers. A child may see a representation demonstrated as app icons on a smartphone, where each icon represents a different app. Other "buttons" are visible in the computing world. For example, on/off switches and digital dashboards in cars are all examples of representations of information in the computing world. Teachers can engage children in exploring the different types of representations on the computing devices in their classroom.

Additionally, computers use representations to function more efficiently. For example, computers represent colors with numeric values. Using a simple word processing, drawing, or photo editing application on a computing device, children can play with the RGB (red, green, blue) number values of a color. Teachers can scaffold learning by pointing out how computers represent colors with numbers and how by changing the number values the color that appears on the screen changes (see Figure 9.5).

Figure 9.5: Numeric values that represent colors


HTML Color Codes
HTMLCOLORCODES.COM
Web Safe Color Chart

#CCFF00	#CCCC00	#CC9900	#CC6600	#CC3300	#CC0000	#660000	#663300	#666600	#669900	#66CC00	#66FF00	#00FF00	#00CC00	#009900	#006600	#003300	#000000
#CCFF33	#CCCC33	#CC9933	#CC6633	#CC3333	#CC0033	#660033	#663333	#666633	#669933	#66CC33	#66FF33	#00FF33	#00CC33	#009933	#006633	#003333	#000033
#CCFF66	#CCCC66	#CC9966	#CC6666	#CC3366	#CC0066	#660066	#663366	#666666	#669966	#66CC66	#66FF66	#00FF66	#00CC66	#009966	#006666	#003366	#000066
#CCFF99	#CCCC99	#CC9999	#CC6699	#CC3399	#CC0099	#660099	#663399	#666699	#669999	#66CC99	#66FF99	#00FF99	#00CC99	#009999	#006699	#003399	#000099
#CCFFCC	#CCCCCC	#CC99CC	#CC66CC	#CC33CC	#CC00CC	#6600CC	#6633CC	#6666CC	#6699CC	#66CCCC	#66FFCC	#00FFCC	#00CCCC	#0099CC	#0066CC	#0033CC	#0000CC
#CCFFFF	#CCCCFF	#CC99FF	#CC66FF	#CC33FF	#CC00FF	#6600FF	#6633FF	#6666FF	#6699FF	#66CCFF	#66FFFF	#00FFFF	#00CCFF	#0099FF	#0066FF	#0033FF	#0000FF
#FFFFFF	#FFCCFF	#FF99FF	#FF66FF	#FF33FF	#FF00FF	#9900FF	#9933FF	#9966FF	#9999FF	#99CCFF	#99FFFF	#33FFFF	#33CCFF	#3399FF	#3366FF	#3333FF	#3300FF
#FFFFCC	#FFCCCC	#FF99CC	#FF66CC	#FF33CC	#FF00CC	#9900CC	#9933CC	#9966CC	#9999CC	#99CCCC	#99FFCC	#33FFCC	#33CCCC	#3399CC	#3366CC	#3333CC	#3300CC
#FFFF99	#FFCC99	#FF9999	#FF6699	#FF3399	#FF0099	#990099	#993399	#996699	#999999	#99CC99	#99FF99	#33FF99	#33CC99	#339999	#336699	#333399	#330099
#FFFF66	#FFCC66	#FF9966	#FF6666	#FF3366	#FF0066	#990066	#993366	#996666	#999966	#99CC66	#99FF66	#33FF66	#33CC66	#339966	#336666	#333366	#330066
#FFFF33	#FFCC33	#FF9933	#FF6633	#FF3333	#FF0033	#990033	#993333	#996633	#999933	#99CC33	#99FF33	#33FF33	#33CC33	#339933	#336633	#333333	#330033
#FFFF00	#FFCC00	#FF9900	#FF6600	#FF3300	#FF0000	#990000	#993300	#996600	#999900	#99CC00	#99FF00	#33FF00	#33CC00	#339900	#336600	#333300	#330000

5. Sequencing: Sequencing Is the Process of Arranging Events, Ideas, and Objects in a Specific Order

Children often learn about sequence through early literacy and math. For example, children learn that stories follow a sequence (beginning, middle, end). Similarly, sequencing is explored through ordinal numbers (first, second, third) as well as size and magnitude (smallest to largest). In computer science, sequencing is an important foundation for algorithms, which are precise sets of instructions that computers follow to accomplish a specific task. It is critical that people give instructions in the proper sequence because computers do exactly what they are programmed to do; if the instructions are not sequenced properly, the algorithm will not achieve the desired result.

Framework connections: P3.Recognizing and Defining Computational Problems, P4.Developing and Using Abstractions, and P5.Creating Computational Artifacts

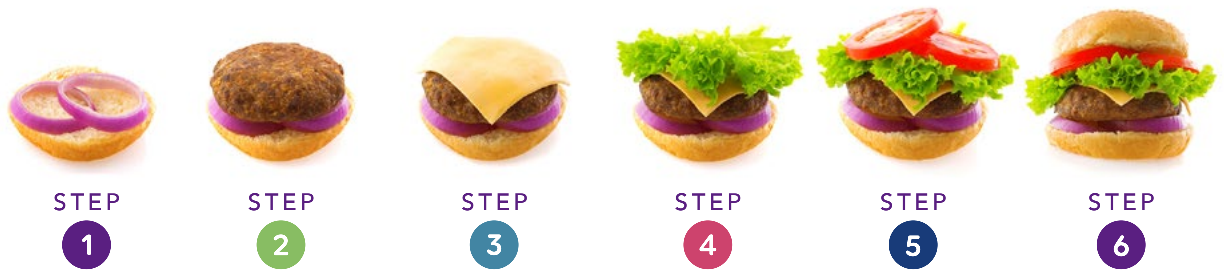
Learning about sequencing in the early years can build a foundation for learning one of the five core concepts of the framework, *Algorithms and Programming*—key ideas in computational problem solving, abstraction, and artifact creation. For example, understanding that instructions follow a specific sequence sets the foundation for children being able to break down (or decompose) complex problems into smaller steps that, if followed sequentially, will solve the problem.

Everyday example

Children love to tell stories; they talk about what happened over the weekend, about family events, and about different happenings in the classroom. Each of these stories can be broken down into a sequence of activities. Teachers can ask targeted questions to help children extend these ideas. For example, a teacher might ask, “What happened last?” “What happened first?” and “What happened in the middle?”

Another way to extend the idea of sequencing is for teachers to ask children to give instructions for an everyday task, like getting dressed. Teachers can ask children to sequence the activities into what they do first, second, and third as they dress themselves for the day. They can also have children draw the outcome of specific sequences to show how the order of events in a sequence can make a big difference (e.g., putting socks on after shoes, taking a shower after getting dressed). Alternatively, teachers can have students engage in a story sequencing activity in which children have to put a series of pictures in the correct order so that the story makes sense. Figure 9.6 shows a sequence for making a cheeseburger.

Figure 9.6: Sequence of steps to make a cheeseburger



Computer science example

A sequence of tasks can be explained in the context of using a digital tool. For example, a teacher may explain that food items are scanned at the grocery store and that is how the price is indicated. The sequence is (1) a food item is scanned and (2) the price is indicated by the cash register. This input/output of digital information from scanning the item to showing the price in the register can be acted out by children through pretend play.

Block-based programming environments employ touch-based interfaces and reduced commands to make programming accessible.

Students who are developmentally ready can use simple block-based programming environments to create simple algorithms and programs composed of sequences of commands. These environments allow students to create programs without the obstacle of typing found in traditional text-based languages. They often employ touch-based interfaces and reduced command sets to make programming accessible to young learners. The visual blocks are representations of commands a computer follows to run programs such as animations (Strawhacker & Bers, 2014). There are also robotics environments created for pre-K students that use tangible wooden blocks to create sets of commands that can be read by the robot to move, make sounds, and flash lights (Elkin, Sullivan, & Bers, 2014).

From Pre-K Powerful Ideas to the K–12 Framework

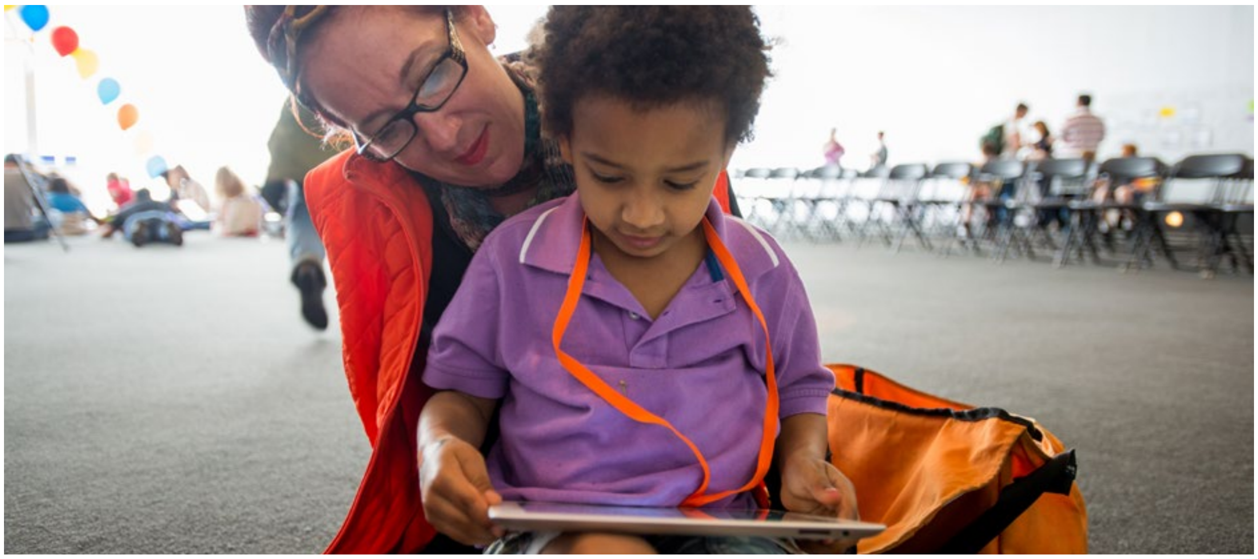
The importance of high-quality early childhood education cannot be overstated, and just as children engage in early literacy, math, and science activities, so too can they engage in foundational computer science learning. In spite of the recent push to make early learning environments more academic (NRC, 2009), computer science is a tool for developing more than technical skills and content knowledge, and it can be embedded into developmentally appropriate, play-based early learning practices (Copple & Bredekamp, 2009; Bers et al., 2002). As Resnick (2003) explained, computer science is well-suited for early childhood education as it offers a learning environment where young children can "play to learn while learning to play" (Bers, Flannery, Kazakoff, & Sullivan, 2014, p. 146). Further, the early learning environment is often characterized by open-ended, project-based activities that traverse content areas, and learning is often an ongoing process throughout the day, instead of siloed into 30- or 60-minute segments (Copple & Bredekamp, 2009). This interdisciplinary context provides the unique opportunity to integrate computer science into other subject domains as well as use computer science as a vehicle for interdisciplinary learning (Bers & Horn, 2010; Morgado, Cruz, & Kahn, 2010).

Computer science is well-suited for early childhood education as it offers a learning environment where young children can "play to learn while learning to play" (Bers et al., 2014, p. 146).

While some work has been conducted with pre-K children, the majority of studies still focus on kindergarten-aged or older students, and few studies provide insight on how educators can implement computer science into their early childhood classroom practices (see DevTech Research Group, 2016 and Morgado et al., 2010 for exceptions). As Fessakis, Gouli, and Mayroudi (2013) explain, "[It] is not the availability of developmentally-appropriate computer programming environments but rather the development of appropriately designed learning activities and supporting material which would have been applied and verified and could be easily integrated in everyday school practice by well informed and prepared teachers" (p. 90). As such, this chapter began the task of articulating specific, research-based ways in which computer science can be integrated into the early childhood learning environment. A review of the research in early childhood education related to computer science can be found in **Appendix D**.

As the examples presented here suggest, integrating computer science-related practices into early childhood education is not a departure from traditional notions of developmentally appropriate practice; rather, early computer science practices support play-based pedagogy and extend what educators are already doing in their classrooms, and they can guide young learners to notice, name, and recognize how computing shapes the modern world. In this way, computer science in pre-K brings to life the discipline of computer science, which is expanded in the larger K–12 Computer Science Framework. Indeed, the framework connections described under each powerful idea in this chapter make explicit how these computer science-related ideas explored in early childhood build the foundation for engaging in more specific concepts and practices outlined in the framework.

By understanding computer science as a discipline, teachers can break down the field into manageable lessons and make the computational world "visible" to students (Welch & Dooley, 2013; Dooley & Welch, 2015). As such, educators provide a way for children to become active participants in digital societies, which ultimately will better position them to become thinkers, creators, and leaders in our increasingly digital world.



References

Bers, M. U., & Horn, M. S. (2010). Tangible programming in early childhood: Revisiting developmental assumptions through new technologies. In I. R. Berson & M. J. Berson (Eds.), *High-tech tots: Childhood in a digital world* (pp. 49–70). Greenwich, CT: Information Age Publishing.

Bers, M.U., Flannery, L.P., Kazakoff, E.R, & Sullivan, A. (2014). Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers & Education*, *72*, 145–157.

Bers, M. U., Ponte, I., Juelich, K., Viera, A., & Schenker, J. (2002). Teachers as designers: Integrating robotics into early childhood education. *Information Technology in Childhood Education Annual, 2002*(1), 123–145.

Boston Public Schools. (2016). *Computer science in BPS.* Retrieved from http://www.bostonpublicschools.org/domain/2054

Camilli, G., Vargas, S., Ryan, S., & Barnett, W. S. (2010). Meta-analysis of the effects of early education interventions on cognitive and social development. *The Teachers College Record*, *112*, 579–620.

Campbell, F. A., Ramey, C. T., Pungello, E., Sparling, J., & Miller-Johnson, S. (2002). Early childhood education: Young adult outcomes from the Abecedarian Project. *Applied Developmental Science*, *6*, 42–57.

Chetty, R., Friedman, J. N., Hilger, N., Saez, E., Schanzenbach, D., & Yagan, D. (2011). How does your kindergarten classroom affect your earnings? Evidence from Project STAR. *The Quarterly Journal of Economics*, *126*(4), 1593–1660. doi: 10.1093/qje/qjr041

Collaborative for Academic, Social, and Emotional Learning. (2012). *Effective social and emotional learning programs—preschool and elementary school education*. Chicago, IL: Author. Retrieved from http://static1.squarespace.com/static/513f79f9e4b05ce7b70e9673/t/526a220de4b00a92c90436ba/1382687245993/2013-casel-guide.pdf

Copple, C., & Bredekamp, S. (2009). *Developmentally appropriate practice in early childhood programs serving children from birth through age 8*. Washington, DC: National Association for the Education of Young Children.

DevTech Research Group. (2016). *Early childhood robotics curriculum*. Medford, MA: Tufts University DevTech Research Group. Retrieved from http://tkroboticsnetwork.ning.com/page/robotics-curriculum

Dillenbourg, P. (1999). *Collaborative learning: Cognitive and computational approaches. Advances in learning and instruction series.* New York, NY: Elsevier Science, Inc.

Dooley, C. M., & Welch, M. M. (2015). Emergent comprehension in a digital world. In A. DeBruin-Parecki & S. Gear (Eds.), *Developing early comprehension: Laying the foundation for reading success*. Baltimore, MD: Brookes.

Elkin, M., Sullivan, A., & Bers, M. U. (2014). Implementing a robotics curriculum in an early childhood Montessori classroom. *Journal of Information Technology Education: Innovations in Practice, 13,* 153–169.

Fessakis, G., Gouli, E., & Mayroudi, E. (2013). Problem solving by 5-6 year old kindergarten children in a computer programming environment: A case study. *Computers & Education, 63*, 87–97.

Goodyear, P., Jones, C., & Thompson, K. (2014). Computer-supported collaborative learning: Instructional approaches, group processes, and educational designs. In J. M. Spector, M. D. Merrill, J. Elen, & M. J. Bishop (Eds.), *Handbook of research on educational communications and technology* (pp. 439–451). New York, NY: Springer Science and Business Media.

Heckman, J. (2006). Skill formation and the economics of investing in disadvantaged children. *Science, 312*, 1900–1902.

Heckman, J., & Masterov, D. (2007). The productivity argument for investing in young children. *Review of Agricultural Economics, 29*, 446–493.

Klem, A. M., & Connell, J. P. (2004). Relationships matter: Linking teacher support to student engagement and achievement. *Journal of School Health, 74*(7), 262–273.

Magnuson, K., Meyers, M. K., Ruhm, C. J., & Waldfogel, J. (2004). Inequality in preschool education and school readiness. *American Educational Research Journal, 41*, 115–157.

Maryland State Department of Education. (2015). *Computer science.* Retrieved from http://archives.marylandpublicschools.org/MSDE/divisions/dccr/cs.html

Mashburn, A. J., Pianta, R. C., Hamre, B. K., Downer, J. T., Barbarin, O. A., Bryant, M., . . . & Howes, C. (2008). Measures of classroom quality in prekindergarten and children's development of academic, language, and social skills. *Child Development, 79*(3), 832–749. doi: 10.1111/j.1467-8624.2008.01154.x

Morgado, L., Cruz, M., & Kahn K. (2010). Preschool cookbook of computer programming topics. *Australasian Journal of Educational Technology, 26*(3), 309–326.

National Research Council. (2009). *Mathematics learning in early childhood: Paths toward excellence and equity.* Committee on Early Childhood Mathematics. C. T. Cross, T. A. Woods, & H. Schweingruber (Eds.). Center for Education. Division of Behavioral and Social Sciences and Education. Washington, DC: The National Academies Press.

National Science Board. (2012). *Science and engineering indicators 2012* (NSB 12-01). Arlington, VA: National Science Foundation. Retrieved from https://www.nsf.gov/statistics/seind12/pdf/seind12.pdf

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas.* NY: Basic Books.

Pianta, R. C., Barnett, W. S., Burchinal, M., & Thornberg, K. R. (2011). The effect of preschool education: What we know, how public policy is or is not aligned with the evidence base, and what we need to know. *Psychological Sciences in the Public Interest, 10*(2), 49–88. doi: 10.1177/1529100610381908

Pianta, R. C., La Paro, K. M., Payne, C., Cox, M. J., & Bradley, R. (2002). The relation of kindergarten classroom environment to teacher, family, and school characteristics and child outcomes. *The Elementary School Journal*, *102*(3), 225–238.

Resnick, M. (2003), Playful learning and creative societies. *Education Update, 8*(6). Retrieved May 1, 2009 from http://web.media.mit.edu/~mres/papers/education-update.pdf

Schweinhart, L. J., Montie, J., Xiang, Z., Barnett, W. S., Belfield, C. R., & Nores, M. (2011). *Lifetime effects: The High/Scope Perry preschool study through age 40: Summary, conclusions, and frequently asked questions.* Ypsilanti, MI: High/Scope Press. Retrieved from http://www.highscope.org/file/Research/PerryProject/specialsummary_rev2011_02_2.pdf

Smith, M. (2016). *Computer science for all*. Washington, DC: Office of Science and Technology Policy, Executive Office of the President. Retrieved from https://www.whitehouse.gov/blog/2016/01/30/computer-science-all

Strawhacker, A. L., & Bers, M. U. (2014, August). *ScratchJr: Computer programming in early childhood education as a pathway to academic readiness and success.* Poster presented at DR K–12 PI Meeting, Washington, DC.

Twarek, B. (2015). *Pre-K to 12 computer science scope and sequence*. Retrieved from http://www.csinsf.org/curriculum.html

U.S. Congress Joint Economic Committee. (2012). *STEM education: Preparing for the jobs of the future.* Washington, DC: Author. Retrieved from http://www.jec.senate.gov/public/_cache/files/6aaa7e1f-9586-47be-82e7-326f47658320/stem-education---preparing-for-the-jobs-of-the-future-.pdf

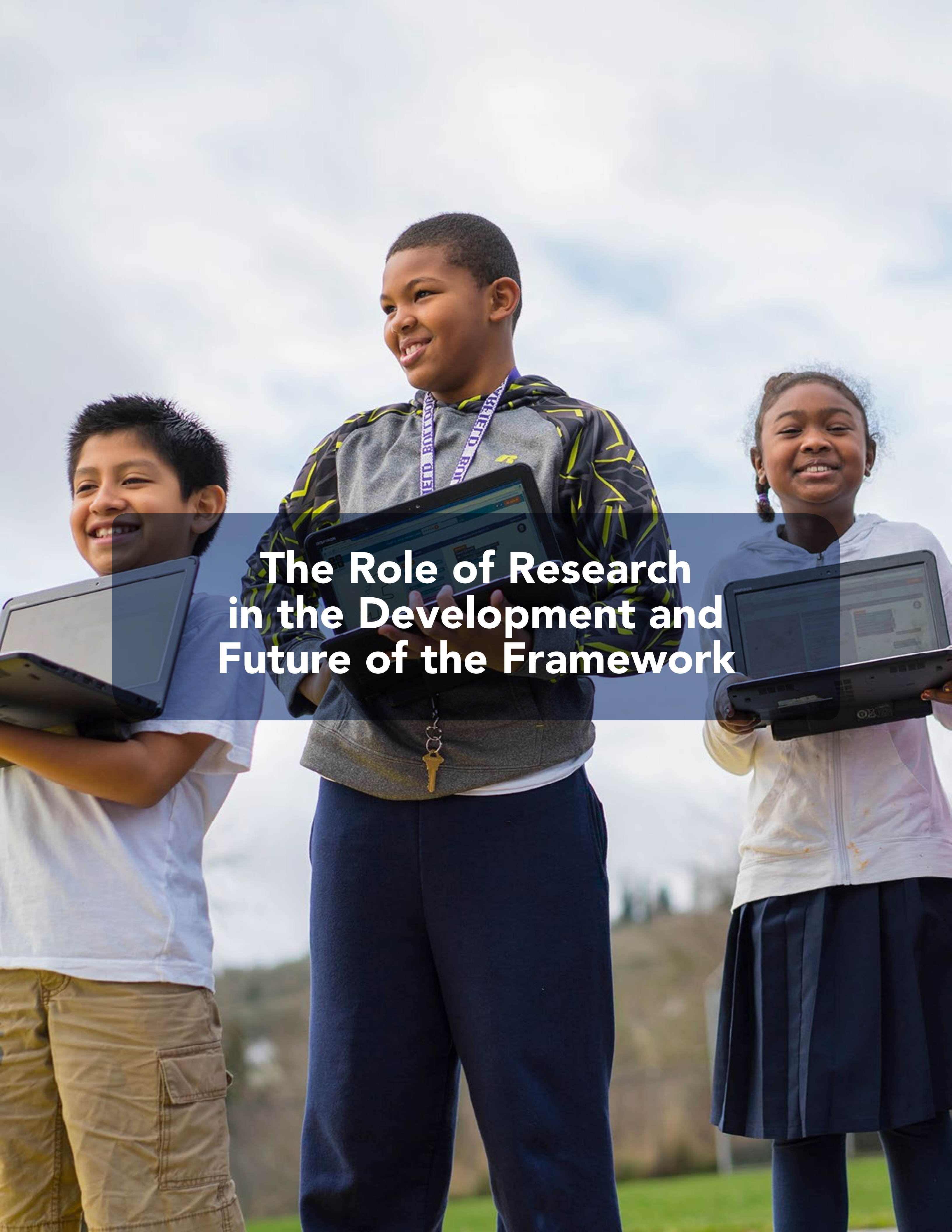
U.S. President's Council of Advisors on Science and Technology. (2010). *Prepare and inspire: K–12 education in science, technology, engineering, and math (STEM) for America's future*. Washington, DC: Executive Office of the President. Retrieved from https://www.whitehouse.gov/sites/default/files/microsites/ostp/pcast-stemed-report.pdf

Weissberg, R. P., & Cascarino, J. (2013). Academic + social-emotional learning = national priority. *Phi Delta Kappan, 95*(2), 8–13.

Weissberg, R. P., Durkak, J. A., Doitrovich, C. E., & Gullota, T. P. (2015). Social emotional learning: Past, present, and future. In J. A. Durlak, C. E. Domitrovich, R. P. Weissberg, & T. P. Gullota (Eds.), *Handbook of social and emotional learning: Research and practice* (pp. 3–19). New York, NY: Guilford.

Welch, M. M., & Dooley, C. M. (2013, May). Digital equity for young children: A question of participation. *Learning and Leading with Technology,* 28–29.

Yoshikawa, H., Weiland, C., Brooks-Gunn, J., Burchinal, M. R., Espinosa, L. M., Gormley, W. T., . . . & Zazlow, M. J. (2013). *Investing in our future: The evidence base on preschool education.* Ann Arbor, MI: Society for Research in Child Development; New York, NY: Foundation for Child Development.



The Role of Research in the Development and Future of the Framework

10

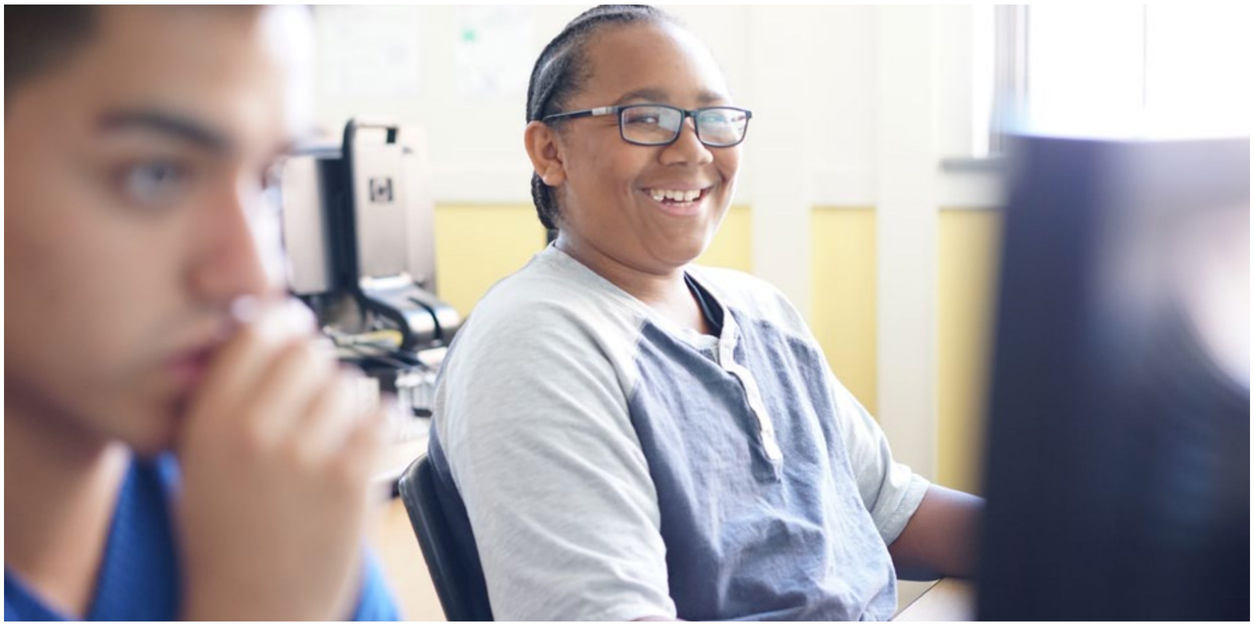
The Role of Research in the Development and Future of the Framework

The K–12 Computer Science Framework was informed by a growing body of research on computer science education, as well as broader literature from the fields of science, technology, engineering, and mathematics (STEM) education. In particular, the concepts, practices, and learning progressions at the heart of the framework were influenced by research on such topics as how students learn computer science, how they interact with one another in computing environments, and at what age they demonstrate proficiency in specific concepts.

The framework was informed by a growing body of research on computer science education.

As computer science education is a young field relative to other K–12 subject areas, the accompanying educational research has a number of areas in which to improve, and important questions remain to be addressed (Lishinski, Good, Sands, & Yadav, 2016). High public demand for computer science learning opportunities and a shortage of trained computer science education researchers have led to much research being performed by practitioners (i.e., computer science teachers), if research is performed at all (Franklin, 2015). In addition, the fast-paced technological developments of the past few decades pose another challenge: research is unable to keep up with new technologies and the increasingly diverse ways in which we can interact with them. Indeed, the broader literature on technology in education has shifted away from a focus on the technology medium to a systemic view on media content, context, and users in an effort to recognize and understand the complex interactions afforded by novel technologies (e.g., Guernsey, 2012; Valkenburg & Peter, 2013).

Although technology continues to change, the conceptual foundations of computer science education and the challenges faced by educators and learners in the field remain the same. For example, the establishment of block-based programming tools has not removed the conceptual difficulties of learning and teaching programming concepts. Likewise, misconceptions about specific concepts, such as variables, loops, and boolean logic, that were articulated in research conducted in the 1980s (e.g., De Boulay, 1989; Pea & Kurland, 1984; Pea, Soloway, & Spohrer, 1987; Soloway, 1986) have not dissipated in more recently developed computing environments (e.g., Cooper, Grover, Guzdial, & Simon, 2014; Grover, Pea, & Cooper, 2016; Guzdial, 2016).



The writers consulted relevant literature to inform early drafts of the framework statements. Later they used a more structured approach for collecting and incorporating research into the specific concept and practice statements as well as for determining the structure and placement of such statements within the K–12 learning progressions. This approach involved searching the existing research in the field, interviewing experts in key research areas, identifying research related to the core concepts and practices in the framework, mapping the research to specific concept and practice statements, and making modifications based on research findings. Considering the maturity of computer science as a K–12 discipline and the charge of delineating the core concepts and practices in computer science, research was often insufficient or still developing. In those cases, the writing team relied on its own expertise and years of experience in K–12 computer science education, as well as the collective expertise of the computer science education and research communities.

Advisors played a significant role in constructing the research base used to inform the framework, as well as providing guidance beyond the available research. As leaders in the fields of computer science, education, and computer science education research, the advisors brought diverse knowledge and perspectives from a variety of research areas, such as diversity and equity, elementary education, computer-supported collaborative learning, human–computer interaction, social computing, teacher preparation, computational thinking, and interdisciplinary integration. Their work, and the work of their research partners, is reflected in the research base that informed the framework's vision, structure, and content.

This chapter is not intended to be a complete review of all the research pertaining to K–12 computer science education. Several recent reviews on K–12 computational thinking research (Grover & Pea, 2013) and computing education (Guzdial, 2016) provide broader insights that are beyond the scope of this chapter but were referenced during the framework development process. Rather, the intent of this chapter is to describe examples of the ways in which research was used to inform the framework statements and acknowledge and offer considerations for addressing gaps in the current computer science education research literature that emerged through the framework development process. The end of this chapter includes considerations for examining policy and implementation issues.

Research That Informs the Framework

The following sections provide examples of the research that guided the development of the framework's core concepts and practices, as well as the specific concept and practice statements. **Appendix E** contains a full list of references that the framework writers referred to in their development of the framework content.

Research That Informs the Concepts

The framework concept statements were arranged according to learning progressions—conceptual milestones on a path that directs a learner from basic ideas to more sophisticated knowledge. Fundamentally, learning progressions are built on research from developmental psychology pertaining to the broader cognitive, social, and behavioral processes that occur throughout a child's development as well as the logical structure of specific content within a subject domain. Indeed, content-specific learning progressions based on such research serve as the foundation for several sets of state math and science standards (NGA Center for Best Practices & CCSSO, 2010; NGSS Lead States, 2013).

To develop the computer science learning progressions, the framework writers and advisors identified emergent core ideas that can be introduced early in a student's education and built upon in later years. While limited in comparison to other subject domains, computer science learning progression research was consulted where available.

Even when evidence for developing learning progressions exists, challenges remain regarding capturing the complex relationship between students' prior knowledge, cognitive abilities, and developmental changes over time and then generalizing the same learning progression to a diverse population of students (Duschl, Schweingruber, & Shouse, 2007). Learning progressions are especially pertinent for computer science education, where efforts to build coherent K–12 instructional pathways for *all* students are at the forefront of many local and state initiatives (Cernavskis, 2015). Suggestions for further research to address the challenges in developing learning progressions are discussed later in this chapter.

For areas where computer science-specific learning progressions were nonexistent, the writers drew on learning progressions in science and mathematics to guide the placement of related computer science concepts in a particular grade band. As previously described in the **Development Process** chapter, the placement of procedural abstraction, in which procedures use variables as parameters to generalize behavior, as an expectation by the end of eighth grade was informed by the placement of a related concept in mathematics learning progressions—writing equations with variables. An understanding of bits (basic units of digital information) was placed as an expectation across upper elementary and middle school grade bands based on the placement of the analogous concepts of particles and atoms in science learning progressions. In some cases science learning progressions were used to inform the placement of very similar concepts, such as models and simulations. As in science and mathematics, the writers and advisors recognize the need for computer science learning progressions to continue to evolve as new research emerges.

Writers drew on learning progressions in science and mathematics to guide the placement of related computer science concepts.

To increase support for the appropriate placement of concept statements, the learning progressions were also informed by the experiences of the framework writers and advisors with K–12 students from diverse populations as well as the experiences of the broader group of K–12 educators and researchers who reviewed the framework. Below are several more examples of how research informed the development of concept statements.

Example 1: Algorithms and programming

Current research on computational thinking, algorithm development, and early programming was informative for the *Algorithms and Programming* statements. Research on fourth graders' attempts to develop step-by-step instructions and algorithms suggests what students know prior to formal instruction and potential opportunities for developmentally appropriate curricula (Dwyer, Hill, Carpenter, Harlow, & Franklin, 2014). This research into early algorithmic development is an important aspect of this expectation by the end of Grade 5 in the *Algorithms and Programming* core concept: "People develop programs using an iterative process involving design, implementation, and review. Design often involves reusing existing code or remixing other programs within a community. People continuously review whether programs work as expected, and they fix, or debug, parts that do not. Repeating these steps enables people to refine and improve programs" (3–5.Algorithms and Programming.Program Development).

Additional research informed the grade-band placement of specific programming concepts in the framework. Seiter and Foreman (2013) collected projects created in a block-based programming environment from students in Grades 1 through 6 and evaluated each project's demonstration of computational thinking concepts. They found that programming actions associated with data representation, such as variable referencing and assignment, are not significantly present until later grades (i.e., Grades 5 and 6). This finding guided the placement of variables in the 6–8 grade band of the framework. Seiter and Foreman also found that uses of conditional logic begin to appear in Grade 3 and increase through Grade 6, which guided the placement of conditional statements in the 3–5 grade band of the framework.

The methodology used by Seiter and Foreman to analyze students' projects evokes questions of the conditions under which it works and does not work. Future research is required to validate whether analyzing students' program code can accurately infer what they understand. For example, Aivaloglou and Hermans (2016) analyzed 250,000 Scratch projects and found that procedures and conditional loops were not commonly used. These results do not mean that students are not able to use procedures or conditional loops or that these concepts should be taught at a later age; the absence of these constructs may be due to the specific programming environment used or simply students' desire to use the tool in a particular way. A promising avenue of research is using written reflections and student interviews to evaluate computational artifacts. For example, Brennan and Resnick (2012) have explored the use of artifact-based interviews to assess what students understand and their rationale for programming decisions.

Example 2: Impacts of Computing

There is also research on the influence of culture on individuals' interactions with technology (e.g., Evers & Day, 1997; Leidner & Kayworth, 2006). This research primarily informs the *Impacts of Computing* statements in the Culture subconcept. For example, the following statement draws upon the findings of Evers and Day (1997) that culture influences design preferences and acceptance of an interface: "The development and modification of computing technology is driven by people's needs and wants and can affect groups differently. Computing technologies influence, and are influenced by, cultural practices" (3–5.Impacts of Computing.Culture). The next statement in the Culture progression draws upon the work of Leidner and Kayworth (2006), whose review of culture and information technology literature concluded that culture influences information and technology and vice versa: "Advancements in computing technology change people's everyday activities. Society is faced with tradeoffs due to the increasing globalization and automation that computing brings" (6–8.Impacts of Computing.Culture).

In each of these examples, it should be noted that this research relates to and informs the concepts in the framework but that more research is needed to specifically target the developmental appropriateness of the concepts within the context of a learning progression.

Research That Informs the Practices

Key research studies provided a foundation for the identification of the core practices in the framework as well as the practice statements in each core practice. Weintrop et al. (2015) define a computational thinking taxonomy for science and mathematics consisting of practices in four main categories: data, modeling and simulation, computational problem solving, and systems thinking. This research was helpful in addressing the framework's goal of empowering students to learn, perform, and express themselves in other fields and interests by helping identify computational practices that had application within and beyond computer science. Another report that influenced the set of core practices in the framework was *Assessment Design Patterns for Computational Thinking Practices in*

Secondary Computer Science (Bienkowski, Snow, Rutstein, & Grover, 2015). In this report, Bienkowski et al. describe their approach to designing assessments to measure students' computational thinking abilities. This work included the creation of a test domain consisting of key constructs, such as "Design and apply abstractions and models" and "Collaborate with peers on computing activities" (p. 10), many of which inspired and overlap with the framework's practices.

The following paragraphs provide some examples of research that informed specific core practices and practice statements. It should be noted that although these studies informed the practices in the framework, some require additional research to provide direct support within a K–12 context.

Research focusing on students traditionally underrepresented in computer science, such as students with disabilities, females, and students from some minority groups, influenced the identification of the first practice, *Fostering an Inclusive Computing Culture*. Ladner and Israel (2016) argue for considerations for including all students in computer science and outline challenges that include the need for culturally relevant pedagogy and increasing relevance for nontraditional computer science students. Other work identifies strategies and resources that teachers can integrate into lessons to encourage a diverse set of students to participate and learn (Israel, Wherfel, Pearson, Shehab, & Tapia, 2015). Although not in a classroom context, workplace research about the greater effectiveness of culturally diverse groups as compared to culturally homogeneous groups (e.g., Watson, Kumar, & Michaelsen, 1993) informed this practice statement: "Include the unique perspectives of others and reflect on one's own perspectives when designing and developing computational products" (P1.Fostering an Inclusive Computing Culture.1). Research also suggests that self-determination (or self-advocacy) is critically important for students with disabilities (Wehmeyer, 2015; Wehmeyer et al., 2012) and is reflected in the third practice statement, "Employ self- and peer-advocacy to address bias in interactions, product design, and development methods" (P1.Fostering an Inclusive Computing Culture.3).

Specific research related to collaboration informed the statements in *Collaborating Around Computing*. For example, previously mentioned research found that culturally diverse teams in the workplace are more effective at identifying problem perspectives and generating alternative solutions than homogeneous teams (Watson, Kumar, & Michaelsen, 1993). This research influenced the inclusion of this statement: "Cultivate working relationships with individuals possessing diverse perspectives, skills, and personalities" (P2.Collaborating Around Computing.1). Although research points to students of different abilities learning more in similar or mixed-ability levels (i.e., lower ability students learn more in mixed-ability groups) (Lou, Abrami, & d'Apollonia, 2001), *Collaborating Around Computing* is about groups with diverse skills, rather than abilities. Small-scale research on pair programming, a pedagogical technique for groups working at the same computer, suggests that "less equitable pairs sought to complete tasks quickly and this may have led to patterns of marginalization and domination" and that "[t]hese findings are important for understanding mechanisms of inequity and designing equitable collaboration practices in computer science" (Lewis & Shah, 2015, p. 41). This research informed the following practice statement: "Create team norms, expectations, and equitable workloads to increase efficiency and effectiveness" (P2.Collaborating Around Computing.2). Further research is required to study the effect of this practice on learning computer science, in addition to collaboration.

There is a body of literature around students' creation of computational artifacts, such as games, apps, and animations. This research provides some of the foundation for the statements in *Creating Computational Artifacts***.** For example, a study of students' programming projects in the Scratch environment suggests that learning and computational thinking are supported through remixing parts of existing artifacts (Dasgupta, Hale, Monroy-Hernández, & Hill, 2016). Other work suggests that a three-stage progression that helps learners advance from the role of user to modifier to creator of computational artifacts can also build computational thinking (Lee et al., 2011). These examples of research inform the act of remixing and inform the learning progression for this practice statement: "Modify an existing artifact to improve or customize it" (P5.Creating Computational Artifacts.3).

Additional literature in the field provides general guidance for other core practices. For example, arguments about the benefits and importance of teaching and learning abstraction (e.g., Sprague & Schahczenski, 2002) generally inform *Developing and Using Abstractions* as a core practice. The second practice statement in *Recognizing and Defining Computational Problems*, "Decompose complex real-world problems into manageable subproblems that could integrate existing solutions or procedures," is informed by the benefits of breaking down a program and labeling sections with subgoals (Morrison, Margulieux, & Guzdial, 2015). Research on university students in introductory programming classes, most of whom were computer science and engineering majors, shows positive outcomes associated with testing programs early in the development process (Buffardi & Edwards, 2013) and informs the practice of *Testing and Refining Computational Artifacts*. Further research is required to validate this outcome with K–12 students in a general population.

Research Agenda

A vision of computer science education, at the scale, rigor, and level of coherence that the framework promotes, is a new frontier for the American education system. Research will be required to both evaluate the implementation of K–12 computer science and evolve the goals and scope of computer science education to reflect the lessons learned from the early years of implementation. Further research will illuminate more effective ways for students of all ages to learn computer science and for teachers to deliver instruction and provide meaningful experiences for all students.

The need for research on K–12 computer science education provides a unique opportunity for researchers in computing education and learning sciences. The following sections describe a suggested research agenda that is guided by areas identified in the framework development process as lacking a solid research foundation or requiring further study: equity and access, learning progressions, pedagogical content knowledge, and facilitating learning in other disciplines. These categories do not stand alone; research in these areas can and should overlap and must be explored together to advance computer science education as a whole.

Learner-Centered Design of Computing Education (Guzdial, 2016) provides a research review of issues such as how computer science is currently taught, why it should be taught, and challenges in teaching it to all students. Chapter seven includes additional K–12 computer science research questions pertaining to elementary and secondary school.

The primary goal of the following research considerations is to inform future iterations of the current framework, which, in turn, can inform the broader field of K–12 computer science education. The suggestions are guided by the framework's vision of K–12 computer science and intentionally adopt a policy and practice lens in an effort to ensure that future research has real-world, practical implications that drive computer science education teaching and learning in classroom.

Equity and Access

The framework's intent is to provide a computer science foundation for all students regardless of their background, physical challenges, learning differences, or future career aspirations. Despite the growing demand for individuals with computer science skills (BLS, 2015) and an increased focus on achieving a more diverse cohort of computer scientists (Sullivan, 2014), the field remains predominantly male and disproportionately White and Asian (Marcus, 2015). Indeed, of all the STEM fields, computer science consistently has the largest gender disparities, with women making up only 23% of the computing industry—a figure that has not changed for more than a decade (BLS, 2015; NSF, 2014). Similarly, Black and Hispanic individuals make up only 14% of the computing field, and they are less likely to have access to computer science courses and technology (Google & Gallup, 2015b; Margolis, Estrella, Goode, Holme, & Nao, 2010).

Previous work on equity and access in K–12 computer science has approached the topic from a variety of angles and provides some initial insight to build on in future research endeavors. Projects have explored interpersonal communication (Lewis & Shah, 2015), systemic issues and preparatory privilege within schools (Margolis et al., 2010), and developing culturally relevant curriculum and evaluating its effectiveness at mitigating inequity (Margolis et al., 2012). Based on a review of literature regarding gender disparities in STEM, Blickenstaff (2005) suggested seven practical ways for educators, administrators, and curriculum developers to create more positive learning environments for all students. Above and beyond providing more equitable access to courses and resources, Blickenstaff (2005) suggests creating curricular materials that emphasize the real-world impact of STEM, including how the fields can help improve quality of life and help solve societal issues. Others have noted the critical role that societal stereotypes play in dissuading girls and students of color from entering the computer science field (e.g., Goode, Estrella, & Margolis, 2006; Google & Gallup, 2015a).

Researchers suggest creating curriculum that emphasizes the real-world impact of STEM, including how the fields can help improve quality of life and help solve societal issues.

Unless K–12 computer science education grows in an equitable way, there is little reason to believe the call for increased diversity in the field will be answered. To ensure equity in computer science education and technology-related industries, the research field can work toward an evidence-based understanding of the factors that support inclusive computing environments as well as practical solutions for increasing access, interest, and efficacy in traditionally underrepresented student populations.

At the most basic level, researchers can track not only overall K–12 computer science course enrollment numbers but also the diversity of participation, including any differences in access and engagement for students from traditionally underrepresented groups. This work will become even more critical as state and district implementation efforts go to scale, opening up the very real possibility of mismatched intentions (i.e., computer science for *all*) and reality (i.e., computer science for a select few). Researchers can probe deeper into understanding the role of teachers and school administrators, as well as characteristics of the school-, district-, and state-level contexts, to better understand how individual characteristics and dispositions (e.g., socioeconomic status, race/ethnicity, gender, spoken language, self-efficacy, prior experience, etc.) intersect with these relational and institutional contexts to shape students' experience, success, and sustained engagement in computer science. This work includes studying the classroom experiences of students from underrepresented groups; potential differences in levels of interest, self-efficacy, and achievement; and the effectiveness of approaches aimed at proactively addressing such gaps.

By monitoring K–12 computer science education trends now, researchers set the stage for longitudinal studies that can provide insight into the long-term implications of students' computer science experiences as well as a better understanding of the differences between those who choose to pursue computer science and those who take other academic and career paths. Such long-term data will be critical for understanding the most influential factors—such as access to computer science opportunities, student experiences, and interest and self-efficacy in computer science—for engaging and sustaining a diverse computing industry.

More information about equity and access can be found in the **Equity in Computer Science Education** chapter.

Learning Progressions

Learning progressions provide organizational structure to the framework and are critical for ensuring coherence throughout a computer science education pathway. Learning progressions have been developed in science and mathematics education research (e.g., Achieve, 2015) but have room to grow in computer science education (Seiter & Foreman, 2013). Learning progression research is crucial for informing future versions of the framework and influencing student learning and teacher practices. Students are directly affected by developmentally inappropriate and incomplete learning progressions, as they could lead to poor transfer of knowledge (Perkins & Salomon, 1988) or the development of misconceptions. As suggested in *Taking Science to School,* "well-tested ideas about learning progressions could provide much needed guidance for both the design of instructional sequences and large-scale and classroom-based assessments" (NRC, 2007, p. 220).

Importantly, the development of learning progressions does not merely entail decreasing the rigor of content and processes found appropriate for older students and pushing it down to younger grade levels. Also, it cannot be assumed that what works in high school or postsecondary contexts will work—and is developmentally appropriate—for the primary school context. Indeed, the framework writers devoted critical attention to ensuring that the progressions made both logical sense in terms of computer science content and developmental sense to match the different levels of learners across the K–12 grade span. Thus, while prior research on the postsecondary computer science education environment can inform current and future studies in the K–12 context, simply replicating this work is insufficient for the development of sound computer science learning progressions at the K–12 level.

In the K–12 environment, research on computer science learning progressions has just begun, and much of the current work focuses on either broad computational thinking practices or very specific computing concepts. For example, work by Dwyer and colleagues (2014) investigated fourth graders' prior knowledge before engaging in a computational thinking curriculum to construct beginning anchor points for elementary learning progressions. Follow-up studies examined students' use of interactive control structures in a block-based programming language and suggested that the explicit instruction of programming elements related to user-centered design (e.g., handling events such as mouse clicks) should wait until the fifth grade (Hansen et al., 2015; Hansen, Iveland, Carlin, Harlow, & Franklin, 2016). It should be noted that these results are highly contextualized to the programming environment as outcomes may vary with different environments. This finding also contrasts with research on children's creation of digital games and programs, which illustrates that children can engage in user-centered design processes alongside learning novel computing content and practices (e.g., Brennan & Resnick, 2012; Kafai & Burke, 2015). Further, while some work has investigated computer science learning trajectories across multiple grade levels (e.g., Brennan & Resnick, 2012; Seiter & Foreman, 2013), there remains no consistent basis for learning progressions across the entire K–12 range.

The framework proposes a K–12 learning progression that can be evaluated and built upon through future research.

These mixed and limited findings directly point to the need for more research to understand not just what computer science children are cognitively capable of doing and when but also the most effective ways for such learning to occur for a larger and more diverse student population. The framework itself proposes a K–12 learning progression that can be evaluated and built upon through future research. These studies can help identify and validate learning benchmarks within and across grade bands based on the framework's progressions, as well as use the framework as a starting point to propose alternative learning progressions to evaluate. This line of work can also investigate the specific concepts that students have trouble learning, potential misconceptions, and the age at which these difficulties begin to help inform future revisions to the framework progressions.

Related, studies can evaluate how specific teaching practices and pedagogical approaches influence different types of learning outcomes in computer science. Research could address the effects of highly scaffolded programming environments compared to more open and "free play" contexts; the influence of social computing (e.g., pair programming) contexts; and the impact of "plugged" versus "unplugged" curricula on students' computer science learning and dispositions. Importantly, this work cannot be siloed to one age or grade level but should be studied with a broader goal of understanding if and how certain teaching and pedagogical practices differentially influence student outcomes at a variety of developmental and knowledge levels.

Similarly, studies of younger children can investigate how early acquisition of certain computer science concepts and practices influences later learning experiences. In addition to investigating whether and how much computer science knowledge children retain throughout the K–12 experience, even when there are gaps in learning (Guzdial, 2016, p. 100), longitudinal studies can also emphasize the role that early computer science experiences play in the development of future computer science attitudes and self-efficacy.

Further, future research can explore the influence of different learning progressions on students' attitudes toward and self-efficacy in computer science in addition to achievement outcomes to better understand if and how different pathways influence future engagement and success in the computing field. This line of work can be extended to understand such effects for different student populations. Then it can inform the development of adaptable and flexible learning progressions to ensure that all students have sound computer science learning opportunities across their elementary and secondary education.

Pedagogical Content Knowledge

Pedagogical content knowledge (PCK) is the knowledge that teachers have about teaching as a practice combined with their subject expertise (Shulman, 1986). Additionally, the importance of contextual factors—including teachers' attitudes, self-efficacy, and value judgments—as well as domain specificity permeate more recent conceptualizations of PCK (Bender et al., 2015). In computer science, PCK might include best practices for explaining computer science concepts, how to address common misconceptions that teachers and students might have, or understanding how to create an inclusive computing environment with students from varied backgrounds and prior experience.

Exploring PCK is particularly critical for the scaling up of computer science education, especially given the varied (and more often lack of) computer science teacher certification requirements (CSTA Teacher Certification Task Force, 2008). The lack of consistent requirements leaves schools and districts with the challenge of trying to offer computer science courses without sufficiently trained teachers. Indeed, one of the major barriers to offering computer science in K–12 classrooms, according to school leaders, is a lack of teachers with the necessary skills to teach it (Gallup, 2015b). Achieving the framework's vision of computer science education for all students requires that the research and professional development communities build teachers' PCK related to each core area in the framework. This requirement applies to preservice computer science teachers as well as inservice teachers integrating computer science into other subjects.

Achieving the framework's vision requires that the research and professional development communities build teachers' PCK.

Prior research on teachers' computer science PCK primarily exists at the postsecondary level and focuses on teaching introductory courses. For example, research has looked at the effectiveness of sharing PCK through examples that present an exercise and describe correct and incorrect answers, expected misconceptions, appropriate feedback to give to students, and methods for provoking discussion (Koppelman, 2007; Koppelman, 2008). Other research has looked at specific techniques for decreasing cognitive load and increasing performance when teaching programming through worked examples, such as prelabeled sections of code or asking students to generate their own labels
(Morrison, Margulieux, & Guzdial, 2015).

Foundational steps have recently been taken to create a theoretical model for framing primary and secondary teachers' computer science PCK (Bender et al., 2015; Hubwieser, Magenheim, Mühling, & Ruf, 2013; Saeli, Perrenet, Jochems, & Zwaneveld, 2011). Saeli and colleagues (2011) revealed coaching as a critical component of secondary school teachers' PCK specific to teaching programming, noting that coaching can help drive students' problem-solving, reflection, and algorithmic thinking processes. The authors also delineated common difficulties for students, including translating human language into language that is understandable by a computer and being able to shift between different programming languages (Saeli et al., 2011). In a follow-up study, Saeli, Perrenet, Jochems, and Zwaneveld (2012) further conceptualized PCK for programming, suggesting secondary school teachers use simple programming languages to help offset syntactical challenges as well as provide several meaningful problems for students to solve instead of many problems that lack real-world and personally relevant meaning.

Work by Hubwieser and colleagues (2013) and Bender and colleagues (2015) to develop a competency model of teaching computer science is perhaps the most rigorous conceptualization of K–12 computer science PCK to date. The authors posit five main content dimensions and five categories of beliefs/motivational orientations required for effective computer science teaching, as well as provide specific descriptions of what such competencies look like in practice. For example, Content Dimension 3: Learner-Related Issues describes the need for teachers to understand how to "adapt their teaching methods, contexts, content representations, and material to the different requirements that occur due to the diversity of computer science students" (Bender et al., 2015, p. 528). Similarly, Beliefs/Motivational Orientation Category 2: Beliefs about Teaching and Learning in Computer Science suggests teachers should be "convinced that students are learning in an autonomous way and by critically approaching computer science contents" to be effective computer science teachers (Bender et al., 2015, p. 528).

Recent projects, such as CSTeachingTips.org, have taken an applied research approach to document PCK by collecting and reporting best practices from the field (Lewis, 2016). Additionally, the computer science education community is exploring best practices, such as pair programming (e.g., Denner, Werner, Campe, & Ortiz, 2014; Hanks, 2008), peer instruction (Kothiyal, Majumdar, Murthy, & Iyer, 2013), and identifying and addressing misconceptions (Ohrndorf, 2015).

Despite the work already being done around computer science PCK, this body of research remains nascent, and questions regarding computer science PCK at different grade levels, the influence of contextual factors on teacher practices and student learning, and the role of preservice and inservice professional development remain unanswered. Although research has conceptualized general models of K–12 or 9–12 computer science PCK (e.g., Bender et al., 2015; Hubwieser et al., 2013; Saeli et al., 2011), specific teacher competencies within each grade band are noticeably absent. Thus, future researchers can evaluate current PCK models across different grade levels, as well as propose and test new models that provide a more granular understanding of PCK specific to elementary, middle, and high school teachers.

Understanding PCK models becomes even more relevant for the younger grade levels, where if computer science is taught currently, it is integrated into other subject areas. Yadav, Mayfield, Zhou, Hambrusch, and Korb (2014) suggest teachers should learn computational thinking concepts themselves to be better prepared to integrate it into their teaching and better able to articulate broader uses of it as a problem-solving tool in other disciplines. Future studies can extend this work by exploring how teachers from a variety of disciplines incorporate and implement computational thinking practices in the classroom and the specific training and preparation they need to do so. Additionally, research can investigate the most effective integration practices for driving student interest and learning in computer science and in other subject areas as well.

As learning computer science becomes increasingly popular and moves from an elective to a mainstream core offering, computer science teachers will need to understand not only what aspects of computer science students may struggle to learn but also how to apply established techniques for differentiating instruction to meet the needs of a more diverse population. Recent research has explored the use of Universal Design for Learning to develop and refine introductory computer science experiences for a wide range of learners (Hansen, Hansen, Dwyer, Harlow, & Franklin, 2016), but more work is necessary to fully understand the key components of PCK necessary for ensuring that all students not only have access to computer science but also have differentiated opportunities to engage in the field that meet their unique learning needs. As part of this line of inquiry, research can also explore the unique instructional support and resources needed for students with and without prior computing experience. This support might include developing and testing supplemental materials for both ends of the spectrum (e.g., less complex materials for those without experience and more advanced opportunities for those with more extensive experience), as well as exploring the role that extended learning opportunities, such as afterschool clubs, camps, internships, and college-level course taking, can have on student outcomes.

Further, given continued misperceptions of what computer science is, combined with rampant stereotypes and a clear lack of diversity in the computing field, understanding how teachers view and think of computer science is just as important to consider as those of students. If teachers are the ones creating the computing environment, they are also the ones who may be reinstating cultural norms and stereotypes in their students. A sense of belonging in STEM classrooms is a strong predictor of women's interest and motivation in the fields (Good, Rattan, & Dweck, 2012; Smith, Lewis, Hawthorne, & Hodges, 2013), such that ensuring teachers have strong PCK can have direct impacts on the way they approach computer science and the classroom environment they create. Future research can explore how teachers' computer science attitudes and expectations affect student learning in and attitudes toward computer science. Part of this work could include research on preservice and inservice professional development programs to understand how they can help teachers develop strong self-efficacy and positive attitudes toward computer science alongside helping build their content knowledge and implementation practices.

Facilitating Learning in Other Disciplines

As schools work to implement computer science education pathways, many teachers, especially those teaching in K–8 classrooms, may be asked to integrate computer science content into their current curriculum. Additionally, middle or high school teachers from other disciplines may be asked to teach a dedicated computer science course, making it imperative for these teachers to know how to apply knowledge and skills from their primary disciplines to teaching computer science and vice versa. Significantly, computing is a medium that can facilitate and support learning across disciplines and throughout the traditional elementary and middle school curriculum. More than just creating interdisciplinary connections, computer science is the basis for a form of expression, just like writing and art—empowering students' voice beyond the stringent subject domains of "literacy" or "programming."

Computer science is the basis for a form of expression, just like writing and art.

Whether integrating computer science across the curriculum or teaching it on its own, teachers will require training and support beyond content knowledge—they will have to deal with not only learning a new subject area but also potentially changing pedagogical practices to reflect a more interdisciplinary and student-centered learning environment. As with any new education innovation, strengthening teachers' confidence in their ability to effectively teach computer science as well as clearly communicating the value of computer science for student learning will be necessary to ensure high-quality instruction with meaningful impacts.

Further, outside expertise from educators and scholars from other disciplines is critical as the K–12 computer science knowledge base grows, and collaborations with other fields can lead to interesting, unforeseen findings. Collaborations among researchers from the fields of education and computer science are important to building a rigorous and meaningful research community that has broader impact within and beyond these disciplines.

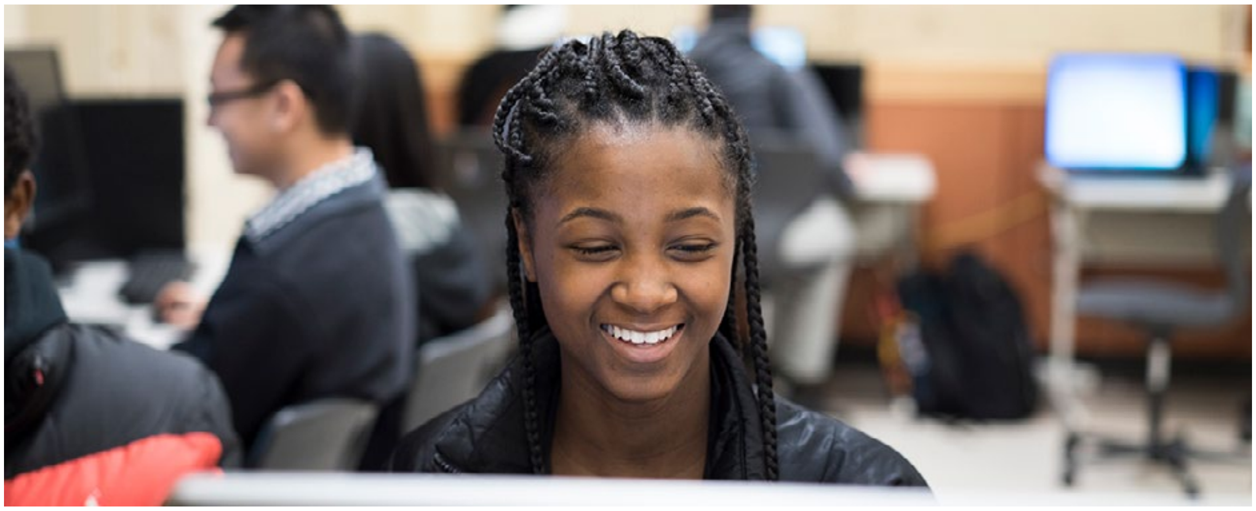
At the postsecondary level, studies have investigated the feasibility of integrating computer science with other subject areas, including bioinformatics (LeBlanc & Dyer, 2004), as well as how the use of multiple forms of media, such as images and sounds, can enhance introductory college-level computer science courses (Guzdial, 2013). Schulz and Pinkwart (2015) also explored how physical computing could be integrated into preservice high school-level STEM education courses, finding that student teachers believed that the essential competencies for computing, physics, and biology were sufficiently covered in the curriculum. At the K–12 level, researchers have also investigated the feasibility of integrated computer science curricula. Goldschmidt, MacDonald, O'Rourke, and Milonovich (2011) discussed simple ways in which computer science concepts can be integrated across *all* subject domains at the K–12 level, including gym, art, and music. A similar approach was taken by Goldberg, Grunwald, Lewis, Feld, and Hug (2012), who brought computing to classes that middle and high school students already take—including art, health, and social studies. Though the authors did not explicitly measure student-level outcomes, teachers reported that they, along with their students, increased their understanding and improved their perceptions of computer science (Goldberg et al., 2012).

Researchers have also explicitly sought to understand the transfer of computer science learning to other subjects, especially mathematics, with mixed results (for reviews, see Palumbo, 1990; Simon et al., 2006). Others have focused on using computing as a medium for learning in other subject areas and the conditions that must be met to facilitate such learning. For example, Kafai, Franke, Ching, and Shih (1998) showed that programming could be used as a medium for elementary students to express fractions, while Schanzer, Fisler, Krishnamurthi, and Felleisen (2015) found positive algebra achievement outcomes for students who engaged a curriculum that taught algebra through computer programming. Additionally, Grover, Pea, and Cooper (2016) showed that among middle school students engaged in an introductory computer science curriculum focused on programming and algebraic thinking, prior math and computing experience, as well as English language proficiency, were critical predictors of algebraic thinking outcomes and conceptions of computer science. Lewis and Shah's (2012) study of elementary students showed similar results; students with higher incoming math scores did better on programming assessments. In a subsequent study, Lewis (2014) suggests that middle school students' understanding of algebraic substitution can transfer to computer science, noting difficulties that students have with such knowledge transfer.

Given the mixed findings on learning transfer as well as the lack of research on transfer to non-STEM domains, more research is needed to provide a solid understanding of whether, how, and in what context learning computer science can transfer to different subjects and vice versa. Foundational research can help identify complementary content in noncomputer science subjects and develop and test different models of content integration. This line of work can include identifying best practices and effective teaching strategies that lead to student learning across subject areas, including specific PCK relevant to the integration of multiple subject areas. As many noncomputer science teachers may be teaching computer science concepts in their classrooms, researchers can also explore what training and support is necessary to ensure that these teachers develop not only effective teaching strategies to support student learning but also positive attitudes and self-efficacy in teaching computer science concepts.

In addition to exploring the more obvious connections between computer science learning and other STEM content areas, future research can investigate student achievement and attitudinal outcomes in non-STEM areas, such as literacy, art, and social studies, after receiving computer science instruction embedded in the given content area. Further, such research can approach these topics with varying degrees of granularity. For example, at a more granular level, studies could investigate whether learning debugging in a computer science context transfers to problem solving in mathematics, while at a broader level, studies could explore the effects of an entire integrated computer science curriculum on student learning across all subject areas. Researchers can extend this work to develop a more nuanced understanding of the conditions that help support such knowledge transfer, including teacher practices, prior experience, and differences of transfer for different subject domains.

Further, such research can be enhanced through collaborations between computer science researchers and those from other content areas to explore similarities and differences among PCK and learning progressions in different domains. Even the National Science Foundation (NSF) has recognized the importance of such collaborations in its STEM+Computing Partnerships solicitation, which requires proposed projects to integrate computing into STEM education. Currently funded projects hold promise for advancing the field as they investigate topics ranging from integrating computer science into elementary and secondary science and math classrooms to the development of novel interdisciplinary computer science curricula to a focus on professional development for teachers and school administrators for scaling such initiatives. Future studies can build upon findings from these projects and continue the collaborative efforts under way in an effort to advance not only computer science education but also teaching and learning across the K–12 education sector more broadly.



Limitations

Despite continued progress in K–12 computer science education research, the field remains in need of rigorous studies to provide empirical evidence to address many of the unanswered questions described in the above research agenda. The evidence base on which the framework rests is incomplete, and limitations in current research are discussed below. These limitations are not all-encompassing, yet they can be viewed as actionable issues that offer additional insight for future studies.

First, and perhaps most pressing, much of the published research on U.S. computer science education is conducted with university students, many of whom have elected into undergraduate programs in computer science, with fewer studies targeting K–12 students. While the international research community provides some insight into the K–12 education context, international school systems vary dramatically from the U.S. system. As neither of these two contexts—domestic postsecondary or international K–12—are easily transferred to the U.S. K–12 environment, there remains a limited research base related to K–12 computer science classrooms in the United States (Hubwieser et al., 2011). The K–12 Computer Science Framework promotes a renewed focus on computer science education research and calls specifically for further studies to validate the proposed learning progressions and to inform future revisions to the framework.

Second, the research that does exist in the K–12 setting is often limited in sample and scope, making it difficult to generalize findings across different grade levels and student populations as well as computer science concepts and curricula. More recent initiatives have started tackling this limitation by adapting mainstream curricular materials for students with learning differences (e.g., Israel et al., 2015; Wille, Pike, & Century, 2015); integrating computer science into elementary literacy into a large, diverse school district (Milenkovic, Acquavita, & Kim, 2015); evaluating activities aimed at helping diverse learners with varying levels of mathematics and English language preparation in large urban school districts (Grover, Jackiw, & Lundh, 2015); developing learning progressions for the K–8 level (Isaacs et al., 2016); and conducting large-scale implementation evaluations across the entire K–12 learning
environment (Mark & DeLyser, 2016). While more research is certainly needed, the ideas and recommendations that are produced from these early initiatives may inform future revisions to the framework and its byproducts, such as standards, curriculum, and professional development.

Finally, as with any applied research setting, methodological limitations pose threats to the validity of research findings; this issue may be even more pertinent in K–12 computer science education because of the novelty of conducting research at this level and thus the lack of pre-established valid and reliable measures to do so. Although there is ongoing work to develop sound instruments (e.g., Wille & Kim, 2015), the variety of curricular materials and implementation approaches in computer science education efforts calls for more methodological research studies to understand best practices for measuring what K–12 computer science education looks like at the classroom level, the factors that affect implementation, and ultimately student and teacher attitudinal and learning outcomes.

Policy and Implementation Considerations

The recent influx of attention from federal, state, and local education agencies has brought K–12 computer science education into the national spotlight, and at all levels of government, new and recommended policies will continue to be advanced. These policies include counting computer science toward a graduation requirement and expecting all schools within a district to engage all students in at least one computer science experience at all levels of learning (i.e., elementary, middle, and high school). At the same time, how local entities enact such policies and what these policies look like in the classroom are yet to be clearly understood. Implementation of any education innovation is sure to be complex, with many moving parts and factors that affect both the fidelity of implementation and subsequent teacher and student outcomes.

An extensive review of education innovation implementation research by Century and Cassata (in press) provides contextual framing for understanding the complexities of computer science implementation in the K–12 education environment. The authors suggest five facets of implementation research:

1. informing the design and development of innovations;
2. evaluating whether and to what extent an innovation achieves its desired outcomes;
3. understanding why an innovation works, for whom, and in what contexts;
4. improving innovation design and use in applied settings; and
5. informing theory development (Century & Cassata, in press).

These five categories move beyond fidelity of implementation to understand the what, why, how, and for whom of education innovations in an effort to capture all facets that influence the outcomes of such initiatives. At a more granular level, implementation research can work to measure both structural factors (e.g., frequency and duration; content covered and omitted; modification and supplementation) and interaction factors (e.g., teacher facilitation of pedagogical practices, attitudes toward the innovation, teacher and student interest and self-efficacy in the content domain) (Century, Cassata, Rudnick, & Freeman, 2012).

Based on these broader conceptions of education innovation implementation research, there are several considerations regarding policy and curricular innovations in K–12 computer science education. These considerations include understanding and measuring curriculum and assessment; course and instructional pathways; teacher professional development; and community and industry stakeholder involvement. For example, in addition to using traditional achievement assessments to measure computer science learning (e.g., multiple-choice and short-answer tests), more "nontraditional" measures could be developed, such as performance tasks and student work portfolios, that capture the richness and creativity of the computing environment otherwise missed on most standard assessments. Similarly, various course models are sure to be explored (e.g., integrated courses, standalone courses, pathways of courses, and career and technical education), such that an important component to understanding implementation will be exploring the most effective models for different grade levels, student populations, and learning goals. Part of this work necessarily includes understanding the best practices for ensuring that teachers have sufficient training and support both prior to and during their time in the classroom, as well as establishing more consistent certification opportunities and requirements. Additionally, given the focus on encouraging students to persist in the computing field over the long term, community and industry partnerships are also likely to play a role in how schools and districts engage their students in computer science.

To address these implementation components, school and district leaders will also require additional knowledge, resources, and support in their effort to provide high-quality computer science instruction and professional development. Indeed, a major factor in education innovation implementation is leadership support and the value placed on enacting the innovation (Century et al., 2012; Century & Cassata, in press). One initiative, LeadCS.org, offers researcher-developed tools and anecdotes from teacher, school, district, and partner leaders to help education leaders bring or enhance existing computer science initiatives to their own contexts. The website also provides links to additional resources related to infrastructure, instruction, and implementation.

Looking Toward the Future

The K–12 Computer Science Framework is informed by research that is presently available, but there is still a long way to go. A review of the current research found areas of alignment with the framework's core concepts, practices, and statements, resulting in a more compelling and robust framework. However, this research review also identified gaps in the field, which were gathered to create a research agenda moving forward. This agenda creates an opportunity for current and future computer science education researchers to explore meaningful research questions to advance the field and support the framework.

LeadCS.org offers researcher-developed tools to help education leaders with computer science initiatives.

As computer science education spreads, particularly with the use of this framework, more questions will need to be asked and more studies completed to suggest answers. The community of stakeholders in computer science education, including researchers, teachers, administrators, and policymakers, must come together to advance the research that will underpin the future of K–12 computer science education.

References

Achieve. (2015). *The role of learning progressions in competency-based pathways.* Retrieved from http://www.achieve.org/files/Achieve-LearningProgressionsinCBP.pdf

Aivaloglou, E., & Hermans, F. (2016, September). How kids code and how we know: An exploratory study on the Scratch repository. In *Proceedings of the Twelfth Annual International Conference on International Computing Education Research* (pp. 53–61).

Bender, E., Hubwieser, P., Schaper, N., Margaritis, M., Berges, M., Ohrndorf, L., . . . Schubert, S. (2015). Towards a competency model for teaching computer science. *Peabody Journal of Education, 90*(4), 519–532. doi: 10.1080/0161956X.2015.1068082

Bienkowski, M., Snow, E., Rutstein, D. W., & Grover, S. (2015). *Assessment design patterns for computational thinking practices in secondary computer science: A first look* (SRI technical report). Menlo Park, CA: SRI International. Retrieved from http://pact.sri.com/resources.html

Blickenstaff, J. C. (2005). Women and science careers: Leaky pipeline or gender filter? *Gender and Education, 17*(4), 369–386. doi: 10.1080/09540250500145072

Brennan, K. & Resnick, M. (2012). *Using artifact-based interviews to study the development of computational thinking in interactive media design*. Paper presented at the annual meeting of the American Educational Research Association, Vancouver, BC, Canada.

Buffardi, K., & Edwards, S. H. (2013, August). Effective and ineffective software testing behaviors by novice programmers. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research* (pp. 83–90).

Bureau of Labor Statistics. (2015). *Occupational employment statistics* [Data file]. Retrieved from http://www.bls.gov/oes

Century, J., & Cassata, A. (in press). Measuring implementation and implementation research: Finding common ground on what, how and why. *Review of Research in Education, Centennial Edition*. American Educational Research Association.

Century, J., Cassata, A., Rudnick, M., & Freeman, C. (2012). Measuring enactment of innovations and the factors that affect implementation and sustainability: Moving toward common language and shared conceptual understanding. *Journal of Behavioral Health Services & Research*, *39*(4), 343–361.

Cernavskis, A. (2015, June 25). In San Francisco, computer science for all . . . soon. *The Hechinger Report.* Retrieved from http://hechingerreport.org/san-francisco-plans-to-be-first-large-district-to-bring-computer-science-to-all-grades/

Computer Science Teachers Association Teacher Certification Task Force. (2008). *Ensuring exemplary teaching in an essential discipline: Addressing the crisis in computer science teacher certification.* New York, NY: Computer Science Teachers Association and the Association for Computing Machinery.

Cooper, S., Grover, S., Guzdial, M., & Simon, B. (2014). A future for computing education. *Communications of the ACM, 57*(11), 34–46.

Dasgupta, S., Hale, W., Monroy-Hernández, A., & Hill, B. M. (2016, February). Remixing as a pathway to computational thinking. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing* (pp. 1438–1449).

De Boulay, B. (1989). Some difficulties of learning to program. In E. Soloway and J. Spohrer (Eds.), *Studying the novice programmer*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Denner, J., Werner, L., Campe, S., & Ortiz, E. (2014). Pair programming: Under what conditions is it advantageous for middle school students? *Journal of Research on Technology in Education*, *46*(3), 277–296.

Dwyer, H., Hill, C., Carpenter, S., Harlow, D., & Franklin, D. (2014, March). Identifying elementary students' pre-instructional ability to develop algorithms and step-by-step instructions. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (pp. 511–516).

Evers, V., & Day, D. (1997). The role of culture in interface acceptance. In S. Howard, J. Hammond, & G. Lindgaard (Eds.), *Proceedings of Human-Computer Interaction INTERACT'97* (pp. 260–267). Sydney, Australia: Chapman & Hall.

Franklin, D. F. (2015, February). Putting the computer science in computing education research. *Communications of the ACM, 58*(2), 34–36.

Goldberg, D. S., Grunwald, D., Lewis, C., Feld, J. A., & Hug, S. (2012). Engaging computer science in traditional education: The ECSITE project. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education* (pp. 351–356).

Goldschmidt, D., MacDonald, I., O'Rourke, J., & Milonovich, B. (2011). An interdisciplinary approach to injecting computer science into the K–12 classroom. *Journal of Computing Science in College, (26)*6, 78–85.

Good, C., Rattan, A., & Dweck, C. S. (2012). Why do women opt out? Sense of belonging and women's representation in mathematics. *Journal of Personality and Social Psychology*, *102*(4), 700–717.

Goode, J., Estrella, R., & Margolis, J. (2006). Lost in translation: Gender and high school computer science. In W. Aspray & J. M. Cohoon (Eds.), *Women and information technology: Research on underrepresentation* (pp. 89–113). Cambridge, MA: MIT Press.

Google & Gallup. (2015a). *Images of computer science: Perceptions among students, parents, and educators in the U.S.* Retrieved from http://g.co/cseduresearch

Google & Gallup. (2015b). *Searching for computer science: Access and barriers in U.S. K–12 education*. Retrieved from http://g.co/cseduresearch

Grover, S., Jackiw, N., & Lundh, P. (2015). Thinking outside the box: Integrating dynamic mathematics to advance computational thinking for diverse student populations. Abstract. Retrieved from http://www.nsf.gov/awardsearch/showAward?AWD_ID=1543062

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher, 41*(1), 38–43.

Grover, S., Pea, R., & Cooper, S. (2016, March). Factors influencing computer science learning in middle school. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (pp. 552–557), Memphis, TN.

Guernsey, L. (2012). *Screen time: How electronic media from baby videos to educational software affects your young child.* Philadelphia, PA: Basic Books.

Guzdial, M. (2013, August). Exploring hypotheses about media computation. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research* (pp. 19–26).

Guzdial, M. (2016). *Learner-centered design of computing education: Research on computing for everyone*. Synthesis lectures on human-centered informatics. Morgan and Claypool.

Hanks, B. (2008). Problems encountered by novice pair programmers. *Journal on Educational Resources in Computing (JERIC)*, *7*(4), Article 2.

Hansen, A., Dwyer, H., Hill, C., Iveland, A., Martinez, T., Harlow, D., & Franklin, D. (2015). Interactive design by children: A construct map for programming. In *Proceedings of the 14th International Conference on Interaction Design and Children* (pp. 267–270).

Hansen, A., Hansen, E., Dwyer, H., Harlow, D., & Franklin, D. (2016). Differentiating for diversity: Using universal design for learning in computer science education. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (pp. 376–381).

Hansen, A., Iveland, A., Carlin, C., Harlow, D., & Franklin, D. (2016, June). User-centered design in block-based programming: Developmental and pedagogical considerations for children. In *Proceedings of the 15th International Conference on Interaction Design and Children* (pp. 147–156).

Hubwieser, P., Armoni, M., Brinda, T., Dagiene, V., Diethelm, I., Giannakos, M. N., . . . Schubert, S. (2011, June). Computer science/informatics in secondary education. In *Proceedings of the 16th Annual Conference Reports on Innovation and Technology in Computer Science Education—Working Group Reports* (pp. 19–38).

Hubwieser, P., Magenheim, J., Mühling, A., & Ruf, A. (2013, August). Towards a conceptualization of pedagogical content knowledge for computer science. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research* (pp. 1–8).

Isaacs, A., Binkowski, T. A., Franklin, D., Rich, K., Strickland, C., Moran, C., . . . Maa, W. (2016). Learning trajectories for integrating K–5 computer science and mathematics. In *2016 CISE/EHR Principal Investigator & Community Meeting for CS in STEM Project Description Booklet* (p. 79). Retrieved from https://www.ncwit.org/sites/default/files/file_type/pi_book_compressed_2016.pdf

Israel, M., Wherfel, Q., Pearson, J., Shehab, S., & Tapia, T. (2015). Empowering K–12 students with disabilities to learn computational thinking and computer programming. *TEACHING Exceptional Children, 48*(1), 45–53.

Kafai, Y. B., Franke, M. L., Ching, C. C., & Shih, J. C. (1998). Game design as an interactive learning environment for fostering students' and teachers' mathematical inquiry. *International Journal of Computers for Mathematical Learning, 3*(2), 149–184. doi: 10.1023/A:1009777905226

Kafai, Y. B., & Burke, Q. (2015). Constructionist gaming: Understanding the benefits of making games for learning. *Educational Psychologist, 50*(4), 313–334.

Koppelman, H. (2007). Exercises as a tool for sharing pedagogical knowledge. In *Proceedings of the 12th Annual SIGCSE Conference in Innovation and Technology in Computer Science Education* (p. 361). doi: 10.1145/1268784.1268933

Koppelman, H. (2008, June). Pedagogical content knowledge and educational cases in computer science: An exploration. In *Proceedings of the Informing Science and IT Education Conference (InSITE)* (pp. 125–133), Varna, Bulgaria.

Kothiyal, A., Majumdar, R., Murthy, S., & Iyer, S. (2013, August). Effect of think-pair-share in a large CS1 class: 83% sustained engagement. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research* (pp. 137–144).

Ladner, R., & Israel, M. (2016). "For all" in "computer science for all." *Communications of the ACM, 59*(9), 26–28.

LeBlanc, M. D., & Dyer, B. D. (2004). Bioinformatics and computing curricula 2001: Why computer science is well positioned in a post-genomic world. *ACM SIGCSE Bulletin*, *36*(4), 64–68.

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., . . . Werner, L. (2011). Computational thinking for youth in practice. *ACM Inroads, 2*(1), 32–37.

Leidner, D. E., & Kayworth, T. (2006). A review of culture in information systems research: Toward a theory of information technology culture conflict. *MIS Quarterly*, *30*(2), 357–399.

Lewis, C. M. (2014, July). Exploring variation in students' correct traces of linear recursion. In *Proceedings of the Tenth Annual Conference on International Computing Education Research* (pp. 67–74).

Lewis, C. M. (2016). You wouldn't know it from SIGCSE proceedings, but we don't only teach CS1 (Abstract Only). In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (p. 494).

Lewis, C. M., & Shah, N. (2012). Building upon and enriching grade four mathematics standards with programming curriculum. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (pp. 57–62).

Lewis, C. M., & Shah, N. (2015, July). How equity and inequity can emerge in pair programming. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research* (pp. 41–50).

Lishinski, A., Good, J., Sands, P., & Yadav, A. (2016, September). Methodological rigor and theoretical foundations of CS education research. In *Proceedings of the Twelfth Annual International Conference on International Computing Education Research* (pp. 161–169).

Lou, Y., Abrami, P. C., & d'Apollonia, S. (2001). Small group and individual learning with technology: A meta-analysis. *Review of Educational Research, 71*, 449–521.

Marcus, B. (2015, August 12). The lack of diversity in tech is a cultural issue. *Forbes.* Retrieved from http://www.forbes.com/sites/bonniemarcus/2015/08/12/the-lack-of-diversity-in-tech-is-a-cultural-issue/#622c464a3577

Margolis, J., Estrella, R., Goode, J., Holme, J. J., & Nao, K. (2010). *Stuck in the shallow end: Education, race, and computing.* Cambridge, MA: MIT Press.

Margolis, J., Ryoo, J., Sandoval, C., Lee, C., Goode, J., & Chapman, G. (2012). Beyond access: Broadening participation in high school computer science. *ACM Inroads*, *3*(4), 72–78.

Mark, J., & DeLyser, L. (2016). CSNYC knowledge forum: Launching research and evaluation for computer science education, for every school and every student in New York City. Abstract. Retrieved from http://www.nsf.gov/awardsearch/showAward?AWD_ID=1637654

Milenkovic, L., Acquavita, T., & Kim, D. (2015). Investigating conceptual foundations for a transdisciplinary model integrating computer science into the elementary STEM curriculum. Abstract. Retrieved from http://www.nsf.gov/awardsearch/showAward?AWD_ID=1542842

Morrison, B. B., Margulieux, L. E., & Guzdial, M. (2015, July). Subgoals, context, and worked examples in learning computing problem solving. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research* (pp. 21–29).

National Governors Association Center for Best Practices & Council of Chief State School Officers. (2010). *Common core state standards*. Washington DC: Author.

National Research Council. (2007). *Taking science to school: Learning and teaching science in grades K–8.* Committee on Science Learning-Kindergarten Through Eighth Grade. R. A. Duschl, H. A. Schweingruber, & A. W. Shouse (Eds.). Board on Science Education, Center for Education. Division of Behavioral and Social Sciences and Education. Washington, DC: The National Academies Press.

National Research Council. (2012). *A framework for K–12 science education: Practices, crosscutting concepts, and core ideas.* Committee on a Conceptual Framework for New K–12 Science Education Standards. Board on Science Education, Division of Behavioral and Social Sciences and Education. Washington, DC: The National Academies Press.

National Science Foundation. (2014). *Women, minorities, and persons with disabilities in science and engineering, Table 5–1: Bachelor's degrees awarded, by sex and field: 2002–2012* [Data file]. Retrieved from https://www.nsf.gov/statistics/wmpd/2013/sex.cfm

Next Generation Science Standards Lead States. (2013). *Next generation science standards: For states, by state*s. Washington, DC: The National Academies Press.

Ohrndorf, L. (2015, July). Measuring knowledge of misconceptions in computer science education. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research* (pp. 269–270).

Palumbo, D. B. (1990). Programming language/problem-solving research: A review of relevant issues. *Review of Educational Research*, *60*(1), 65–89.

Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New Ideas in Psychology, 2*, 137–168.

Pea, R. D., Soloway, E., & Spohrer, J. C. (1987). The buggy path to the development of programming expertise. *Focus on Learning Problems in Mathematics, 9*, 5–30.

Perkins, D. N., & Salomon, G. (1988). Teaching for transfer. *Educational Leadership*, *46*(1), 22–32.

Saeli, M., Perrenet, J., Jochems, W. M., & Zwaneveld, B. (2011). Teaching programming in secondary school: A pedagogical content knowledge perspective. *Informatics in Education*, *10*(1), 73–88.

Saeli, M., Perrenet, J., Jochems, W. M., & Zwaneveld, B. (2012). Programming: Teachers and pedagogical content knowledge in the Netherlands. *Informatics in Education*, *11*(1), 81–114.

Schanzer, E., Fisler, K., Krishnamurthi, S., & Felleisen, M. (2015, February). Transferring skills at solving word problems from computing to algebra through Bootstrap. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (pp. 616–621).

Schulz, S., & Pinkwart, N. (2015, November). Physical computing in STEM education. In *Proceedings of the Workshop in Primary and Secondary Computing Education* (pp. 134–135).

Seiter, L., & Foreman, B. (2013, August). Modeling the learning progressions of computational thinking of primary grade students. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research* (pp. 59–66).

Shulman, L. S. (1986). Those who understand: Knowledge growth in teaching. *Educational Researcher 15*(2), 4–14. doi: 10.3102/0013189X015002004

Simon, Fincher, S., Robins, A., Baker, B., Box, I., Cutts, Q., . . . Tutty, J. (2006, January). Predictors of success in a first programming course. In *Proceedings of the Eighth Australasian Computing Education Conference* (pp. 189–196), Hobart, Tasmania, Australia.

Smith, J. L., Lewis, K. L., Hawthorne, L., & Hodges, S. D. (2013). When trying hard isn't natural: Women's belonging with and motivation for male-dominated STEM fields as a function of effort expenditure concerns. *Personality and Social Psychology Bulletin*, *39*(2), 131–143.

Soloway, E. (1986). Learning to program = learning to construct mechanisms and explanations. *Communications of the ACM, 29*(9), 850–858.

Sprague, P., & Schahczenski, C. (2002). Abstraction the key to CS1. *Journal of Computing Sciences in Colleges, 17*(3), 211–218.

Sullivan, G. (2014, May 29). Google statistics show Silicon Valley has a diversity problem. *The Washington Post.* Retrieved from https://www.washingtonpost.com/news/morning-mix/wp/2014/05/29/most-google-employees-are-white-men-where-are-allthewomen/

Valkenburg, P. M., & Peter, J. (2013). The differential susceptibility to media effects model. *Journal of Communication*, *63*, 221–243. doi: 10.1111/jcom.12024

Watson, W. E., Kumar, K., & Michaelsen, L. K. (1993). Cultural diversity's impact on interaction process and performance: Comparing homogeneous and diverse task groups. *Academy of Management Journal, 36*(3), 590–602.

Wehmeyer, M. L. (2015). Framing the future self-determination. *Remedial and Special Education*, *36*(1), 20–23.

Wehmeyer, M. L., Shogren, K. A., Palmer, S. B., Williams-Diehm, K. L., Little, T. D., & Boulton, A. (2012). The impact of the self-determined learning model of instruction on student self-determination. *Exceptional Children, 78*(2), 135–153.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2015). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology, 25*(1), 127–147.

Wille, S. J., & Kim, D. (2015). Factors affecting high school student engagement in introductory computer science classes. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (pp. 675–675), New York, NY. doi: 10.1145/2676723.2691891

Wille, S. J., Pike, M., & Century, J. (2015). Bringing AP Computer Science Principles to students with learning disabilities and/or an ADHD: The hidden underrepresented group. Abstract. http://www.nsf.gov/awardsearch/showAward?AWD_ID=1542963

Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., & Korb, J. T. (2014). Computational thinking in elementary and secondary teacher education. *ACM Transactions on Computing Education, 14*(1), Article 5, 1–16.

Appendices

Appendix A: Feedback and Revisions

Public review and feedback were essential for the development of the framework. The framework underwent three public review periods. The first review period included a draft of the 9–12 grade band concept statements and the practice overviews. The second and third included full drafts of concept statements, overviews, and practices and included a glossary and a preface that described the framework and provided a brief overview of the important chapters in the framework. The other chapters intended for inclusion in this document were not included in this public review process.

Each public review period lasted two to three weeks. The review periods were announced widely: each organization on the steering committee sent out an email or posted an announcement on its blog; each state participating in the development was asked to hold a focus group; each advisor was asked to review the framework; emails were sent to dozens of organizations in computer science and education; and the framework was emailed to the hundreds of members of the K–12 Computer Science Framework email list.

The questions in the online review form differed slightly for each review, as the type of feedback the writers needed changed as the statements developed. All three review forms asked reviewers to rate their overall impression of the draft (choosing from excellent, very good, good, fair, and poor). On each review form, reviewers were asked to rate each concept statement and practice on two criteria: whether it was clear and understandable to a computer science novice and whether it was important for all students to know (the phrasing of the questions varied slightly between review periods). Open comment boxes were also provided throughout the form for the reviewers to highlight strengths and give suggestions for improvement. Additionally, the second review form asked reviewers to rate each practice progression and concept statement on whether it was developmentally appropriate for the grade band. The third review form asked reviewers to rate each concept overview, subconcept overview, practice overview, and practice progression on whether the level of detail was appropriate for understanding the overview or progression.

Throughout the writing process, the writing teams reviewed the concept statements using a list of criteria. The second public review asked reviewers to rate the framework using the same list:

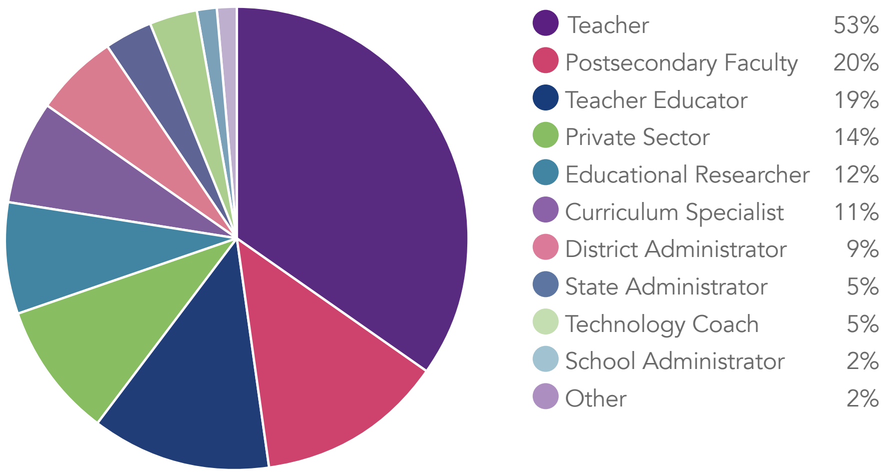
- **Essential:** Is this a core idea of computer science? Is it important and essential for all students? How does it make for a computationally literate person? What benefit does it have for the person and society?
- **Powerful in Application:** Is knowing the concept or performing the practice useful? Is it useful for solving problems, useful for illuminating other ideas downstream, and helpful for understanding a larger body of knowledge? Does it elicit extensions, foster interdisciplinary connections, and show potential for a wide range of applications?

- **Relevant and Clear:** Is the statement understandable by teachers and relevant to students? Will computer science novices feel the concept statement is approachable/inviting?
- **Diverse:** How well do the framework statements describe a diverse, equitable, and accessible vision of computer science?
- **Research-Informed:** Is the statement informed by research? How can the statement be revised to reflect computer science education research? Does the statement point to possible areas of research?
- **Developmentally Appropriate:** Is it developmentally appropriate and suitable for high school?
- **Interdisciplinary:** Is the framework statement useful and applicable outside of the domain of computer science? Are there opportunities to make interdisciplinary connections? Does it complement concepts and practices in math, science, etc.?
- **Promoting College and Career Readiness:** How well do the concepts and practices contribute to career and college readiness?

Overall, 152 individuals submitted reviews (40 of whom also participated in a group review), and more than 306 individuals participated in a group review. More than 50 group reviews were held over the three review periods. These reviews included representation from 38 U.S. states, 1 U.S. territory, and 7 international locations. Figure A.1 shows the general breakdown of occupations reported by reviewers.

At the end of each review period, the development team reviewed all the input, identified major themes, analyzed the ratings, and provided recommendations to the writing team based on the data. The writing team was provided access to the themes, recommendations, and all of the anonymized raw comments and survey data.

Figure A.1: Occupations of reviewers



The next three sections describe the overall feedback themes that applied to the overall framework, major themes that applied to core concepts, and major themes for the practices. Although the writers examined all feedback, they were given license to decide how the feedback should be addressed. They applied the data they were presented from the various reviews, along with their professional judgment, to determine their revisions. In some cases, the writers decided not to address, or only minorly address, the suggested feedback. Those decisions and rationale are included below.

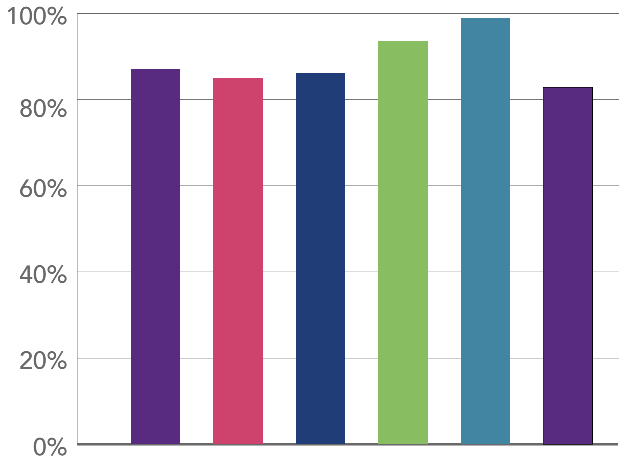
Overall Feedback Themes

Positive Feedback

In addition to providing constructive feedback, reviewers were asked to comment on the strengths of the framework. The writers used positive feedback as a guide for helping them better understand what was ideal in terms of structure, grain size, voice, and general writing style. Themes emerged from the comments that pointed to the framework being well-developed, comprehensive, and a good progression for K–12 learning. In review periods two and three, people wrote about how much the framework had improved since the first review period, that added examples made the content easier to understand, and that changes in language had made the document easier to understand. Reviewers also felt that the framework covers the right concepts and practices and appreciated that it is inclusive of much more than just "coding." Other reviewers wrote about how much a document like this is needed in computer science education and were excited about the effort overall. High percentages of reviewers also indicated that they believed the framework would be useful in a variety of ways (see Figure A.2).

Figure A.2: Survey responses on the importance of the framework

PERCENTAGE OF REVIEWERS WHO BELIEVE THE FRAMEWORK IS USEFUL FOR...



●	Classroom Instruction	88%
●	Advocacy for Computer Science	86%
●	Teacher Training	87%
●	Curriculum Development	94%
●	Standards Development	99%
●	Research into Computer Science Education	84%

During review period three, reviewers were also asked to report on the overall usefulness of the framework in terms of what it might inform. Reviewers were overwhelmingly positive about the different potential uses of the framework.

Additionally, when asked about their overall impression of the framework, 92% of reviewers rated it as excellent, very good, or good, showing that it was a highly favored document.

While there was positive feedback on the document throughout the process, reviewers also offered extensive constructive feedback in the interest of continually improving upon the framework with every revision cycle.

Constructive Feedback Overview

Reviewers also indicated several areas of concern that applied to both concepts and practices. These areas of concern fell into three categories: mechanics and language, content, and usage.

- Mechanics and Language
  - Language too technical
  - Voice inconsistent
  - Content unclear
- Content
  - Content too broad
  - Content overlaps
  - Content missing
  - Content not essential for all learners
  - Renaming the core concepts and practices
- Usage
  - Audience unclear
  - Integration with and alignment to other disciplines not sufficient

The following sections describe each of these areas of concern and the writing team's response.

Language Too Technical

In all three review periods, commenters critiqued various terms used in the framework. Many reviewers wrote that the vocabulary was too technical and confusing, particularly for someone who is a novice to computer science. Acronyms were occasionally used that were unknown to some reviewers. There was also some confusion expressed about terms that have different meanings in computer science than in other fields. Because the framework was intended for novices to computer science, many reviewers questioned whether technical terminology was appropriate. Some reviewers felt that the language was uninviting, and reviewers frequently asked for terms to be defined.

On the other hand, some reviewers wanted to see computer science terms in the framework. Some commenters with knowledge of computer science questioned why the writers seemed to be using other words to avoid using a term, rather than simply using the term itself.

Response

The writers attempted to find a balance between using terms that are essential for understanding computer science and removing all jargon, acronyms, or overly technical or advanced terms. The writing team reviewed each technical term and removed any that were not necessary for understanding the framework. Terms that were deemed essential to computer science were used in the framework and were defined in a a **Glossary** (see **Appendix C**). The draft glossary was provided during the second and third review periods.

The writers also considered reading level, and the development team tested the reading level of the statements after the second and third revisions. The writers and an editor removed potentially confusing language and used simple words whenever possible. Additionally, some of the more complex computer science terminology was described or explained in a concept or practice statement.

Voice Inconsistent

In all three review periods, reviewers commented on the lack of a consistent voice for the framework statements. The feedback ranged from very general (e.g., the overviews are written differently, some concept statements read very differently from other concept statements) to very specific about language choice (e.g., the subject of the statements as "groups," "students," or "people"; using "computers" versus "computing devices"). One area of concern about the practice statements was that some statements sounded like end of 12th grade goals but others sounded like general goal statements that could apply across K–12.

Response

Throughout the process, the development team created and refined guidelines around writing style and structure. An editor was brought onto the writing team to create a style guide for consistent language choice and sentence structure. The editor also edited all concepts and practices prior to each of the following review periods and prior to the final release.

Small writing teams addressed issues of consistent voice in concepts, practices, and overviews. For each (concepts, practices, and overviews), a team of three writers and a member of the development team made the language style and format consistent.

The practices writing team clarified that all practice statements should reflect goals "by the end of Grade 12," and the team revised all of the statements to reflect this endpoint. The practices team also revised the overviews to align in content and verb usage with the statements and progressions.

Content Unclear

Many comments from the reviews addressed clarity in the concept statements and practices. The reviewers wanted more examples or explanation to make the statements clear. Some comments conveyed that the reviewers were unsure about the intent of a statement, or they misinterpreted the intent. Most of these comments were made about the statements in the K–2 and 3–5 grade bands. Reviewers of the practices appreciated the examples, saying that the examples bring the practices to life. Other practices were less clear, and reviewers asked for more examples.

Response

The writing team added depth to the statements and practices. Some drafts of the framework had statements that were so concise that the intent was no longer clear. The writers added more description so that the topic of the statement was clear.

The K–5 concept statements in particular, while originally written to be approachable, received feedback that students already know or are already learning these ideas. The writing team used this feedback, along with the Lexile reading level scores for these statements, to expand on the computer science ideas.

The practices writing team made many edits for clarification. For example, in response to feedback about the complexity and lack of clarity around abstraction, the writers attempted to strike a balance between the needs of a novice reader and the needs of an expert reader. The writers clarified the definition of abstraction and extended the examples to contextualize the practice of abstraction.

The concept and practices writing teams added examples and clarification into the descriptive material for each concept and into the progressions for the practice statements. They also chose to include only the most salient and specific examples and to describe how each example reflected the concept or practice statement.

Content Too Broad

Reviewers noted that several subconcepts and concept statements were very broad and included many different ideas. Reviewers who had experience writing standards were concerned that the concept and practice statements would not translate well to standards. Other comments said that the phrasing was confusing and that more clarity was needed to understand the intent of each statement.

Response

The writing team narrowed the focus of each concept and practice statement and prioritized the key topics and ideas. Each statement was revised to focus on simpler and fewer ideas. For example, in the

Algorithms and Programming core concept, program development originally contained many different topics but eventually became focused on the iterative process of designing, implementing, and reviewing programs, including taking into account diverse users and diverse teams. Another example is in the *Computing Systems* core concept, in which a subconcept focused on system software or operating systems was combined with the hardware and software subconcept.

Content Overlaps

Some reviewers identified areas of overlap between subconcepts or between practice statements. For example, many reviewers identified areas of overlap between the practices *Fostering an Inclusive Computing Culture* and *Collaborating Around Computing* or between the practice *Creating Computational Artifacts* and multiple other practices.

Response

The writing team carefully reviewed the specific feedback about areas of overlap and attempted to minimize this overlap.

Some ideas were removed entirely from one core concept or practice and left in another. When a topic was left in multiple sections, an effort was made to ensure that only the specific aspect of the topic addressed in the concept or practice was incorporated. For example, in response to feedback about data representation appearing in both the *Data and Analysis* and the *Algorithms and Programming* core concepts, the overlapping ideas were retained in *Data and Analysis* concept statements, and the *Algorithms and Programming* statements were reframed to focus on variables and data types.

The practices team used the specific reviewer feedback to more clearly distinguish between the practices. For example, phrasing that focused on collaborating in teams was removed from *Fostering an Inclusive Computing Culture* to more clearly delineate its boundary with *Collaborating Around Computing*.

On the other hand, some of the overlap was deemed necessary by the writing team. For example, some topics that were deemed essential to multiple concepts, like human–computer interaction and privacy and security, became crosscutting concepts. For the practices, some terms and ideas from *Fostering an Inclusive Computing Culture* were intentionally left in other practices; this overlap was deliberate to emphasize the importance of diversity throughout the practices. Similarly, the writing team received conflicting feedback about *Creating Computational Artifacts* and the other practices: reviewers thought this practice was essential but were concerned about the overlap with the other practices. The writers attempted to minimize the overlap by focusing this practice on the purpose of modifying an artifact.

Content Missing

Some reviewers suggested content that was not yet included in the framework concepts and practices.

Response

The framework writers responded to these suggestions in several different ways.

In some cases, when content did not appear in the statements yet, the writers agreed with the reviewers. For example, based on feedback to include cybersecurity, the writing team enlisted advice from the National Initiative for Cybersecurity Education and the Cyber Innovation Center about the core ideas of cybersecurity education and created a cybersecurity subconcept. Other examples include cyberbullying, computer science components of digital citizenship, careers in computer science, and bits. In other cases, reviewers wanted to see a greater focus on specific topics. For example, based on feedback, the writers broadened the focus of systems in the *Computing Systems* core concept and decided to include modeling in the practice *Developing and Using Abstractions* to convey how to interact with models and simulations and how to be able to contribute to part of the process of making one.

Another suggestion from reviewers was ensuring that the statement did not restrict or limit the content that could be taught. For example, some reviewers believed that the references to object-oriented programming and order of presentation in *Algorithms and Programming* statements removed functional programming as a possible paradigm at the middle school and high school levels. The writers revised the statements to allow functional programming to be used. In the descriptive material, the writers provided examples that were described as "possible" ways to do things rather than the expected ways to do something.

Other suggestions were considered, but ultimately the writers decided that the suggested content went beyond the computer science that all learners should know. For example, reviewers suggested including in the practice statements ideas such as intelligent machines and computer graphics, analyzing artifacts made by others, and contributing back to code communities. The writing team determined either that not all students should be required to do these or that some of the ideas were outside the scope of the practices and would necessitate overly technical language.

Content Not Essential for All Learners

Some reviews flagged particular concept or practice statements as not being essential for all students to learn or not being core and central to computer science. Other reviews flagged particular statements as being too advanced for a high school class.

Response

Over the course of the three drafts, the number of concept statements was drastically reduced. Writers with expertise in each grade band reviewed all of the concept statements for that grade band for whether each statement was (1) essential for all students and (2) developmentally appropriate for the given grade band.

The writing team reconsidered all statements, with a focus on those flagged by the reviewers. Because the framework describes concepts and practices for all students, the writers removed content that would be taught in a specialized high school course. The writers also questioned whether each statement was of equal importance and deleted parts of statements that were deemed nonessential for all students. For example, the writers removed ideas in the *Algorithms and Programming* core concept that reviewers said were not necessary for all students to learn in high school, such as recursion and object-oriented programming, and made these ideas optional. In the practices, the writers carefully considered the extent to which modeling was included.

Renaming the Core Concepts and Practices

Some reviewers were concerned about the message sent by the names of the core concepts, subconcepts, and practices. They agreed with the division of the five core concepts, for example, but were critical of the naming as being too technical or not representing the power of computer science.

Response

The writers consulted with the advisors and referred to specific feedback to modify the names of the core concepts, practices, and subconcepts. The writers made thoughtful and precise edits to these names for clarity and to emphasize the powerful computer science ideas that each encompassed. For example, Networks and Communication became *Networks and the Internet,* Data and Information became *Data and Analysis,* and Testing and Iteratively Refining *became Testing and Refining Computational Artifacts.* For subconcepts, an illustrative example is in the *Impacts of Computing* core concept. After early feedback about the subconcepts, the writing team restructured the subconcepts into more natural groupings and changed names to be more reader-friendly and clear.

Audience Unclear

Several commenters in the first review period wanted clarification about the audience for the framework. Other commenters reviewed the framework under an incorrect assumption about the audience, such as thinking that the framework was written for students. Reviewers of the practices, on the other hand, commented that computer scientists would not engage in some of the practices.

Response

The intended audience for the framework was added to the preface for the following drafts and included in the introductory chapters of the final draft (see **A Vision for K–12 Computer Science** for a full description of the intended audience). The writers also kept the audience in mind while they wrote, with the understanding that some readers may not have a background in computer science, which influenced language revisions.

The practices writing team clarified the audience and intent of the practices in the Preface of the **Practices** chapter Although the practices are written for a professional adult audience, the practices themselves are aimed at all students. Thus, although a computer scientist may not engage in the practice, all students, regardless of future profession, should be able to engage in the practices.

Integration with and Alignment to Other Disciplines Not Sufficient

In all three review periods, reviewers wanted to see explicit connections to other disciplines. Comments on the practices indicated that reviewers wanted to see more connections with practices in science, engineering, literacy, and mathematics. On the other hand, reviewers also mentioned that some of the framework content was already taught in other disciplines.

Response

The writers attempted to include interdisciplinary connections for each concept statement or concept area but realized that this was beyond the scope of the framework and the expertise of the writers. Interdisciplinary connections and teaching computer science integrated into other content areas is broadly addressed in the **Implementation Guidance** chapter. Supplemental materials to the framework related to interdisciplinary connections could be released in the future or could be created at the level of standards or curriculum.

The writers also reviewed the statements that were tagged by comments as being similar to content taught in other subjects. Although overlap with other content was assumed by the writers to be desirable (and could aid in integration), the writers decided to focus each statement on the computer science content. The writers revised the statements to emphasize the computer science aspect of the statement. For example, the writing team modified the K–2 statements in the *Data and Analysis* core concept to focus on digital tool and automated collection, differentiating the content from data in mathematics.

The practices were revised to focus more specifically on the computer science context of each practice, rather than more general 21st century skills. However, there is some natural overlap in how the computational thinking practices align with engineering design practices or mathematical practices. The **Practices** chapter includes a diagram (Figure 5.2) showing some of these potential connections.

Major Themes for Concepts Only

Some of the feedback from reviewers applied only to the concepts. Major themes from these comments were

- crosscutting concepts needed to be revised, and
- subconcept progressions were not consistent.

Crosscutting Concepts Needed to be Revised

Feedback from the first review period requested the inclusion of crosscutting concepts. After this list was presented in the next review, feedback suggested that the list be revised.

Response

Although crosscutting concepts were a writing tool from the start of the project, the list was not presented during the first public review. After receiving feedback requesting crosscutting concepts, the draft list was shared during the next public review. From this draft list, however, some of the original crosscutting concepts eventually seemed to fit better in a single core concept, whereas some ideas originally placed in a single core concept were determined to be applicable to multiple core concepts, suggesting that they were crosscutting. For example, Human–Computer Interaction was originally a subconcept in *Impacts of Computing* but became a crosscutting concept due to overlap with other concept areas. Ethics and Security was originally a crosscutting concept but eventually ethics was incorporated into *Impacts of Computing*, and the crosscutting concept became Privacy and Security.

After the second review period, a small team of advisors and writers carefully read the draft statements, the feedback from the review, and relevant literature to identify the final list of crosscutting concepts. The full list is included in the preface of the **Concepts** chapter.

Subconcept Progressions Were Not Consistent

During the second and third review periods, reviewers commented on the "jumps"—or increases in sophistication and content—from one grade-band endpoint to the next within a progression. Some of the comments were that the jumps were too big, or too small, for the specified subconcept or that the jumps were inconsistently sized across the framework. Some reviewers wrote that the content in a particular grade-band is not essential as part of the progression and that this combined with a small jump did not warrant the inclusion of the concept statement.

Response

After the second and third review periods, the writers closely examined the progressions across the grade bands and wrote subconcept overviews to describe the progression. When necessary, the writers inserted language to ensure that there was a fluid transition between the different grade bands and that there was not too much added at each jump. For example, reviewers wanted to see programming as an expectation in the K–2 grade band and the removal of functional programming from the 3–5 grade band, changing these K–5 progressions in the *Algorithms and Programming* subconcepts. In some cases, writers considered starting some subconcept progressions at the 3–5 grade band, or ending a progression at the 6–8 grade band, but eventually all progressions spanned K–12 in the final draft.

Some subconcepts with smaller jumps were condensed or combined to produce fewer concepts overall. For example, three subconcepts within *Networks and the Internet* were combined into a single subconcept with greater jumps in the progression.

Major Themes for Practices Only

Some of the feedback from reviewers applied only to the practices. Major themes from these comments were

- computational thinking was not emphasized enough,
- there was confusion about practices written in progressions rather than grade bands, and
- practices were too narrowly focused on programming.

Computational Thinking Was Not Emphasized Enough

In the first review, many reviewers asked why computational thinking was not included. Computational thinking was not explicitly called out in the practices, and many reviewers believed that it was important to include the words.

Response

The practices writing team wrote a section of the **Practices** chapter that explained why computational thinking was not a practice. Instead, they included four practices that they specifically called out as computational thinking practices. Within these practices, ideas core to computational thinking were included, such as decomposition and abstraction. The writing team and development team frequently revisited the way that computational thinking was included in the practices.

There Was Confusion About Practices Written in Progressions Rather Than Grade Bands

During the second and third reviews, reviewers expressed confusion about the way that the practices were written differently than the concepts. During the second review period, reviewers were confused by the language of the progressions because they appeared to be written for grade bands. Reviewers were also concerned that the practices might be ignored if the reader could not easily apply the practices. Some reviewers wanted consistency between concepts and practices and expressed a preference for grade bands. On the other hand, many other reviewers, including some who had experience writing standards, were opposed to grade bands for the practices and encouraged the use of progressions.

Response

In attempting to make the practices as usable as possible, the writing team wrote the progressions in the format that could be most useful to standards writers and practitioners. The writing team took the position that the narrative structure provides flexibility and attempting to write the practices into grade bands would create artificial benchmarks. In response to critical feedback, the progressions were reworked to make them grade level independent and to alleviate confusion about the phrasing and language. Progressions included a starting point and an ending point but did not include grade bands. The writers also aligned each progression with the corresponding practice statement and chose specific verbs to better reflect what is actually done in the practice.

Practices Were Too Narrowly Focused on Programming

During the first review period, reviewers said that the practices were too narrowly focused on programming. Many of the examples included in the practices were specific to programming and did not help the reader make connections between the practices and other core concepts.

Response

The intent of the practices is that each practice could be combined with concept statements in all five core concepts, so the writing team attempted to make this clear with the language and examples in the practices. The writing team reworked examples in multiple practices to make sure that examples aided connection to core concepts and elements of computer science besides programming. In particular, *Communicating About Computing* and *Creating Computational Artifacts* were revised based on feedback to focus on multiple aspects of computer science.

Organizations That Convened Reviews

- AccessCS10K
- Achieve
- Association of State Supervisors of Mathematics
- California Department of Education
- Center for Applied Special Technology (CAST)
- Center for Elementary Math and Science Education, University of Chicago
- Change the Equation
- Chicago Public Schools, IL
- Code.org
- CodeCombat
- Codesters
- CodeVA
- Computing At School, United Kingdom
- Capital Region Academies for the Next Economy
- CS Teachers Ann Arbor Public Schools
- Computer Science Teachers Association, New Hampshire
- Computer Science Teachers Association, Delaware
- Computer Science Teachers Association, Minnesota
- CSNYC
- Cyber Innovation Center
- Deer Valley Unified School District, Computer Science Education Committee
- Expanding Computing Education Pathways, Maryland
- Gesellschaft für Informatik e.V., Germany
- Google Inc.
- Green Bay Area Public Schools, WI
- International Association of Privacy Professionals
- Indiana Department of Education
- Johnston County Schools, North Carolina
- Maine Mathematics and Science Alliance
- Maryland State Department of Education
- Massachusetts Computing Attainment Network
- Massachusetts Department of Education
- Microsoft Corporation
- Minnetonka Public Schools, MN
- Mississippi Department of Education
- Museum of Science, Boston
- New Jersey Department of Education
- National Initiative for Cybersecurity Education
- North Carolina Business and Industry—Programming and Engineers
- North Carolina Career and Technical Education Directors
- North Carolina Department of Public Instruction
- Nevada Computer Science Team
- New York City Department of Education, NY
- Ohio Review Team
- Oracle Academy
- Pacific Northwest National Laboratory (PNNL)
- Project Lead The Way
- San Francisco Unified School District (SFUSD), CA
- SRI, Center for Technology in Learning
- Washington Computer Science Learning Standards Advisory Committee
- Washington State Leadership and Assistance for Science Education Reform
- Washington State
- Washington STEM

Appendix B: Biographies of Writers and Development Staff

Writers

Julie Alano

Computer Science Teacher, Hamilton Southeastern High School
Fishers, Indiana

Julie Alano teaches computer science at Hamilton Southeastern High School. She has expanded the computer science program since starting there as a math teacher in August 1998. The school now offers four levels of computer science, and she is working to include computer science in the K–8 curriculum. With a master's degree in educational technology, Julie also started a student-led tech squad in the school. In May 2016, Julie was named the Hamilton Southeastern Schools District Teacher of the Year. Julie serves as president of the Hoosier Heartland Chapter of the Computer Science Teachers Association (CSTA) after helping to start the group. She is also a member of the CSTA Computer Science Advocacy Leadership Team and a Code.org Computer Science Principles Facilitator.

Derek Babb

Computer Science Teacher, Omaha North Magnet High School
Omaha, Nebraska

Derek Babb is a computer science teacher at Omaha North Magnet High School. He has taught computer science for 11 years in both suburban and urban high school settings. In addition to writing for the K–12 Computer Science Framework, he has been involved in writing computer science standards for the state of Nebraska as well as local school districts. He has been involved in computer science advocacy at the local level, serving as a founding member and president of the Omaha Computer Science Teachers Association chapter. Derek is committed to expanding computer science education in his school and district and hopes to serve as a coach and advisor to new computer science teachers as they get started.

Julia Bell

Associate Professor of Computer Science, Walters State Community College
Morristown, Tennessee

Julia Bell is an associate professor of computer science at Walters State Community College. She previously worked with Northwest Arkansas Community College in Bentonville, AR. She has worked as a networking program director and systems analyst for Fayetteville Police Department, TN Code Academy programming instructor, tnAchieves scholars mentor, Quality Matters certified designer and course reviewer, and A.C.E.-certified forensic examiner. She received the 2012 Faculty of the Year award and multiple Good as Gold Faculty awards from Phi Theta Kappa. For two years she has worked with Nicewonger Foundation Summer Coding Camp teaching coding and networking to middle and high school students, as a writer for the Interim CSTA K–12 Computer Science Standards, Revised 2016, and as a Nicewonger Foundation mobile presenter. Julia's research interests include cybersecurity, cybersecurity impacts on children, and NSX networks of the future.

Tiara Booker-Dwyer

Education Program Specialist, Maryland State Department of Education
Baltimore, Maryland

Tiara Booker-Dwyer is an education program specialist for the Maryland State Department of Education (MSDE). In this position, she provides leadership to local school systems and postsecondary institutions to plan, develop, and implement computer science, engineering, and technology education instructional programs. She develops, coordinates, and facilities professional learning experiences and assists in departmental initiatives related to school reform and science, technology, engineering, and math education. Prior to joining MSDE, Tiara was a program director for the Maryland Business Roundtable, where she developed strategic alliances and led stakeholder groups in the implementation of programs designed to prepare students for future job markets. Tiara began her career conducting research in neuroscience at Johns Hopkins before transitioning into education, where she is collaboratively leading efforts to implement high-quality computer science learning experiences statewide.

Leigh Ann DeLyser

Director of Education and Research, CSNYC
New York, New York

Leigh Ann DeLyser is the director of education and research at the NYC Foundation for Computer Science Education (CSNYC). In this role, Leigh Ann is working to expand computer science to all schools in the New York City public school system. CSNYC is the private partner in the $80 million initiative requiring every school to offer one unit of computer science to every student in public schools. She is a co-author of the Running on Empty report, a 50-state analysis of computer science standards. Prior to obtaining her doctorate in computer science and cognitive psychology from Carnegie Mellon University, Leigh Ann was a high school computer science and math teacher and a two-term member of the board of directors of the Computer Science Teachers Association. She also helped start the Academies for Software Engineering in New York City as a proof of concept that all students could learn computer science.

Caitlin McMunn Dooley

Deputy Superintendent for Curriculum and Instruction,
Georgia Department of Education
Associate Professor, Georgia State University
Atlanta, Georgia

Caitlin McMunn Dooley is the deputy superintendent for curriculum and instruction for Georgia's public schools and an associate professor at Georgia State University. Her research on children's and teachers' learning, digital literacy development, and computational thinking has been published in more than 50 articles, chapters, and editorials. Caitlin's latest National Science Foundation-funded project studies how to integrate computer science across the curriculum in Grades 3–5. Caitlin promotes the integration of computer science as an essential part of K–12 academic learning and of digital literacy development. Caitlin taught early childhood and elementary grades in Virginia before becoming a teacher educator, professor, mother, researcher, and school leader.

Diana Franklin

Director of Computer Science Education, UChicago STEM Ed
Chicago, Illinois

Diana Franklin is the director of computer science education at UChicago STEM Ed. She has taught college-level computing for 14 years as tenured teaching faculty at University of California, Santa Barbara and as an associate professor at California Polytechnic State University, San Luis Obispo. Her research focuses on understanding how children learn computing concepts in elementary school to design learning environments and curricula. She is a recipient of the National Science Foundation CAREER award, the National Center for Women & Information Technology faculty mentoring award, and three teaching awards. She is the author of A Practical Guide to Gender Diversity for CS Faculty, from Morgan Claypool.

Dan Frost

Senior Lecturer, University of California, Irvine
Irvine, California

Dan Frost has maintained a strong interest in K–12 computer science education during the two decades he has taught computer science at the university level. His 1997 SIGCSE (Special Interest Group on Computer Science Education) paper Fourth Grade Computer Science, based on many years of in-classroom teaching and research, contributed to the recent upswing in computer science education at the primary and secondary levels. From 1997 to 1999, Dan chaired the Computer Science Teachers Association committee that wrote A Model Curriculum for K–12 Computer Science: Level 1 Objectives and Outlines. He was the principal investigator on an National Science Foundation grant that intertwined computer science, game design, and cultural education for American Indian high school students, who created games that retold traditional stories and cultural practices.

Mark A. Gruwell

Co-Facilitator, Iowa STEM Council Computer Science Workgroup
Estherville, Iowa

Mark A. Gruwell co-facilitates the Iowa STEM Council Computer Science Workgroup, which advocates and promotes K–12 computer science education initiatives in Iowa. Mark started computer programming in high school and continued programming in college, where he achieved recognition from the Florida Bandmasters Association for creating BandBase, an application that automates scheduling and other processes for district and statewide music festivals. While serving as chief academic officer of Iowa Lakes Community College, Mark led efforts to create and implement the college's two-year Computer Gaming Design & Development Program. In addition to teaching summer computer camps, Mark is an entrepreneur who designs computer applications that assist colleges with student advising, adjunct faculty scheduling and credentialing, and accreditation.

Maya Israel

Assistant Professor, University of Illinois at Urbana Champaign
Champaign, Illinois

Maya Israel is an assistant professor in the College of Education at the University of Illinois at Urbana Champaign. Her primary areas of specialization include supporting students with disabilities and other struggling learners' meaningful engagement in science, technology, engineering, and math (STEM), with emphases on computational thinking and computer programming. She researches accessible instructional models and technologies that promote student engagement, collaborative problem solving, and persistence. Maya is currently a co-principal investigator on a National Science Foundation STEM+C grant to create learning trajectories that align computational thinking with math instruction. She has published in top-ranking journals such as Exceptional Children, Journal of Research on Technology in Education, Journal of Research in Science Teaching, and Computers & Education.

Vanessa Jones

Instructional Technology Design Coach, Austin Independent School District
Austin, Texas

Vanessa Jones is an instructional technology design coach for the Austin Independent School District. She is a Code.org Texas facilitator and has trained hundreds of educators in computer science basics. She was named one of Intel's Education 20 Most Inspiring Educators and has presented at numerous national and state conferences, such as the International Society for Technology in Education, showcasing computer science initiatives. Vanessa has written several grants to enrich computer science infusion in the elementary and middle school classroom. She is a member of the CS4TX (Computer Science for Texas) organization, and her passion is to continue to develop a community of computer science learners to learn something new every day. She believes that all students should have access to understand the basics in computer science and that computer science is the great equity equalizer.

Richard Kick

Mathematics and Computer Science Teacher, Newbury Park High School
Newbury Park, California

Richard Kick teaches math and computer science at Newbury Park High School. Rich earned a mathematics education degree from the University of Illinois at Urbana Champaign and a master's degree in mathematics from Chicago State University. He taught Advanced Placement® (AP) computer science using Pascal beginning in the first year of AP computer science, followed by C++ and then Java. After working as a C++ programmer at Fermi National Accelerator Laboratory, Rich served as a College Board exam reader, table leader, question leader, and Computer Science Test Development Committee member. He is a five-time Computer Science Principles Pilot instructor and is currently the co-chair of the Computer Science Principles Development Committee.

Heather Lageman

Executive Director of Leadership Development, Baltimore County Public Schools
Towson, Maryland

Heather Lageman serves as the executive director of leadership development for Baltimore County Public Schools in the Office of Organizational Development. She is president of the Learning Forward Maryland Affiliate, president-elect of the Learning Forward Foundation, and vice president of Maryland Affiliate of the Association for Supervision and Curriculum Development. Heather has served as the director of curriculum for the Maryland State Department of Education (MSDE), and she managed the statewide implementation of the Maryland Teacher Induction Program. During Race to the Top, she served as Race to the Top local education agency director for Maryland and managed both programmatic and fiscal aspects of the district projects. Heather formerly served as an MSDE specialist managing No Child Left Behind Title IIA and providing leadership for the state teacher professional development programs and policies, as well as the professional development coordinators. Prior to that, she served as a specialist in MSDE's Secondary English Language Arts Office, where her responsibilities included development and implementation of county curriculum, assessments, and professional development. Heather is dedicated to supporting the professional learning and development of inspired and innovative educators.

Todd Lash

Doctoral Student/Contributing Member, University of Illinois Doctoral Student/ CSTA K–8 Task Force
Champaign, Illinois

Todd Lash is an elementary educator of 17 years. Having enjoyed time as a classroom teacher and school library media specialist over the last three years, Todd worked as an instructional coach for computer science. Currently a first-year doctoral student at the University of Illinois, Todd served on the team for the Interim CSTA K–12 Computer Science Standards, Revised 2016 and is active in the Computer Science Teachers Association (CSTA) K–8 Task Force. As part of a National Science Foundation STEM+C grant, Todd is part of a team working to develop computer science learning trajectories through an integrated math curriculum.

Irene Lee

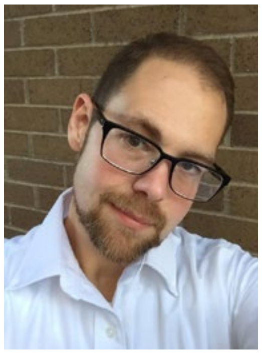
Researcher, Massachusetts Institute of Technology
Cambridge, Massachusetts

Irene Lee is a researcher in the Massachusetts Institute of Technology's Scheller Teacher Education Program and Education Arcade. She is the founder and director of Project GUTS: Growing Up Thinking Scientifically and previously was the principal investigator of New Mexico Computer Science for All, Young Women Growing Up Thinking Computationally, and GUTS y Girls. Irene is the chair of the Computer Science Teachers Association (CSTA) Computational Thinking Task Force and served as a member of the team for the Interim CSTA K–12 Computer Science Standards, Revised 2016. Previously, she designed and developed educational and video games for Electronic Arts and Theatrix Interactive and worked in informal education as a science specialist. Irene is the past president of the Supercomputing Challenge and the Swarm Development Group and the past director of the Learning Lab at Santa Fe Institute.

Carl Lyman

Specialist over Information Technology Class Cluster, Utah State Board of Education
Salt Lake City, Utah

Carl Lyman started teaching programming to his third grade students in 1982. He brought his Apple II+ computer from home to school each day. He taught problem solving and programming to his students using Turtle Graphics and Terrapin Logo. His students learned problem solving, loops, if-then statements, and procedures. Today it is called "coding." Carl spent more than 30 years as a teacher—six years as an elementary teacher and more than 27 years teaching computer classes, computer applications, programming, digital media, and information technology support classes. For 10 years, he worked at the Utah State Board of Education in career and technical education, overseeing information technology (which includes computer science), digital media, web development, and computer programming courses. He worked hard in Utah to train teachers to teach computer science and make more computer science opportunities available for students. Carl has recently retired.



Daniel Moix

Computer Science Education Specialist, Arkansas School for Mathematics, Sciences & Arts
Hot Springs, Arkansas

Daniel Moix has taught computer science since 2003 at the Arkansas School for Mathematics, Sciences & Arts; College of the Ouachitas; and Bryant High School. He is the Computer Science Teachers Association (CSTA) Arkansas vice president, a member of the CSTA Computer Science Advocacy Leadership Team, and Arkansas's first K–12 computer science education specialist. Daniel was the 9–12 grade-level lead for the Interim CSTA K–12 Computer Science Standards, Revised 2016 and a recipient of the 2015 Presidential Award for Excellence in Mathematics and Science Teaching.

Dianne O'Grady-Cunniff

Computer Science Teacher, La Plata High School
La Plata, Maryland

Dianne O'Grady-Cunniff is working on projects to bring computer science to all students in K–12, after teaching computer science in university, college, and high school. Focusing on curriculum development and teacher training and support for the past few years, she is a lead teacher for the CS Matters in Maryland team and a Code.org facilitator. She worked with Charles County Public Schools to train hundreds of teachers to teach computer science and bring computer science to every school in the district for the past two years. Computer science education is Dianne's passion, and she believes that every child should have the opportunity to create with technology.

Anthony A. Owen

Coordinator of Computer Science, Arkansas Department of Education
Little Rock, Arkansas

Anthony A. Owen serves as Arkansas's coordinator of computer science within the Arkansas Department of Education (ADE). He began his career in education as a math and science teacher and then served as ADE's K–12 mathematics and computer science specialist. Anthony currently serves as the state lead for the development and implementation of Gov. Asa Hutchinson's computer science initiative. In this role, he advises and coordinates with multiple national and state entities, including serving as a member of the Southern Regional Education Board's Commission on Computer Science, Information Technology and Related Career Fields, as well as Gov. Hutchinson's Computer Science Task Force, which identifies the state's computer science and technology needs. Anthony was recently elected as the state department representative to the Computer Science Teachers Association. Anthony received a bachelor of science degree in mathematics with minors in education and computer science and a master's degree in educational leadership from Henderson State University. He received a juris doctorate from the Bowen School of Law at the University of Arkansas at Little Rock in 2013 and was admitted to the Arkansas bar in 2014.

Minsoo Park

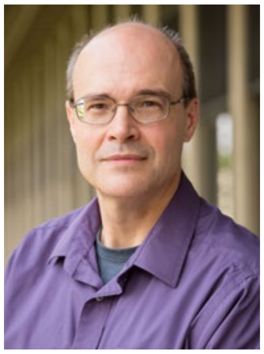
Director of Teaching and Learning, Countryside School
Champaign, Illinois

Minsoo Park is the director of teaching and learning at Countryside School. In 10 years of teaching, he has primarily served as a middle school computer science/algebra teacher, a technology coordinator, and an Middle Years Program International Baccalaureate coordinator in Chicago Public Schools. For past three years, he served as an enrichment and technology specialist in Unit 4 Champaign School District, implemented student-driven projects, and developed schoolwide computer science and math integration units that emphasize the metacognition and learning process through computer science concepts and computational thinking practices. He has been recognized with the Those Who Excel Award by the Illinois State Board of Education for technology innovation. He is certified in computer science, math, social science, physical science, and technology education.

Shaileen Crawford Pokress

Visiting Scholar, Wyss Institute at Harvard; K–12 Curriculum Designer
Cambridge, Massachusetts

Shay Pokress is a curriculum developer specializing in K–12 computer science. Shay is currently a visiting scholar at Harvard's Wyss Institute for Bioinspired Engineering, where she is developing standards-based computer science curricula around the unique capabilities of Root, a robot designed specifically for learning computational thinking. Prior to joining the Root team, Shay served as director of instruction at Project Lead The Way, where she developed Advanced Placement® computer science courses and was the lead writer for Launch Computer Science, a widely adopted curriculum that uses an infusion approach to connect problem-based computer science to K–5 content standards. At the Massachusetts Institute of Technology Media Lab, she developed and directed the education program for App Inventor, a platform for building mobile apps that aims to democratize mobile computing. As senior research associate at TERC, Shay focused on teacher professional development in mathematics and science. Shay earned her bachelor of science degree in computer science from Cornell University and her master's degree from Harvard Graduate School of Education. She believes that access to quality computer science education is a social justice issue.



George Reese

Director of MSTE, MSTE Office at University of Illinois at Urbana Champaign
Champaign, Illinois

George Reese is the director of the Office for Mathematics, Science, and Technology Education (MSTE) in the College of Education at the University of Illinois at Urbana Champaign. The MSTE office works to enhance technology-supported teaching and learning in mathematics and science through curriculum design and teacher professional development partnerships with schools and districts. Prior to working at MSTE, George was a high school mathematics teacher at the Santa Fe Indian School in Santa Fe, NM. He is currently the board president of the Illinois Council of Teachers of Mathematics.

Hal Speed

Founder, CS4TX
Austin, Texas

Hal Speed is an advocate for computer science education for all students in grades K–12 and believes these skills are necessary for socioeconomic mobility and the future prosperity of nations in the digital age. He founded CS4TX (Computer Science for Texas) to coordinate activities across Texas and represent the state in the national CSforAll initiative and the Expanding Computing Education Pathways Alliance. Hal is a product experience engineer at Dell and serves as a Code.org facilitator, Computer Science Teachers Association committee chair, and Texas Computer Education Association Tech-Apps/Computer Science Special Interest Group vice president. He holds a bachelor's degree in electrical engineering and a master's degree in business administration from Virginia Tech.

Alfred Thompson

Computer Science Teacher, Bishop Guertin High School
Nashua, New Hampshire

Alfred Thompson is a high school computer science teacher at Bishop Guertin High School and is a member of the Computer Science Teachers Association board. He has been a professional software developer, a textbook author, a developer evangelist, a school technology coordinator, a school board member, and more. Alfred sees himself as a computer science education activist working to help reach more young people with the knowledge that they can make the world a better place through software. He is the author of the popular Computer Science Teacher blog.

Bryan Twarek

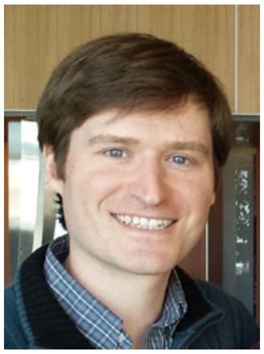
Computer Science Program Administrator, San Francisco Unified School District
San Francisco, California

Bryan Twarek (BT) is the computer science program administrator for the San Francisco Unified School District (SFUSD), where he is working to expand computer science instruction to all students and all schools within San Francisco public schools. His goal is to ensure that all SFUSD students have equitable access to rigorous and engaging computer science instruction, from prekindergarten to 12th grade. To this end, he oversees policy, curriculum development, and professional development. He is also a writer for the Interim CSTA K–12 Computer Science Standards, Revised 2016 and a board member for Computer Using Educators San Francisco (an affiliate of the International Society for Technology in Education). Previously, he has worked as dean, teacher, instructional coach, and technology integration specialist. BT graduated from Yale University with a degree in psychology and human neuroscience. He earned his master's degree in urban education policy and administration from Loyola Marymount University.

A. Nicki Washington

Associate Professor, Computer Science, Winthrop University
Rock Hill, South Carolina

Nicki Washington is an associate professor of computer science at Winthrop University. Prior to this, she was an associate professor at Howard University. Her research interests focus on computer science education, specifically increasing the participation of underrepresented minorities. Her research projects have included partnerships with District of Columbia Public Schools, Exploring CS, and Google. Her most recent research includes the development of the Computer Science Attitude and Identity Survey, a tool for measuring the impact of ethnic identity on student attitudes toward and pursuit and persistence in computer science. She is a 2000 graduate of Johnson C. Smith University.



David Weintrop

Postdoctoral Researcher, UChicago STEM Ed
Chicago, Illinois

David Weintrop is a postdoctoral researcher at UChicago STEM Ed at the University of Chicago. He has a doctorate in learning sciences from Northwestern University and a bachelor of science degree in computer science from the University of Michigan. Before starting his academic career, he spent five years working as a developer at a pair of software startups in Chicago. David's research focuses on the design, implementation, and evaluation of accessible and engaging introductory programming environments. He is also interested in the use of technological tools in supporting exploration and expression across diverse contexts including science, technology, engineering, and math classrooms and informal spaces. His work lies at the intersection of human-computer interaction, design, and learning sciences. David won the gold medal in the Student Research Competition at the 2015 ACM Computer Science Education conference for his dissertation work and has presented his research at Google, the Massachusetts Institute of Technology, and conferences around the world.

Development Staff

Pat Yongpradit

Chief Academic Officer, Code.org

Pat Yongpradit is the chief academic officer for Code.org, a nonprofit dedicated to promoting computer science education. As a national voice on K–12 computer science education, his passion is to bring computer science opportunities to every school and student in the United States. Throughout his career as a high school computer science teacher, he inspired students to create mobile games and apps for social causes and implemented initiatives to broaden participation in computer science among underrepresented groups. He has been featured in the book American Teacher: Heroes in the Classroom and in 2010 was recognized as a Microsoft Worldwide Innovative Educator. He hold a bachelor's degree in neurobiology, a master's degree in secondary education, and is certified in biology, physics, math, health, and technology education. While Pat currently spends more time focused on computer science from a national perspective, his heart is still in the classroom.

Katie Hendrickson

Advocacy and Policy Manager, Code.org

Katie Hendrickson is an advocacy and policy manager at Code.org. She works on state policy and advocacy issues, including state implementation of computer science education initiatives. Prior to joining Code.org, she was a 2014–15 Albert Einstein Distinguished Educator Fellow, placed at the National Science Foundation in the Computer and Information Sciences and Engineering Directorate. She taught secondary mathematics for six years at Alexander Middle School and Athens Middle School, where she received the Buck Martin Secondary State Award for excellence in mathematics teaching from the Ohio Council of Teachers of Mathematics. She co-founded the Southeast Ohio Math Teachers' Circle, and her dissertation research explored teacher identity and professional development. She holds a doctorate in curriculum and instruction and a master's degree in cultural studies in education from Ohio University.

Rachel Phillips

Director of Research and Evaluation, Code.org

Rachel Phillips is the director of research and evaluation at Code.org. Prior to joining Code.org, she conducted evaluations and developed curriculum for the Big History Project. Additionally, she was the program director for a National Science Foundation-funded research project studying the impacts of tinkering and making on low-income youth and how those activities can increase participation in the science, technology, engineering, and math (STEM) fields. Her research interests include student and teacher learning in formal education spaces, with a particular focus on traditionally marginalized and underserved youth. Most of her research has been in the context of the STEM fields, and her more recent publications are related to learning in online gaming environments and the methodology used to study learning in these types of environments. She earned her doctorate in learning sciences from the University of Washington in 2011 and a master of arts in teaching from American University in 2006.

Debbie Carter

Editor, Educational Consultant

Miranda Parker

Intern, Georgia Tech

Lian Halbert

Operations, Code.org

Consultants/Process Advisors

Courtney K. Blackwell

Outlier Research & Evaluation, UChicago STEM Education, University of Chicago

Jeanne Century

Outlier Research & Evaluation, UChicago STEM Education, University of Chicago

Jennifer Childress

Achieve

Thomas Keller

Maine Mathematics and Science Alliance

Heather King

Outlier Research & Evaluation, UChicago STEM Education, University of Chicago

Appendix C: Glossary

The glossary includes definitions of terms used in the statements in the framework. These terms are defined for readers of the framework and are not necessarily intended to be the definitions or terms that are seen by students.

Table C.1: Glossary Terms

TERM	DEFINITION
abstraction	**(***process***):** The process of reducing complexity by focusing on the main idea. By hiding details irrelevant to the question at hand and bringing together related and useful details, abstraction reduces complexity and allows one to focus on the problem. **(***product***):** A new representation of a thing, a system, or a problem that helpfully reframes a problem by hiding details irrelevant to the question at hand. [MDESE, 2016]
accessibility	The design of products, devices, services, or environments for people who experience disabilities. Accessibility standards that are generally accepted by professional groups include the Web Content Accessibility Guidelines (WCAG) 2.0 and Accessible Rich Internet Applications (ARIA) standards. [Wikipedia]
algorithm	A step-by-step process to complete a task.
analog	The defining characteristic of data that is represented in a continuous, physical way. Whereas digital data is a set of individual symbols, analog data is stored in physical media, such as the surface grooves on a vinyl record, the magnetic tape of a VCR cassette, or other nondigital media. [Techopedia]
app	A type of application software designed to run on a mobile device, such as a smartphone or tablet computer. Also known as a *mobile application*. [Techopedia]
artifact	Anything created by a human. *See* computational artifact *for the definition used in computer science.*
audience	Expected end users of a computational artifact or system.
accessibility	The design of products, devices, services, or environments for people who experience disabilities. Accessibility standards that are generally accepted by professional groups include the Web Content Accessibility Guidelines (WCAG) 2.0 and Accessible Rich Internet Applications (ARIA) standards. [Wikipedia]
authentication	The verification of the identity of a person or process. [FOLDOC]

Appendix C: Glossary

automate; automation	**automate:** To link disparate systems and software so that they become self-acting or self-regulating. [Ross, 2016] **automation:** The process of automating.
Boolean	A type of data or expression with two possible values: *true* and *false*. [FOLDOC]
bug	An error in a software program. It may cause a program to unexpectedly quit or behave in an unintended manner. [Tech Terms] The process of finding and correcting errors (bugs) is called debugging. [Wikipedia]
code	Any set of instructions expressed in a programming language. [MDESE, 2016]
comment	A programmer-readable annotation in the code of a computer program added to make the code easier to understand. Comments are generally ignored by machines. [Wikipedia]
complexity	The minimum amount of resources, such as memory, time, or messages, needed to solve a problem or execute an algorithm. [NIST/DADS]
component	An element of a larger group. Usually, a component provides a particular service or group of related services. [Tech Terms, TechTarget]
computational	Relating to computers or computing methods.
computational artifact	Anything created by a human using a computational thinking process and a computing device. A computational artifact can be, but is not limited to, a program, image, audio, video, presentation, or web page file. [College Board, 2016]
computational thinking	The human ability to formulate problems so that their solutions can be represented as computational steps or algorithms to be executed by a computer. [Lee, 2016]
computer	A machine or device that performs processes, calculations, and operations based on instructions provided by a software or hardware program. [Techopedia]
computer science	The study of computers and algorithmic processes, including their principles, their hardware and software designs, their implementation, and their impact on society. [ACM, 2006]
computing	Any goal-oriented activity requiring, benefiting from, or creating algorithmic processes. [MDESE, 2016]
computing device	A physical device that uses hardware and software to receive, process, and output information. Computers, mobile phones, and computer chips inside appliances are all examples of computing devices.
computing system	A collection of one or more computers or computing devices, together with their hardware and software, integrated for the purpose of accomplishing shared tasks. Although a computing system can be limited to a single computer or computing device, it more commonly refers to a collection of multiple connected computers, computing devices, and hardware.

conditional	A feature of a programming language that performs different computations or actions depending on whether a programmer-specified Boolean condition evaluates to *true* or *false*. [MDESE, 2016] (A conditional *could refer to* a conditional statement, conditional expression, *or* conditional construct.)
configuration	(*process*): Defining the options that are provided when installing or modifying hardware and software or the process of creating the configuration (product). [TechTarget] (*product*): The specific hardware and software details that tell exactly what the system is made up of, especially in terms of devices attached, capacity, or capability. [TechTarget]
connection	A physical or wireless attachment between multiple computing systems, computers, or computing devices.
connectivity	A program's or device's ability to link with other programs and devices. [Webopedia]
control; **control structure**	**control:** *(in general)* The power to direct the course of actions. *(in programming)* The use of elements of programming code to direct which actions take place and the order in which they take place. **control structure**: A programming (code) structure that implements control. Conditionals and loops are examples of control structures.
culture; **cultural practices**	**culture:** A human institution manifested in the learned behavior of people, including their specific belief systems, language(s), social relations, technologies, institutions, organizations, and systems for using and developing resources. [NCSS, 2013] **cultural practices:** The displays and behaviors of a culture.
cybersecurity	The protection against access to, or alteration of, computing resources through the use of technology, processes, and training. [TechTarget]
data	Information that is collected and used for reference or analysis. Data can be digital or nondigital and can be in many forms, including numbers, text, show of hands, images, sounds, or video. [CAS, 2013; Tech Terms]
data structure	A particular way to store and organize data within a computer program to suit a specific purpose so that it can be accessed and worked with in appropriate ways. [TechTarget]
data type	A classification of data that is distinguished by its attributes and the types of operations that can be performed on it. Some common data types are integer, string, Boolean (true or false), and floating-point.
debugging	The process of finding and correcting errors (bugs) in programs. [MDESE, 2016]
decompose; **decomposition**	**decompose:** To break down into components. **decomposition:** Breaking down a problem or system into components. [MDESE, 2016]

device	A unit of physical hardware that provides one or more computing functions within a computing system. It can provide input to the computer, accept output, or both. [Techopedia]
digital	A characteristic of electronic technology that uses discrete values, generally 0 and 1, to generate, store, and process data. [Techopedia]
digital citizenship	The norms of appropriate, responsible behavior with regard to the use of technology. [MDESE, 2016]
efficiency	A measure of the amount of resources an algorithm uses to find an answer. It is usually expressed in terms of the theoretical computations, the memory used, the number of messages passed, the number of disk accesses, etc. [NIST/DADS]
encapsulation	The technique of combining data and the procedures that act on it to create a type. [FOLDOC]
encryption	The conversion of electronic data into another form, called ciphertext, which cannot be easily understood by anyone except authorized parties. [TechTarget]
end user (or user)	A person for whom a hardware or software product is designed (as distinguished from the developers). [TechTarget]
event	Any identifiable occurrence that has significance for system hardware or software. User-generated events include keystrokes and mouse clicks; system-generated events include program loading and errors. [TechTarget]
event handler	A procedure that specifies what should happen when a specific event occurs.
execute; execution	**execute**: To carry out (or "run") an instruction or set of instructions (program, app, etc.). **execution**: The process of executing an instruction or set of instructions. [FOLDOC]
hardware	The physical components that make up a computing system, computer, or computing device. [MDESE, 2016]
hierarchy	An organizational structure in which items are ranked according to levels of importance. [TechTarget]
human–computer interaction (HCI)	The study of how people interact with computers and to what extent computing systems are or are not developed for successful interaction with human beings. [TechTarget]
identifier	The user-defined, unique name of a program element (such as a variable or procedure) in code. An identifier name should indicate the meaning and usage of the element being named. [Techopedia]
implementation	The process of expressing the design of a solution in a programming language (code) that can be made to run on a computing device.
inference	A conclusion reached on the basis of evidence and reasoning. [Oxford]

input	The signals or instructions sent to a computer. [Techopedia]
integrity	The overall completeness, accuracy, and consistency of data. [Techopedia]
Internet	The global collection of computer networks and their connections, all using shared protocols to communicate. [CAS, 2013]
iterative	Involving the repeating of a process with the aim of approaching a desired goal, target, or result. [MDESE, 2016]
loop	A programming structure that repeats a sequence of instructions as long as a specific condition is true. [Tech Terms]
memory	Temporary storage used by computing devices. [MDESE, 2016]
model	A representation of some part of a problem or a system. [MDESE, 2016] *Note: This definition differs from that used in science.*
modularity	The characteristic of a software/web application that has been divided (*decomposed*) into smaller modules. An application might have several procedures that are called from inside its main procedure. Existing procedures could be reused by recombining them in a new application. [Techopedia]
module	A software component or part of a program that contains one or more procedures. One or more independently developed modules make up a program. [Techopedia]
network	A group of computing devices (personal computers, phones, servers, switches, routers, etc.) connected by cables or wireless media for the exchange of information and resources.
operation	An action, resulting from a single instruction, that changes the state of data. [Free Dictionary]
packet	The unit of data sent over a network. [Tech Terms]
parameter	A special kind of variable used in a procedure to refer to one of the pieces of data received as input by the procedure. [MDESE, 2016]
piracy	The illegal copying, distribution, or use of software. [TechTarget]
procedure	An independent code module that fulfills some concrete task and is referenced within a larger body of program code. The fundamental role of a procedure is to offer a single point of reference for some small goal or task that the developer or programmer can trigger by invoking the procedure itself. [Techopedia] In this framework, *procedure* is used as a general term that may refer to an actual procedure or a method, function, or module of any other name by which modules are known in other programming languages.
process	A series of actions or steps taken to achieve a particular outcome. [Oxford]

program; programming	**program (n):** A set of instructions that the computer executes to achieve a particular objective. [MDESE, 2016] **program (v):** To produce a program by programming. **programming:** The craft of analyzing problems and designing, writing, testing, and maintaining programs to solve them. [MDESE, 2016]
protocol	The special set of rules used by endpoints in a telecommunication connection when they communicate. Protocols specify interactions between the communicating entities. [TechTarget]
prototype	An early approximation of a final product or information system, often built for demonstration purposes. [TechTarget, Techopedia]
redundancy	A system design in which a component is duplicated, so if it fails, there will be a backup. [TechTarget]
reliability	An attribute of any system that consistently produces the same results, preferably meeting or exceeding its requirements. [FOLDOC]
remix	The process of creating something new from something old. Originally a process that involved music, remixing involves creating a new version of a program by recombining and modifying parts of existing programs, and often adding new pieces, to form new solutions. [Kafai & Burke, 2014]
router	A device or software that determines the path that data packets travel from source to destination. [TechTarget]
scalability	The capability of a network to handle a growing amount of work or its potential to be enlarged to accommodate that growth. [Wikipedia]
security	*See the definition for* cybersecurity.
simulate; simulation	**simulate:** To imitate the operation of a real-world process or system. **simulation:** Imitation of the operation of a real-world process or system. [MDESE, 2016]
software	Programs that run on a computing system, computer, or other computing device.
storage	*(place)* A place, usually a device, into which data can be entered, in which the data can be held, and from which the data can be retrieved at a later time. [FOLDOC] *(process)* A process through which digital data is saved within a data storage device by means of computing technology. Storage is a mechanism that enables a computer to retain data, either temporarily or permanently. [Techopedia]
string	A sequence of letters, numbers, and/or other symbols. A string might represent, for example, a name, address, or song title. Some functions commonly associated with strings are length, concatenation, and substring. [TechTarget]

structure	A general term used in the framework to discuss the concept of encapsulation without specifying a particular programming methodology.
switch	A high-speed device that receives incoming data packets and redirects them to their destination on a local area network (LAN). [Techopedia]
system	A collection of elements or components that work together for a common purpose. [TechTarget] *See also the definition for* computing system.
test case	A set of conditions or variables under which a tester will determine whether the system being tested satisfies requirements or works correctly. [STF]
topology	The physical and logical configuration of a network; the arrangement of a network, including its nodes and connecting links. A logical topology is the way devices appear connected to the user. A physical topology is the way they are actually interconnected with wires and cables. [PCMag]
troubleshooting	A systematic approach to problem solving that is often used to find and resolve a problem, error, or fault within software or a computing system. [Techopedia, TechTarget]
user	*See the definition for* end user.
variable	A symbolic name that is used to keep track of a value that can change while a program is running. Variables are not just used for numbers; they can also hold text, including whole sentences *(strings)* or logical values *(true* or *false)*. A variable has a data type and is associated with a data storage location; its value is normally changed during the course of program execution. [CAS, 2013; Techopedia] *Note: This definition differs from that used in math.*

References

Some definitions can directly from the sources listed in Table C.2, while others were excerpted or adapted to include content relevant to this framework.

Table C.2: Glossary References

ACM, 2006	**A Model Curriculum for K–12 Computer Science** Tucker, A., McCowan, D., Deek, F., Stephenson, C., Jones, J., & Verno, A. (2006). *A model curriculum for K–12 computer science: Report of the ACM K–12 task force curriculum committee* (2nd ed.). New York, NY: Association for Computing Machinery.
CAS, 2013	**Computing At School's Computing in the National Curriculum: A Guide for Primary Teachers** Computing At School. (2013). *Computing in the national curriculum: A guide for primary teachers.* Belford, UK: Newnorth Print. Retrieved from http://www.computingatschool.org.uk/data/uploads/CASPrimaryComputing.pdf
College Board, 2016	**College Board Advanced Placement® Computer Science Principles** College Board. (2016). *AP Computer Science Principles course and exam description.* New York, NY: College Board. Retrieved from https://secure-media.collegeboard.org/digitalServices/pdf/ap/ap-computer-science-principles-course-and-exam-description.pdf
FOLDOC	**Free On-Line Dictionary of Computing** Free on-line dictionary of computing. (n.d.). Retrieved from http://foldoc.org
Free Dictionary	**The Free Dictionary** The free dictionary. (n.d.). Retrieved from http://www.thefreedictionary.com
Kafai & Burke, 2014	**Connected Code: Why Children Need to Learn Programming** Kafai, Y., & Burke, Q. (2014). *Connected code: Why children need to learn programming.* Cambridge, MA: MIT Press.
Lee, 2016	**Reclaiming the Roots of CT** Lee, I. (2016). Reclaiming the roots of CT. *CSTA Voice: The Voice of K–12 Computer Science Education and Its Educators, 12*(1), 3–4. Retrieved from http://www.csteachers.org/resource/resmgr/Voice/csta_voice_03_2016.pdf
MDESE, 2016	**Massachusetts Digital Literacy and Computer Science (DL&CS) Standards** Massachusetts Department of Elementary and Secondary Education. (2016, June). 2016 *Massachusetts digital literacy and computer science (DLCS) curriculum framework.* Malden, MA: Author. Retrieved from http://www.doe.mass.edu/frameworks/dlcs.pdf

Table continues on next page

Table continued from previous page

NCSS, 2013	**College, Career & Civic Life (C3) Framework for Social Studies State Standards** National Council for the Social Studies. (2013). *The college, career, and civic life (C3) framework for social studies state standards: Guidance for enhancing the rigor of K–12 civics, economics, geography, and history.* Silver Spring, MD: Author. Retrieved from http://www.socialstudies.org/system/files/c3/C3-Framework-for-Social-Studies.pdf
NIST/DADS	**National Institute of Science and Technology Dictionary of Algorithms and Data Structures** Pieterse, V., & Black, P. E. (Eds.). (n.d). *Dictionary of algorithms and data structures.* Retrieved from https://xlinux.nist.gov/dads//
Oxford	**Oxford Dictionaries** Oxford dictionaries. (n.d.). Retrieved from http://www.oxforddictionaries.com/us
PCmag	**PCmag.com Encyclopedia** PCmag.com encyclopedia. (n.d.). Retrieved from http://www.pcmag.com/encyclopedia/term/46301/logical-vs-physical-topology
Ross, 2016	**What Is Automation** Ross, B. (2016, May 10). What is automation and how can it improve customer service? *Information Age.* Retrieved from http://www.information-age.com/industry/soft-ware/123461408/what-automation-and-how-can-it-improve-customer-service
STF	**Software Testing Fundamentals** Software testing fundamentals. (n.d). Retrieved from http://softwaretestingfundamentals.com
Tech Terms	**Tech Terms** Tech terms computer dictionary. (n.d.). Retrieved from http://www.techterms.com
Techopedia	**Techopedia** Techopedia technology dictionary. (n.d.). Retrieved from https://www.techopedia.com/dictionary
TechTarget	**TechTarget Network** TechTarget network. (n.d.). Retrieved from http://www.techtarget.com/network
Webopedia	**Webopedia** Webopedia. (n.d.). Retrieved from http://www.webopedia.com
Wikipedia	**Wikipedia** Wikipedia: The free encyclopedia. (n.d.). Retrieved from https://www.wikipedia.org/

Appendix D: Early Childhood Research Review

Defined as "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), computer science is predicated on the use of computer technologies—a topic historically debated in the early childhood education sector. Importantly, engaging with technology is not the same as doing computer science, but technology access and use is often a precursor and lead-in for engaging in computer science. Thus, while caution should be taken when interpreting research on the effects of technology on children's learning and development, this research provides a background for understanding an important contextual factor in computer science education.

Despite the ubiquitous nature of technology in today's world, and even very young children's engagement in digitally mediated experiences (e.g., Rideout, 2013; Blackwell, Wartella, Lauricella, & Robb, 2015), the integration of technology in early childhood education often stands in opposition with traditional notions of prekindergarten learning environments. Though a large body of research exists on the positive impacts of high-quality educational media for young children's learning and development (e.g., Fisch & Truglio, 2001; Huston, Anderson, Wright, Linebarger, & Schmitt, 2001; Pasnik & Llorente, 2013; Penuel et al., 2012), concerns remain over potential negative consequences arising from too much screen time or exposure to violent content (see Anderson & Bushman, 2001, for review). Further, technology is often viewed as disrupting and displacing children's social interactions, imaginative play, and active learning (Donohue, 2015). Indeed, a primary reason why early childhood educators do not use technology more often, even if they have access to it, remains these foundational attitudes toward technology being the antithesis of what early education experiences should be (e.g., Cordes & Miller, 2000; Lindahl & Folkesson, 2012).

Further, in 2015 the American Academy of Pediatrics (AAP, 2015) made a landmark revision to its no screen time stance by recognizing that quality educational media, especially when used with active caregiver involvement, can benefit young children's learning and development.

While concerns remain, several major professional organizations have revised their stance on the role of digital media and technology as they recognize that digital technologies are essential tools for learning and creation (Dooley, Flint, Holbrook, May, & Albers, 2011). In 2012, the National Association for the Education of Young Children (NAEYC) and the Fred Rogers Center (FRC) released a joint position statement supporting the developmentally appropriate and intentional use technology in early childhood education (NAEYC & FRC, 2012). Further, in 2015 the American Academy of Pediatrics (AAP, 2015) made a landmark revision to its no screen time stance by recognizing that quality educational media, especially when used with active caregiver involvement, can benefit young children's learning and development.

As both the NAEYC/FRC (2012) and AAP (2015) statements suggest, technology can have a place in young children's learning but does not substitute traditional educative activities (e.g., peer-to-peer and adult-child social interactions, imaginative play, hands-on learning) that are so foundational to early childhood education. When it comes to computer science, a similar framing is taken, where computing technologies supplement hands-on learning activities. That is, while developmentally appropriate and high-quality digital computer science curricula exist, they can—and should—be used to support physical computing environments (i.e., without digital technology) in early childhood education. As Haugland (1992) pointed out, the pairing of computer-based activities with unplugged ones can enhance young children's problem-solving, abstraction, and verbal skills.

Drawing on notions from constructionism (Papert, 1980) and aligned with developmentally appropriate practices (Copple & Bredekamp, 2009), Bers, Ponte, Juelick, Viera, and Schenker (2002) articulated four tenets of technology integration in the context of computer science in early childhood education:

1. Technology environments can aid student learning by engaging in hands-on, active inquiry, and play-based activities;
2. Physical objects offer critical support for developing concrete thinking skills and understanding abstract phenomena;
3. Crosscurricular "powerful ideas" are necessary to connect all areas of learning; and
4. Self-reflection is critical to engage students in metacognitive thought processes.

While much of the computer science rhetoric has focused on preparing students for the 21st century workforce, these four tenets—and the constructionist framework more generally—elucidate opportunities for computer science education to expand beyond technical skill development and content knowledge acquisition. Viewed as such, computer science provides a platform for students to develop and engage in higher-order thinking, problem solving, and metacognitive thought processes that are transferrable outside the computer programming environment (diSessa, 2000; Papert, 1980; Clements & Natasi, 1999). As Papert (1980) articulated, the value of computer science stems from "seeing ideas from computer science not only as instruments of explanation of how learning and thinking in fact do work, but also as instruments of change that might alter, and possibly improve, the way people learn and think" (pp. 208–209).

Though no computer science concepts or practices are specified in Bers' and colleagues' (2002) tenets, they provide guidance for instructional approaches to computer science in early learning environments. Indeed, the development of tangible user interfaces is one example, in which traditional concrete manipulatives (e.g., building blocks) and digital touchscreens offer a blended learning experience for learning foundational computational knowledge and skills (e.g., Horn, AlSulaiman, & Koh, 2013; Horn, Crouser, & Bers, 2012). For example, Horn and colleagues (2013) designed a blended experience in which children engage in computer programming activities by placing stickers on the paper storybook, which then control the actions of digital characters on a smartphone or tablet computer. Thus, embedded in the traditional reading experience were opportunities for young children to engage in key computer science concepts, including sequencing and loops.

Research shows that engagement in a structured computer programming environment aids young children's number sense, visual memory, and language skills.

Others have developed tangible programming interfaces to bring the traditional online computer programming languages into the real world (e.g., Bers & Horn, 2010; Horn & Jacob, 2007; Wyeth, 2008). Originally conceptualized by Perlman (1976) in the 1970s, tangible interfaces offer a way to remove the text and motor skill barriers that limit young children's ability to engage in computer programming. For example, Bers and Horn (2010) developed a tangible programming language by using interlocking blocks that allow preschool children to physically construct a computer program instead of writing one with a keyboard and mouse on the computer. Thus, by providing multiple entry points—plugged and unplugged—for building computer science skills and knowledge, educators can offer developmentally appropriate and engaging opportunities for young children that can spark an early interest in computer science and learning overall.

A growing body of research on plugged, unplugged, and blended computer science early learning experiences suggest that children as young as 3 and 4 can engage in computer science activities—such as creating, running, and debugging a computer program—and can learn and apply key computer science concepts—including sequencing, loops, parameters, and conditionals (e.g., Bers, 2008a, 2008b; Elkin, Sullivan, & Bers, 2014; Flannery & Bers, 2013; Kazakoff, Sullivan, & Bers, 2013; Martinez, Gomez, & Benotti, 2015; Morgado, Cruz, & Kahn, 2010; Sullivan, Elkin, & Bers, 2015). For example, Martinez and colleagues (2015) found that 3- to 5-year-olds could learn and apply basic computer science concepts of sequencing, conditionals, and loops by engaging in a mix of unplugged activities and plugged robotics activities. Similarly, Gordon, Ackerman, and Breazeal (2015) used social robots to help preschool students explore various computer science concepts, including event-based logic, sequencing, and nondeterminism. Drawing on four years of work with more than 150 3- to 5-year-old children, Morgado and colleagues (2010) developed a "cookbook" of preschool computer science topics that range from simple programming syntax to more complex topics of input/output and client/server relationships.

Others have investigated whether young children can transfer knowledge from computer science-related activities to other content areas. For example, Kazakoff and colleagues (2013) showed that preschool and kindergarten children who engaged in a one-week robotics and programming workshop significantly increased in their story sequencing abilities from pre- to post-workshop, suggesting a transfer of knowledge from the computer science context to literacy. Research on social robots also supports the transfer of computer science learning to literacy (e.g., Fridin, 2014; Gordon & Breazeal, 2015; Kory Westlund & Breazeal, 2015; Movellan, Eckhardt, Virnes, & Rodriguez, 2009). A study by Kory Westlund and Breazeal (2015) that found preschool children learned new words and created stories by engaging in a storytelling game with a social robot, while Movellan and colleagues (2009) showed similar findings with even younger children aged 18–24 months, where engagement with a social robot increased knowledge of target vocabulary words by 27% over a two-week period. Further, a review by Clements (1999) showed that engagement in a structured computer programming environment aided young children's number sense, visual memory, and language skills. Importantly, unstructured computer

programming had little effect on children's ability to learn computer science concepts or transfer learning, highlighting the need for intentional teacher scaffolding that helps students make connections between computer programming and other academic and everyday experiences (Clements, 1999).

Finally, early engagement in computer programming has been noted as one way to increase students' interest in pursuing careers related to computer science and science, technology, engineering, and math (STEM) as well as decrease children's gender-based stereotypes before they fully formalize in later elementary school (Madill et al., 2007; Metz, 2007; Steele, 1997). Prior work suggests that STEM and computer science interest and self-efficacy declines in late elementary and middle school and persists through high school and postsecondary education (e.g., Google & Gallup, 2015; Pajares, 2006; Unfried, Faber, & Wiebe, 2014). This trend is even more stark for females and underrepresented minorities (e.g., Corbett, Hill, & St. Rose, 2008; Google, 2014; Master, Cheryan, & Meltzoff, 2016; Weinburgh, 1995), suggesting a need to provide early computer science-related experiences that are built upon and continued throughout children's K–12 education experience. This appendix and the K–12 Computer Science Framework itself offer a foundation for addressing this need to make computer science learning opportunities available to all students as they progress from prekindergarten to high school and beyond.

References

American Academy of Pediatrics. (2015). *Growing up digital: Media research symposium*. Washington, DC: Author. Retrieved from https://aap.org/en-us/Documents/digital_media_symposium_proceedings.pdf

Anderson, C. A., & Bushman, B. J. (2001). Effects of violent video games on aggressive behavior, aggressive cognition, aggressive affect, physiological arousal, and prosocial behavior: A meta-analytic review of the scientific literature. *Psychological Science*, *12*, 353–359.

Bers, M. U. (2008a). *Blocks to robots: Learning with technology in the early childhood classroom*. New York, NY: Teachers College Press.

Bers, M. U. (2008b). *Blocks, robots and computers: Learning about technology in early childhood*. New York, NY: Teachers College Press.

Bers, M. U., & Horn, M. S. (2010). Tangible programming in early childhood: Revisiting developmental assumptions through new technologies. In I. R. Berson & M. J. Berson (Eds.), *High-tech tots: Childhood in a digital world* (p. 49–70). Greenwich, CT: Information Age Publishing.

Bers, M. U., Ponte, I., Juelich, K., Viera, A., & Schenker, J. (2002). Teachers as designers: Integrating robotics into early childhood education. *Information Technology in Childhood Education Annual, 2002*(1), 123–145.

Blackwell, C. K., Wartella, E., Lauricella, A. R., & Robb, M. (2015). *Technology in the lives of educators and early childhood programs: 2014 survey of early childhood educators*. Report for the Center on Media and Human Development, Northwestern University; the Fred Rogers Center, Latrobe, PA; and the National Association for the Education of Young Children, Washington, DC.

Clements, D. (1999). The future of educational computing research: The case of computer programming. *Information Technology in Childhood Education Annual*, 147–179.

Clements, D., & Nastasi, B. (1999). Metacognition, learning, and educational computer environments. *Information Technology in Childhood Education Annual, 1*, 5–38.

Copple, C., & Bredekamp, S. (2009). *Developmentally appropriate practice in early childhood programs serving children from birth through age 8*. Washington, DC: National Association for the Education of Young Children.

Corbett, C., Hill, S. C., & St. Rose, A. (2008). *Where the girls are: The facts about gender equity in education*. Washington, DC: American Association of University Women. Retrieved from http://www.aauw.org/files/2013/02/Where-the-Girls-Are-The-Facts-About-Gender-Equity-in-Education.pdf

Cordes, C., & Miller, E. (2000). *Fool's gold: A critical look at computers in childhood*. New York, NY: Alliance for Childhood. Retrieved from http://www.allianceforchildhood.net/projects/computers/computers_reports.htm

diSessa, A. A. (2000). Changing minds: *Computers, learning and literacy*. Cambridge, MA: MIT Press.

Donohue, C. (2015). Technology and digital media as tools for teaching and learning in the digital age. In C. Donohue (Ed.), *Technology and digital media in the early years: Tools for teaching and learning* (pp. 21–35). New York, NY: Routledge. Washington, DC: National Association for the Education of Young Children.

Dooley, C. M., Flint, A. S., Holbrook, T., May, L., & Albers, P. (2011). The digital frontier in early childhood education. *Language Arts, 89*(2), 83–85.

Elkin, M., Sullivan, A., & Bers, M. U. (2014). Implementing a robotics curriculum in an early childhood Montessori classroom. *Journal of Information Technology Education: Innovations in Practice, 13*, 153–169.

Fisch, S. M., & Truglio, R. T. (Eds.). (2001). *"G" is for growing: Thirty years of research on children and Sesame Street*. Mahwah, NJ: Lawrence Erlbaum Associates.

Flannery, L. P., & Bers, M. U. (2013). Let's dance the "robot hokey-pokey!" Children's programming approaches and achievement throughout early cognitive development. *Journal of Research on Technology in Education, 46*(1), 81–101.

Fridin, M. (2014). Storytelling by a kindergarten social assistive robot: A tool for constructive learning in preschool education. *Computers & Education, 70*, 53–64.

Google. (2014). *Women who choose computer science—what really matters: The critical role of exposure and encouragement*. Mountain View, CA: Author. Retrieved from https://docs.google.com/file/d/0B-E2rcvhnlQ_a1Q4VUxWQ2dtTHM/edit

Google & Gallup. (2015). *Images of computer science: Perceptions among students, parents, and educators in the U.S.* Retrieved from http://g.co/cseduresearch

Gordon, M., Ackermann, E., & Breazeal, C. (2015, March). Social robot toolkit: Tangible programming for young children. In *Proceedings of the 10th ACM/IEEE International Conference on Human-Robot Interaction: Extended Abstracts,* Portland, OR.

Gordon, G., & Breazeal, C. (2015, January). Bayesian active learning-based robot tutor for children's word-reading skills. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence,* Austin, TX.

Haugland, S. W. (1992). The effect of computer software on preschool children's developmental gains. *Journal of Computing in Childhood Education, 3*(1), 15–30.

Horn, M. S., AlSulaiman, S., & Koh, J. (2013, June). Translating Roberto to Omar. In *Proceedings of the 12th ACM International Conference on Interaction Design and Children,* New York, NY.

Horn, M. S., Crouser, R. J., & Bers, M. U. (2012). Tangible interaction and learning: The case for a hybrid approach. *Personal and Ubiquitous Computing, 16*(4), 379–389.

Horn, M. S., & Jacob, R. J. K. (2007). Designing tangible programming languages for classroom use. In *Proceedings of TEI'07 First International Conference on Tangible and Embedded Interaction*, Baton Rouge, LA.

Huston, A. C., Anderson, D. R., Wright, J. C., Linebarger, D. L., & Schmitt, K. L. (2001). Sesame Street viewers as adolescents: The recontact study. In S. M. Fisch & R. T. Truglio (Eds.), *"G" is for growing: Thirty years of research on children and Sesame Street* (pp.131–144). Mahwah, NJ: Lawrence Erlbaum Associates.

Kazakoff, E. R., Sullivan, A., & Bers, M. (2013). The effect of a classroom-based intensive robotics and programming workshop on sequencing ability in early childhood. *Early Childhood Education Journal, 41*, 245–255.

Kory Westlund, J. M., & Breazeal, C. (2015, March). The interplay of robot language level with children's language learning during storytelling. In *Proceedings of the 10th Annual ACM/IEEE International Conference on Human-Robot Interaction* (pp. 65–66).

Lindahl, M., & Folkesson, A. (2012). Can we let computers change practice? Educators' interpretations of preschool tradition. *Computers in Human Behavior, 28*(5), 1728–1737. doi: 10.1016/j.chb.2012.04.012

Madill, H., Campbell, R. G., Cullen, D. M., Armour, M. A., Einsiedel, A. A., Ciccocioppo, A. L., . . . & Coffin, W. L. (2007). Developing career commitment in STEM-related fields: Myth versus reality. In R. J. Burke & M. C. Mattis (Eds.), *Women and minorities in science, technology, engineering and mathematics: Upping the numbers* (pp. 210–244). Northampton MA: Edward Elgar Publishing.

Martinez, C., Gomez, M. J., & Benotti, L. (2015, July). A comparison of preschool and elementary school children learning computer science concepts through a multilanguage robot programming platform. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 159–164), Vilanius, Lithuania.

Master, A., Cheryan, S., & Meltzoff, A. N. (2016). Computing whether she belongs: Stereotypes undermine girls' interest and sense of belonging in computer science. *Journal of Educational Psychology, 108*(3), 1–14. doi: 10.1037/edu0000061

Metz, S. S. (2007). Attracting the engineering of 2020 today. In R. J. Burke & M. C. Mattis (Eds.), *Women and minorities in science, technology, engineering and mathematics: Upping the numbers* (pp. 184–209). Northampton, MA: Edward Elgar Publishing.

Morgado, L., Cruz, M., & Kahn K. (2010). Preschool cookbook of computer programming topics. *Australasian Journal of Educational Technology, 26*(3), 309–326.

Movellan, J., Eckhardt, M., Virnes, M., & Rodriguez, A. (2009, March). Sociable robot improves toddler vocabulary skills. In *Proceedings of the 4th ACM/IEEE International Conference on Human Robot Interaction*, San Diego, CA.

National Association for the Education of Young Children & Fred Rogers Center. (2012). *Position statement: Technology and young children*. Washington, DC: Author. Retrieved from http://www.naeyc.org/content/technology-and-young-children

Pajares, F. (2006). Self-efficacy during childhood and adolescence: Implications for teachers and parents. In F. Pajares & T. C. Urdan (Eds.), *Self-efficacy beliefs of adolescents* (pp. 339–367). Greenwich, CA: Information Age Publishing.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York, NY: Basic Books.

Pasnik, S., & Llorente, C. (2013). *Preschool teachers can use a PBS KIDS transmedia curriculum supplement to support young children's mathematics learning: Results of a randomized controlled trial. A report to the CPB-PBS Ready To Learn Initiative*. Waltham, MA, and Menlo Park, CA.

Penuel, W. R., Bates, L., Gallagher, L. P., Pasnik, S., Llorente, C., Townsend, E., . . . & VanderBorght, M. (2012). Supplementing literacy instruction with a media-rich intervention: Results of a randomized controlled trial. *Early Childhood Research Quarterly, 27*(2), 115–127. doi: 10.1016/j.ecresq

Perlman, R. (1976). *Using computer technology to provide a creative learning environment for preschool children. Logo memo no 24*. Cambridge, MA: MIT Artificial Intelligence Laboratory.

Rideout, V. J. (2013). *Zero to eight: Children's media use in America 2013*. San Francisco, CA: Common Sense Media.

Steele, C. M. (1997). A threat in the air: How stereotypes shape intellectual identity and performance. *American Psychologist, 52*, 613–629.

Sullivan, A., Elkin, M., & Bers, M. (2015, June). KIBO robot demo: Engaging young children in programming and engineering. In *Proceedings of the 14th ACM International Conference on Interaction Design and Children,* Medford, MA.

Tucker, A., McCowan, D., Deek, F., Stephenson, C., Jones, J., & Verno, A. (2006). *A model curriculum for K–12 computer science: Report of the ACM K–12 task force curriculum committee* (2nd ed.). New York, NY: Association for Computing Machinery.

Unfried, A., Faber, M., & Wiebe, E. N. (2014, April). *Gender and student attitudes toward science, technology, engineering, and mathematics*. Paper presented at the annual meeting of the American Educational Research Association, Philadelphia, PA.

Weinburgh, M. (1995). Gender differences in student attitudes toward science: A meta analysis of the literature from 1970 to 1991. *Journal of Research in Science Teaching, 32*(4), 387–398.

Wyeth, P. (2008). How young children learn to program with sensor, action, and logic blocks. *International Journal of the Learning Sciences, 17*(4), 517–550.

Appendix E: Bibliography of Framework Research

Abelson, H., & diSessa. A. (1986). *Turtle geometry*. Cambridge, MA: MIT Press.

Achieve. (2015). *The role of learning progressions in competency-based pathways*. Retrieved from http://www.achieve.org/files/Achieve-LearningProgressionsinCBP.pdf

Aho, A. (2011, January). Computation and computational thinking. *ACM Ubiquity*. Retrieved from http://ubiquity.acm.org/article.cfm?id=1922682

Aivaloglou. E., & Hermans, F. (2016, September). How kids code and how we know: An exploratory study on the Scratch repository. In *Proceedings of the Twelfth Annual International Conference on International Computing Education Research* (pp. 53–61).

Akgün, L., & Özdemir, M. E. (2006). Students' understanding of the variable as general number and unknown: A case study. *The Teaching of Mathematics*, *9*(1), 45–51.

Anderson, N., Lankshear, C., Timms, C., & Courtney, L. (2008). "Because it's boring, irrelevant and I don't like computers": Why high school girls avoid professionally-oriented ICT subjects. *Computers & Education*, *50*(4), 1304–1318.

Arkansas Department of Education. (2016). *Computer science curriculum framework documents*. Retrieved from http://www.arkansased.gov/divisions/learning-services/curriculum-and-instruction/curriculum-framework-documents/computer-science

Baker, T. R., & White, S. H. (2003). The effects of GIS on students' attitudes, self-efficacy, and achievement in middle school science classrooms. *Journal of Geography*, *102*(6), 243–254.

Barr, D., Harrison, J., & Conery, L. (2011, March/April). Computational thinking: A digital age skill for everyone. *Learning and Leading with Technology*, 20–23.

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K–12: What is involved and what is the role of the computer science education community? *ACM Inroads*, *2*(1), 48–54.

Battista, M. T., & Clements, D. H. (1986). The effects of Logo and CAI problem-solving environments on problem-solving abilities and mathematics achievement. *Computers in Human Behavior, 2*(3), 183–193.

Baxter, G. P., & Glaser, R. (1997). *An approach to analyzing the cognitive complexity of science performance assessments* (CSE Technical Report 452). Los Angeles, CA: University of California, Center for the Study of Evaluation, National Center for Research on Evaluation, Standards, and Student Testing.

Beaumont-Walters, Y., & Soyibo, K. (2001). An analysis of high school students' performance on five integrated science process skills. *Research in Science & Technological Education*, *19*(2), 133–145.

Bell, T., Alexander, J., Freeman, I., & Grimley, M. (2009). Computer science unplugged: School students doing real computing without computers. *The New Zealand Journal of Applied Computing and Information Technology, 13*(1), 20–29.

Bender, E., Hubwieser, P., Schaper, N., Margaritis, M., Berges, M., Ohrndorf, L., . . . Schubert, S. (2015). Towards a competency model for teaching computer science. *Peabody Journal of Education, 90*(4), 519–532. doi: 10.1080/0161956X.2015.1068082

Ben-Zvi, D., & Arcavi, A. (2001). Junior high school students' construction of global views of data and data representations. *Educational Studies in Mathematics*, *45*, 35–65.

Berger, B., Daniels, N. M., & Yu, Y. W. (2016). Computational biology in the 21st century: Scaling with compressive algorithms. *Communications of the ACM, 59*(8), 72–80. doi: 10.1145/2957324

Bers, M. (2008). *Blocks to robots: Learning with technology in the early childhood classroom*. New York City, NY: Teachers College Press.

Bers, M. (2010). The TangibleK robotics program: Applied computational thinking for young children. *Early Childhood Research & Practice, 12*(2). Retrieved from http://ecrp.uiuc.edu/v12n2/bers.html

Bienkowski, M., Snow, E., Rutstein, D. W., & Grover, S. (2015). *Assessment design patterns for computational thinking practices in secondary computer science: A first look* (SRI technical report). Menlo Park, CA: SRI International. Retrieved from http://pact.sri.com/resources.html

Biggers, M., Brauer, A., & Yilmaz, T. (2008). Student perceptions of computer science: A retention study comparing graduating seniors vs. CS leavers. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education* (pp. 402–406), Portland, OR.

Blickenstaff, J.C. (2005). Women and science careers: Leaky pipeline or gender filter? *Gender and Education, 17*(4), 369–386. doi: 10.1080/09540250500145072

Boston Public Schools. (2016). *Computer science in BPS.* Retrieved from http://www.bostonpublicschools.org/domain/2054

Brennan, K., Balch, C., & Chung, M. (2014). *Creative computing*. Retrieved from http://scratched.gse.harvard.edu/guide/files/CreativeComputing20141015.pdf

Brennan, K., & Resnick, M. (2012). *Using artifact-based interviews to study the development of computational thinking in interactive media design*. Paper presented at the annual meeting of the American Educational Research Association, Vancouver, BC, Canada.

Brown, N. C. C., Sentance, S., Crick, T., & Humphreys, S. (2014). Restart: The resurgence of computer science in UK schools. *ACM Transactions on Computing Education, 14*(2), Article 9. doi: 10.1145/2602484

Buchanan, G., Farrant, S., Jones, M., Thimbleby, H., Marsden, G., & Pazzani, M. (2001). Improving mobile Internet usability. In *Proceedings of the 10th International World Wide Web Conference* (pp. 673–680), Hong Kong.

Buffardi, K., & Edwards, S. H. (2013, August). Effective and ineffective software testing behaviors by novice programmers. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research* (pp. 83–90).

Buffum, P. S., Martinez-Arocho, A. G., Frankosky, M. H., Rodriguez, F. J., Wiebe, E. N., & Boyer, K. E. (2014, March). CS Principles goes to middle school: Learning how to teach "big data." In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (pp. 151–156), Atlanta, GA. doi: 10.1145/2538862.2538949

Burns, K., & Polman, J. (2006). The impact of ubiquitous computing in the Internet age: How middle school teachers integrated wireless laptops in the initial stages of implementation. *Journal of Technology and Teacher Education, 14*(2), 363–385.

Buzzetto-More, N., Ukoha, O., & Rustagi, N. (2010). Unlocking the barriers to women and minorities in computer science and information systems studies: Results from a multi-methodolical study conducted at two minority serving institutions. *Journal of Information Technology Education, 9,* 115–131.

Campbell, P. F., & McCabe, G. P. (1984). Predicting the success of freshmen in a computer science major. *Communications of the ACM*, *27*(11), 1108–1113.

Carter, L. (2006). Why students with an apparent aptitude for computer science don't choose to major in computer science. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education* (pp. 27–31), Houston, TX.

Century, J., & Cassata, A. (in press). Measuring implementation and implementation research: Finding common ground on what, how and why. *Review of Research in Education, Centennial Edition.* American Educational Research Association.

Century, J., Cassata, A., Rudnick, M., & Freeman, C. (2012). Measuring enactment of innovations and the factors that affect implementation and sustainability: Moving toward common language and shared conceptual understanding. *Journal of Behavioral Health Services & Research*, *39*(4), 343–361.

Cernavskis, A. (2015, June 25). In San Francisco, computer science for all . . . soon. *The Hechinger Report*. Retrieved from http://hechingerreport.org/san-francisco-plans-to-be-first-large-district-to-bring-computer-science-to-all-grades/

Chou, J.-R., & Hsiao, S.-W. (2007). A usability study on human-computer interface for middle-aged learners. *Computers in Human Behavior, 23*, 2040–2063. doi: 10.1016/j.chb.2006.02.011

Clarke, V. A., & Teague, G. J. (1996). Characterizations of computing careers: Students and professionals disagree. *Computers in Education, 26*(4), 241–246.

Clements, D. H., & Gullo, D. F. (1974). Effects of computer programming on young children's cognition. *Journal of Educational Psychology, 76*(6), 1051–1058.

Cockburn, A., & Williams, L. (2000). The costs and benefits of pair programming. In *Proceedings of XP2000*, Sardinia, Italy.

College Board. (2016). *AP Computer Science Principles course and exam description.* New York, NY: Author. Retrieved from https://secure-media.collegeboard.org/digitalServices/pdf/ap/ap-computer-science-principles-course-and-exam-description.pdf

Common Sense Media. (2012). *Digital literacy and citizenship in a connected culture.*

Computational thinking in STEM: Skills taxonomy. (n.d.). Retrieved from http://gk12northwestern.wikispaces.com/file/view/CTSTEM_Skills_Taxonomy_4teachers.pdf

Computer Science Teachers Association & International Society for Technology in Education. (2011). Computational thinking teacher resources (2nd ed.). Retrieved from https://csta.acm.org/Curriculum/sub/CurrFiles/472.11CTTeacherResources_2ed-SP-vF.pdf

Computer Science Teachers Association Standards Task Force. (2011). *CSTA K–12 computer science standards, revised 2011.* New York, NY: Computer Science Teachers Association and Association for Computing Machinery. Retrieved from http://www.csteachers.org/resource/resmgr/Docs/Standards/CSTA_K-12_CSS.pdf

Computer Science Teachers Association Standards Task Force. (2016). *[Interim] CSTA K–12 computer science standards, revised 2016.* Springfield, OR: Computer Science Teachers Association and Association for Computing Machinery. Retrieved from http://www.csteachers.org/?page=CSTA_Standards

Computer Science Teachers Association Teacher Certification Task Force. (2008). *Ensuring exemplary teaching in an essential discipline: Addressing the crisis in computer science teacher certification.* New York, NY: Computer Science Teachers Association and the Association for Computing Machinery.

Cooper, S., Grover, S., Guzdial, M., & Simon, B. (2014). A future for computing education. *Communications of the ACM, 57*(11), 34–46.

CPALMS. (2016). *Science standards and access points.* Retrieved from http://www.cpalms.org/Downloads.aspx

Cotten, S. R., Shank, D. B., & Anderson, W. A. (2014). Gender, technology use and ownership, and media-based multitasking among middle school students. *Computers in Human Behavior, 35,* 99–106.

Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby, C., & Woollard, J. (2015). *Computational thinking: A guide for teachers.* Retrieved from the Computing at School website: http://computingatschool.org.uk/computationalthinking

Dasgupta, S., Hale, W., Monroy-Hernández, A., & Hill, B. M. (2016, February). Remixing as a pathway to computational thinking. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing* (pp. 1438–1449).

De Boulay, B. (1989). Some difficulties of learning to program. In E. Soloway and J. Spohrer (Eds.), *Studying the novice programmer*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Denner, J. (2011). What predicts middle school girls' interest in computing? *International Journal of Gender, Science and Technology 3*(1), 53–69.

Denner, J., Werner, L., Campe, S., & Ortiz, E. (2014). Pair programming: Under what conditions is it advantageous for middle school students? *Journal of Research on Technology in Education, 46*(3), 277–296.

Denner, J., Werner, L., & Ortiz, E. (2012). Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers & Education, 58,* 240–249.

Denning, P. J. (2013). The science in computer science. *Communications of the ACM, 56*(5), 35–38. doi: 10.1145/2447976.2447988

Denning, P. J., & Martell, C. (2015). *Great principles of computing.* Cambridge, MA: MIT Press.

DevTech Research Group at Tufts University. (2016). *The early childhood robotics network: Early childhood robotics curriculum.* Retrieved from http://tkroboticsnetwork.ning.com/page/robotics-curriculum

Di Vano, D., & Mirolo, C. (2011). "Computer science and nursery rhymes": A learning path for the middle school. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education* (pp.238–242), New York, NY

Dodds, Z., & Erlinger, M. (2013). MyCS: Building a middle-years CS curriculum. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education* (p. 330), New York, NY.

Dorling, M., & Browning, P. (2015). *Computing progression pathways KS1 (Y1) to KS3 (Y9) by topic.* Computing at School. Retrieved from http://community.computingatschool.org.uk/resources/1692

Dwyer, H., Hill, C., Carpenter, S., Harlow, D., & Franklin, D. (2014, March). Identifying elementary students' pre-instructional ability to develop algorithms and step-by-step instructions. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (pp. 511–516).

Elkin, M., Sullivan, A., & Bers, M. U. (2014). Implementing a robotics curriculum in an early childhood Montessori classroom. *Journal of Information Technology Education: Innovations in Practice, 13,* 153–169. Retrieved from http://www.jite.org/documents/Vol13/JITEv13IIPvp153-169Elkin882.pdf

England Department for Education. (2013, September 11). *National curriculum in England: Computing programmes of study.* Retrieved from https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study/national-curriculum-in-england-computing-programmes-of-study

Evers, V., & Day, D. (1997). The role of culture in interface acceptance. In S. Howard, J. Hammond, & G. Lindgaard (Eds.), *Proceedings of Human-Computer Interaction INTERACT'97* (pp. 260–267). Sydney, Australia: Chapman & Hall.

Fessakis, G., Gouli, E., & Mavroudi, E. (2013). Problem solving by 5–6 years old kindergarten children in a computer programming environment: A case study. *Computers & Education, 63*, 87–97.

Fields, D. A., Giang, M., and Kafai, Y. (2014). Programming in the wild: Trends in youth computational participation in the online Scratch community. In *Proceedings of the 9th Workshop in Primary and Secondary Computing Education* (pp. 2–11), New York, NY.

Fincher, S. (2015). What are we doing when we teach computing in schools? *Communications of the ACM, 58*(5), 24–26. doi: 10.1145/2742693

Fish, M. C., Gross, A. L., & Sanders, J. S. (1986). The effect of equity strategies on girls' computer usage in school. *Computers in Human Behavior, 2*, 127–134.

Forte, A., & Guzdial, M. (2004). Computers for communication, not calculation: Media as a motivation and context for learning. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences, Track 4, Volume 4* (p. 40096.1). Washington, DC: IEEE Computer Society. doi: 10.1109/HICSS.2004.1265259

Franklin, D., Conrad, P., Aldana, G., & Hough, S. (2011, March). Animal Tlatoque: Attracting middle school students to computing through culturally-relevant themes. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (pp. 453–458), Dallas, TX.

Franklin, D. F. (2015, February). Putting the computer science in computing education research. *Communications of the ACM, 58*(2), 34–36.

Franklin, D., Hill, C., Dwyer, H., Iveland, A., Hansen, A., & Harlow, D. (2016). Initialization in Scratch: Seeking knowledge transfer. In *Proceedings of the 47th ACM Symposium on Computing Science Education* (pp. 217–222), Memphis, TN.

Franklin, D., Hill, C., Dwyer, H., Iveland, A., Killina, A., & Harlow, D. (2015). Getting started in teaching and researching computer science in the elementary classroom. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (pp. 552–557), Kansas, City, MO.

Freischlad, S. (2007, November). Exploration module for understanding the functionality of the Internet in secondary education. In *Proceedings of the Seventh Baltic Sea Conference on Computing Education Research, Volume 88* (pp. 183–186). Australian Computer Society, Inc.

Friedman, B. (1996). Value-sensitive design. *Interactions, 3*(6), 16–23.

Friend, M., & Robert, C. (2013). Efficient egg drop contests: How middle school girls think about algorithmic efficiency. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research* (pp. 99–106).

Garfield, J. (1995). How students learn statistics. *International Statistical Review/Revue Internationale de Statistique*, *63*(1), 25–34.

Georgia Department of Education. (2016). *Information technology career cluster.* Retrieved from http://www.gadoe.org/Curriculum-Instruction-and-Assessment/CTAE/Pages/cluster-IT.aspx

Goldberg, D., Grunwald, D., Lewis, C., Feld, J., Donley, K., & Edbrooke, O. (2013). Addressing 21st century skills by embedding computer science in K–12 classes. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (pp. 637–638).

Goldberg, D. S., Grunwald, D., Lewis, C., Feld, J. A., & Hug, S. (2012). Engaging computer science in traditional education: The ECSITE project. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education* (pp. 351–356).

Goldschmidt, D., MacDonald, I., O'Rourke, J., & Milonovich, B. (2011). An interdisciplinary approach to injecting computer science into the K–12 classroom. *Journal of Computing Science in College, (26)*6, 78–85.

Good, C., Rattan, A., & Dweck, C. S. (2012). Why do women opt out? Sense of belonging and women's representation in mathematics. *Journal of Personality and Social Psychology*, *102*(4), 700–717.

Goode, J., & Chapman, G. (2013). *Exploring computer science: A high school curriculum exploring what computer science is and what it can do* (Version 5.0).

Grosslight, L., Unger, C., Jay, E., & Smith, C. L. (1991). Understanding models and their use in science: Conceptions of middle and high school students and experts. *Journal of Research in Science Teaching*, *28*(9), 799–822.

Grover, S. (2015). *"Systems of assessments" for deeper learning of computational thinking in K–12.* Paper presented at the annual meeting of the American Educational Research Association, Chicago, IL.

Grover, S., Cooper, S., & Pea, R. (2014). Assessing computational learning in K–12. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education* (pp. 57–62).

Grover, S., Jackiw, N., & Lundh, P. (2015). Thinking outside the box: Integrating dynamic mathematics to advance computational thinking for diverse student populations. Abstract. Retrieved from http://www.nsf.gov/awardsearch/showAward?AWD_ID=1543062

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher, 41*(1), 38–43.

Grover, S., & Pea, R. (2013b). Using a discourse-intensive pedagogy and Android's App Inventor for introducing computational concepts to middle school students. In *Proceedings of the 44th ACM Technical Symposium on Computer Science Education* (pp. 723–728), Denver, CO.

Grover, S., Pea, R., & Cooper, S. (2014). Remedying misperceptions of computer science among middle school students. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (pp. 343–348), Atlanta, GA.

Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education, 25*(2), 199–237.

Grover, S., Pea, R., & Cooper, S. (2016, March). Factors influencing computer science learning in middle school. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (pp. 552–557), Memphis, TN.

Grover, S., Rutstein, D., & Snow, E. (2016, March). "What is a computer?": What do secondary school students think? In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (pp. 564–569), Memphis, TN.

Guernsey, L. (2012). *Screen time: How electronic media from baby videos to educational software affects your young child.* Philadelphia, PA: Basic Books.

Guzdial, M. (2013a). Human-centered computing: A new degree for Licklider's world. *Communications of the ACM, 56*(5), 32–34. doi: 10.1145/2447976.2447987

Guzdial, M. (2013b). Exploring hypotheses about media computation. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research* (pp. 19–26).

Guzdial, M. (2016). *Learner-centered design of computing education: Research on computing for everyone*. Synthesis lectures on human-centered informatics. Morgan and Claypool.

Hanks, B. (2008). Problems encountered by novice pair programmers. *Journal on Educational Resources in Computing (JERIC), 7*(4), Article 2.

Hansen, A., Dwyer, H., Hill, C., Iveland, A., Martinez, T., Harlow, D., & Franklin, D. (2015). Interactive design by children: A construct map for programming. In *Proceedings of the 14th International Conference on Interaction Design and Children* (pp. 267–270).

Hansen, A., Iveland, A., Carlin, C., Harlow, D., & Franklin, D. (2016, June). User-centered design in block-based programming: Developmental and pedagogical considerations for children. In *Proceedings of the 15th International Conference on Interaction Design and Children* (pp. 147–156).

Hansen, A. K., Hansen, E. R., Dwyer, H. A., Harlow, D. B., & Franklin, D. (2016). Differentiating for diversity: Using universal design for learning in computer science education. In *Proceedings of the 47th ACM Technical Symposium on Computer Science Education* (pp. 376–381), Memphis, TN.

Harel, I., & Papert, S. (1990). Software design as a learning environment. *Interactive Learning Environments, 1*(1), 1–32.

Haspékian, M. (2003). Between arithmetic and algebra: A space for the spreadsheet? Contribution to an instrumental approach. In *Proceedings of the 3rd Conference of the European Society for Research in Mathematics Education*, Bellaria, Italy.

Herl, H. E., O'Neil, Jr., H. F., Chung, G. K. W. K., & Schacter, J. (1999). Reliability and validity of a computer-based knowledge mapping system to measure content understanding. *Computers in Human Behavior, 15*, 315–333.

Hoadley, C., Xu, H., Lee, J. J., & Rosson, M. B. (2010). Privacy as information access and illusory control: The case of the Facebook News Feed privacy outcry. *Electronic Commerce Research and Applications*, *9*(1), 50–60.

Hubwieser, P., Armoni, M., Brinda, T., Dagiene, V., Diethelm, I., Giannakos, M. N., . . . Schubert, S. (2011, June). Computer science/informatics in secondary education. In *Proceedings of the 16th Annual Conference Reports on Innovation and Technology in Computer Science Education—Working Group Reports* (pp. 19–38).

Hubwieser, P., Magenheim, J., Mühling, A., & Ruf, A. (2013, August). Towards a conceptualization of pedagogical content knowledge for computer science. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research* (pp. 1–8).

Idaho State Department of Education. (2016). *Idaho K–12 computer science standards.* Draft document.

Indiana Department of Education. (2016). *Science & computer science*. Retrieved from http://www.doe.in.gov/standards/science-computer-science

International Society for Technology in Education. (2012). *Computational thinking toolkit.* Retrieved from http://www.iste.org/learn/computational-thinking/ct-toolkit

International Society for Technology in Education. (2016). *ISTE standards for students*. Retrieved from https://www.iste.org/resources/product?id=3879&childProduct=3848

International Society for Technology in Education & Computer Science Teachers Association. (2011). *Operational definition of computational thinking for K–12 education.* Retrieved from https://csta.acm.org/Curriculum/sub/CurrFiles/CompThinkingFlyer.pdf

Isaacs, A., Binkowski, T. A., Franklin, D., Rich, K., Strickland, C., Moran, C., . . . Maa, W. (2016). Learning trajectories for integrating K–5 computer science and mathematics. In *2016 CISE/EHR principal investigator & community meeting for CS in STEM project description booklet* (p. 79). Retrieved from https://www.ncwit.org/sites/default/files/file_type/pi_book_compressed_2016.pdf

Israel, M., Pearson, J., Tapia, T., Wherfel, Q., & Reese, G. (2015). Supporting all learners in school-wide computational thinking: A cross case analysis. *Electronic Commerce Research and Applications, 82*, 263–279. doi: 10.1016/j.compedu.2014.11.022

Israel, M., Wherfel, Q., Pearson, J., Shehab, S., & Tapia, T. (2015). Empowering K–12 students with disabilities to learn computational thinking and computer programming. *TEACHING Exceptional Children, 48*(1), 45–53.

Jarman, S., & Bell, T. (2014, November). A game to teach network communication reliability problems and solutions. In *Proceedings of the 9th Workshop in Primary and Secondary Computing Education* (pp. 43–49).

Joint Task Force on Computing Curricula, Association for Computing Machinery, & IEEE Computer Society. (2013, December 20). *Computer science curricula 2013: Curriculum guidelines for undergraduate degree programs in computer science.* Association for Computing Machinery and IEEE.Retrieved from https://www.acm.org/education/CS2013-final-report.pdf

Kafai, Y. (2016). From computational thinking to computational participation in K–12 education. *Communications of the ACM, 59*(8), 26–27. doi: 10.1145/2955114

Kafai, Y. B., & Burke, Q. (2014). *Connected code: Why children need to learn programming.* Cambridge, MA: MIT Press.

Kafai, Y. B., & Burke, Q. (2015). Constructionist gaming: Understanding the benefits of making games for learning. *Educational Psychologist, 50*(4), 313–334.

Kafai, Y. B., Franke, M. L., Ching, C. C., & Shih, J. C. (1998). Game design as an interactive learning environment for fostering students' and teachers' mathematical inquiry. *International Journal of Computers for Mathematical Learning, 3*(2), 149–184. doi: 10.1023/A:1009777905226

Kay, A. (n.d.). Squeak Etoys, children, and learning. *Viewpoints Research Institute (VPRI Research Note RN-2005-001).* Retrieved from http://www.squeakland.org/resources/articles/article.jsp?id=1009

Kay, A. (2003). Afterward: Our human condition "from space." In B. J. Allen-Conn & K. Rose (Eds.), *Powerful ideas in the classroom: Using Squeak to enhance math and science learning* (pp. 73–79). Glendale, CA: Viewpoints Research Institute.

Kazakoff, E., & Bers, M. (2012). Programming in a robotics context in the kindergarten classroom: The impact on sequencing skills. *Journal of Educational Multimedia and Hypermedia, 21*(4), 371–391.

Kazakoff, E. R. (2015). Technology-based literacies for young children: Digital literacy through learning to code. In K. L. Heider & M. Renck Jalongo (Eds.), *Young children and families in the information age* (pp. 43–60). New York, NY: Springer.

Koppelman, H. (2007). Exercises as a tool for sharing pedagogical knowledge. In *Proceedings of the 12th Annual SIGCSE Conference in Innovation and Technology in Computer Science Education* (p. 361). doi: 10.1145/1268784.1268933

Koppelman, H. (2008, June). Pedagogical content knowledge and educational cases in computer science: An exploration. In *Proceeding of the Informing Science and IT Education Conference (InSITE)* (pp. 125–133). Varna, Bulgaria.

Kothiyal, A., Majumdar, R., Murthy, S., & Iyer, S. (2013, August). Effect of think-pair-share in a large CS1 class: 83% sustained engagement. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research* (pp. 137–144).

Kreitmayer, S., Rogers, Y., Laney, R., & Peake, S. (2012, May). From participatory to contributory simulations: Changing the game in the classroom. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 49–58).

Kuhn, A., McNally, B., Schmoll, S., Cahill, C., Lo, W. T., Quintana, C., & Delen, I. (2012, May). How students find, evaluate and utilize peer-collected annotated multimedia data in science inquiry with zydeco. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 3061–3070).

Kurland, D. M., & Pea, R. D. (1985). Children's mental models of recursive LOGO programs. *Journal of Educational Computing Research*, *1*(2), 235–243.

Ladner, R., & Israel, M. (2016). "For all" in "computer science for all." *Communications of the ACM, 59*(9), 26–28.

LeBlanc, M. D., & Dyer, B. D. (2004). Bioinformatics and computing curricula 2001: Why computer science is well positioned in a post-genomic world. *ACM SIGCSE Bulletin*, *36*(4), 64–68.

Lee, I. (2016). Reclaiming the roots of CT. *CSTA Voice: The Voice of K–12 Computer Science Education and Its Educators, 12*(1), 3–4. Retrieved from http://www.csteachers.org/resource/resmgr/Voice/csta_voice_03_2016.pdf

Lee, I., Martin, F., & Apone, K. (2014). Integrating computational thinking across the K–8 curriculum. *ACM Inroads, 5*(4), 64–71.

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., . . . Werner, L. (2011). Computational thinking for youth in practice. *ACM Inroads, 2*(1), 32–37.

Lee, M. J., & Ko, A. J. (2011). Personifying programming tool feedback improves novice programmers' learning. In *Proceedings of the Seventh International Workshop on Computing Education Research* (pp. 109–116), Providence, RI. doi: 10.1145/2016911.2016934

Leidner, D. E., & Kayworth, T. (2006). A review of culture in information systems research: Toward a theory of information technology culture conflict. *MIS Quarterly*, *30*(2), 357–399.

Leutenegger, S., & Edgington, J. (2007, March). A games first approach to teaching introductory programming. *ACM SIGCSE Bulletin 39*(1), 115–118.

Lewis, C. M. (2012). *Applications of out-of-domain knowledge in students' reasoning about computer program state* (Doctoral dissertation).

Lewis, C. M. (2014, July). Exploring variation in students' correct traces of linear recursion. In *Proceedings of the Tenth Annual Conference on International Computing Education Research* (pp. 67–74).

Lewis, C. M. (2016). You wouldn't know it from SIGCSE proceedings, but we don't only teach CS1 (Abstract Only). In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (p. 494).

Lewis, C. M., & Shah, N. (2012). Building upon and enriching grade four mathematics standards with programming curriculum. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (pp. 57–62).

Lewis, C. M., & Shah, N. (2015, July). How equity and inequity can emerge in pair programming. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research* (pp. 41–50).

Li, C., Dong, Z., Untch, R. H., & Chasteen, M. (2013). Engaging computer science students through gamification in an online social network based collaborative learning environment. *International Journal of Information and Education Technology, 3*(1), 72–77. doi: 10.7763/IJIET.2013.V3.237

Lishinski, A., Good, J., Sands, P., & Yadav, A. (2016, September). Methodological rigor and theoretical foundations of CS education research. In *Proceedings of the Twelfth Annual International Conference on International Computing Education Research* (pp. 161–169).

Lou, Y., Abrami, P.C., & d'Apollonia, S. (2001). Small group and individual learning with technology: A meta-analysis. *Review of Educational Research, 71*, 449–521.

Lowther, D. L., Bassoppo-Moyo, T., & Morrison, G. R. (1998). Moving from computer literate to technologically competent: The next educational reform. *Computers in Human Behavior, 14*(1), 93–109.

Lu, J. J., & Fletcher, G. H. L. (2009, March). Thinking about computational thinking. In *Proceedings of the 40th SIGCSE Technical Symposium on Computer Science Education* (pp. 260–264), Chattanooga, TN.

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K–12? *Computers in Human Behavior, 41*, 51–61.

Malik, S., & Agarwal, A. (2012). Use of multimedia as a new educational technology tool—A study. *International Journal of Information and Education Technology, 2*(5), 468–471. doi: 10.7763.IJIET.2012.V2.181

Manduca, C., & Mogk, D., (2002, April). *Using data in undergraduate science classrooms: Final report on an interdisciplinary workshop at Carleton College*. Northfield, MN: Science Education Resource Center, Carleton College. Retrieved from http://serc.carleton.edu/usingdata/report.html

Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Ronaldsson, L., & Settle, A. (2014, June). Computational thinking in K–9 education. In *Proceedings of the Working Group Reports of the 2014 Innovation & Technology in Computer Science Education Conference*, Uppsala, Sweden. doi: 10.1145/2713609.2713610

Marcus, B. (2015, August 12). The lack of diversity in tech is a cultural issue. *Forbes*. Retrieved from http://www.forbes.com/sites/bonniemarcus/2015/08/12/the-lack-of-diversity-in-tech-is-a-cultural-issue/#622c464a3577

Margolis, J., Estrella, R., Goode, J., Holme, J. J., & Nao, K. (2010). *Stuck in the shallow end: Education, race, and computing*. Cambridge, MA: MIT Press.

Margolis, J., Ryoo, J., Sandoval, C., Lee, C., Goode, J., & Chapman, G. (2012). Beyond access: Broadening participation in high school computer science. *ACM Inroads*, *3*(4), 72–78.

Mark, J., & DeLyser, L. (2016). CSNYC knowledge forum: Launching research and evaluation for computer science education, for every school and every student in New York City. Abstract. Retrieved from http://www.nsf.gov/awardsearch/showAward?AWD_ID=1637654

Maryland State Department of Education. (2005). *Maryland technology education state curriculum*. Retrieved from http://mdk12.msde.maryland.gov/instruction/curriculum/technology_education/vsc_technologyeducation_standards.pdf

Maryland State Department of Education. (2015). *Computer science*. Retrieved from http://archives.marylandpublicschools.org/MSDE/divisions/dccr/cs.html

Massachusetts Department of Elementary and Secondary Education. (2016, June). *2016 Massachusetts digital literacy and computer science (DLCS) curriculum framework*. Malden, MA: Author. Retrieved from http://www.doe.mass.edu/frameworks/dlcs.pdf

Matias, J. N., Dasgupta, S., & Hill, B. M. (2016, May). *Skill progression in Scratch revisited*. Paper presented at the Conference on Human Factors in Computing Systems, San Jose, CA.

Matuk, C., & King Chen, J. (2011, March). *WISE Ideas: A technology-enhanced curriculum to scaffold students' generating data, managing evidence, and reasoning about the seasons*. Teacher design focus group presented at the Cyberlearning Tools for STEM Education Conference, Berkeley, CA.

McCrickard, D. S., & Lewis, C. (2012). *Workshop on designing for cognitive limitations*. Paper presented at the Designing Interactive Systems Conference, Newcastle, UK.

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2010, August). Learning computer science concepts with Scratch. In *Proceedings of the Sixth International Workshop on Computing Education Research* (pp. 69–76).

Mesaroş, A. M., & Diethelm, I. (2012, November). Ways of planning lessons on the topic of networks and the Internet. In *Proceedings of the 7th Workshop in Primary and Secondary Computing Education* (pp. 70–73).

Milenkovic, L., Acquavita, T., & Kim, D. (2015). Investigating conceptual foundations for a transdisciplinary model integrating computer science into the elementary STEM curriculum. Abstract. Retrieved from http://www.nsf.gov/awardsearch/showAward?AWD_ID=1542842

Milner, S. (1973). *The effects of computer programming on performance in mathematics*. Paper presented at the annual meeting of the American Educational Research Association, New Orleans, LA.

Moore, T., Wick, M., & Peden, B. (1994). Assessing student's critical thinking skills and attitudes toward computer science. *ACM SIGSCE Bulletin, 26*(1), 263–267.

Morgan, C., Mariotti, M. A., & Maffei, L. (2009). Representation in computational environments: Epistemological and social distance. *International Journal of Computers for Mathematical Learning*, *14*(3), 241–263.

Morrison, B. B., Margulieux, L. E., & Guzdial, M. (2015, July). Subgoals, context, and worked examples in learning computing problem solving. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research* (pp. 21–29).

Moursund, D. (1983). *Introduction to computers in education for elementary and middle school teachers*. Eugene, OR: International Council for Computers in Education.

Moursund, D., & Ricketts, D. (2016). Computational thinking. *IAE-pedia.* Retrieved from http://iae-pedia.org/Computational_Thinking

National Association of State Directors of Career Technical Education Consortium & National Career Technical Education Foundation. (2012). *Common career technical core.* Silver Spring, MD: Authors.

National Governors Association Center for Best Practices & Council of Chief State School Officers. (2010). *Common core state standards.* Washington DC: Author.

National Research Council. (2007). *Taking science to school: Learning and teaching science in grades K–8.* Committee on Science Learning-Kindergarten Through Eighth Grade. R. A. Duschl, H. A. Schweingruber, & A. W. Shouse (Eds.). Board on Science Education, Center for Education. Division of Behavioral and Social Sciences and Education. Washington, DC: The National Academies Press.

National Research Council. (2010). *Report of a workshop on the scope and nature of computational thinking.* Washington, DC: The National Academies Press. Retrieved from http://www.nap.edu/catalog/12840.html

National Research Council. (2012). *A framework for K–12 science education: Practices, crosscutting concepts, and core ideas.* Committee on a Conceptual Framework for New K–12 Science Education Standards. Board on Science Education, Division of Behavioral and Social Sciences and Education. Washington, DC: The National Academies Press.

The New Zealand Curriculum Online. (2014). *Technology.* Retrieved from http://nzcurriculum.tki.org.nz/The-New-Zealand-Curriculum/Technology

Newell, A., & Simon, H. A. (1972). *Human problem solving.* Englewood Cliffs, NH: Prentice Hall.

Next Generation Science Standards Lead States. (2013). *Next generation science standards: For states, by state*s. Washington, DC: The National Academies Press.

North, A. S., & Noyes, J. M. (2002). Gender influences on children's computer attitudes and cognitions. *Computers in Human Behavior, 18*, 135–150.

Ohrndorf, L. (2015, July). Measuring knowledge of misconceptions in computer science education. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research* (pp. 269–270).

Ouyang, Y., Wolz, U., & Rodger, S. H. (2010, March). Effective delivery of computing curriculum in middle school: Challenges and solutions. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (pp. 327–328). Milwaukee, WI.

Palumbo, D. B. (1990). Programming language/problem-solving research: A review of relevant issues. *Review of Educational Research, 60*(1), 65–89.

Papert, S. (1971). *A computer laboratory for elementary schools.* Cambridge, MA: Massachusetts Institute of Technology.

Papert, S. (1980). *Mindstorms: Children, computers and powerful ideas.* New York, NY: Basic Books.

Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New Ideas in Psychology, 2*, 137–168.

Pea, R. D., Soloway, E., & Spohrer, J. C. (1987). The buggy path to the development of programming expertise. *Focus on Learning Problems in Mathematics, 9*, 5–30.

Pellegrino, J. W., & Hilton, M. L. (Eds.). (2012). *Education for life and work: Developing transferable knowledge and skills in the 21st century.* Washington, DC: The National Academies Press.

Perkins, D. N., & Salomon, G. (1988). Teaching for transfer. *Educational Leadership*, *46*(1), 22–32.

Perkovic, L., & Settle, A. (2009). *Computational thinking across the curriculum: A conceptual framework.* Retrieved from http://compthink.cs.depaul.edu/FinalFramework.pdf

Perkovic, L., Settle, A., Hwang, S., & Jones, J. (2010, June). A framework for computational thinking across the curriculum. In *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education* (pp. 123–127).

Pillars of cyber security. (n.d.). Retrieved from https://www.usna.edu/CyberCenter/si110/lec/pillarsCybSec/lec.html

Plant, E. A., Baylor, A. L., Doerr, C. E., & Rosenberg-Kima, R. B. (2009). Changing middle-school students' attitudes and performance regarding engineering with computer-based social models. *Computers & Education, 53,* 209–215.

Porter, L., Guzdial, M., McDowell, C., & Simon, B. (2013). Success in introductory programming: What works? *Communications of the ACM, 56*(8), 34–36. doi: 10.1145/24920072492020

Proulx, V. K. (1993, January). Computer science in elementary and secondary schools. In *Proceedings of the IFIP TC3/*WG3.1/ WG*3.5 Open Converence on Informatics and Changes in Learning.* Retrieved from http://www.ccs.neu.edu/home/vkp/Papers/Gmunden93.pdf

Repenning, A., Webb, D., & Ioannidou, A. (2010). Scalable game design and the development of a checklist for getting computational thinking into public schools. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (pp. 265–269).

Resnick, M., Ocko, S., & Papert, S. (1988). LEGO, Logo, and design. *Children's Environments Quarterly, 5*(4), 14–18.

Robertson, J., & Howells, C. (2008). Computer game design: Opportunities for successful learning. *Computers and Education, 50*, 559–578. doi: 10.1016/j.compedu.2007.09.020

Rodger, S. H., Brown, D., Hoyle, M., MacDonald, D., Marion, M., Onstwedder, E., . . . Ward, E. (2014, June). Weaving computing into all middle school disciplines. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education* (pp. 207–212), Uppsala, Sweden. doi: 10.1145/2591708.2591754

Rodger, S. H., Hayes, J., Lezin, G., Qin, H., Nelson, D., Tucker, R., . . . Slater, D. (2009). Engaging middle school teachers and students with Alice in a diverse set of subjects. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education* (pp. 271–275), Chattanooga, TN.

Rose, D., & Meyer, A. (2000). Universal design for learning. *Journal of Special Education Technology, 15*(1), 67–70.

Saeli, M., Perrenet, J., Jochems, W. M., & Zwaneveld, B. (2011). Teaching programming in secondary school: A pedagogical content knowledge perspective. *Informatics in Education*, *10*(1), 73–88.

Saeli, M., Perrenet, J., Jochems, W. M., & Zwaneveld, B. (2012). Programming: Teachers and pedagogical content knowledge in the Netherlands. *Informatics in Education*, *11*(1), 81–114.

Schacter, J., Herl, H. E., Chung, G. K. W. K., Dennis, R. A., & O'Neil, Jr., H. F. (1999). Computer-based performance assessments: A solution to the narrow measurement and reporting of problem solving. *Computers in Human Behavior, 15*, 403–418.

Schanzer, E., Fisler, K., Krishnamurthi, S., & Felleisen, M. (2015, February). Transferring skills at solving word problems from computing to algebra through Bootstrap. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (pp. 616–621).

Schanzer, E. T. (2015). *Algebraic functions, computer programming, and the challenge of transfer* (Doctoral dissertation).

Schofield, E., Erlinger, M., & Dodds, Z. (2014, March). MyCS: CS for middle-years students and their teachers. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (pp. 337–342), Atlanta, GA. doi: 10.1145/2538862.2538901

Schulte, C., & Knobelsdorf, M. (2007). Attitudes towards computer science—computing experiences as a starting point and barrier to computer science. In *Proceedings of the Third International Workshop on Computing Education Research* (pp. 27–38). doi: 10.1145/1288580.1288585

Schulz, S., & Pinkwart, N. (2015, November). Physical computing in STEM education. In *Proceedings of the Workshop in Primary and Secondary Computing Education* (pp. 134–135).

Seiter, L., & Foreman, B. (2013, August). Modeling the learning progressions of computational thinking of primary grade students. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research* (pp. 59–66).

Sengupta, P., Kinnebrew, J. S., Biswas, G., & Clark, D. (2013). Integrating computational thinking with K–12 science education: A theoretical framework. *Education and Information Technologies, 18*, 351–380.

Settle, A., Franke, B., Hansen, R., Spaltro, F., Jurisson, C., Rennert-May, C., & Wildeman, B. (2012, July). Infusing computational thinking into the middle- and high-school curriculum. In *Proceedings of the 17th Annual Conference on Innovation and Technology in Computer Science Education* (pp. 22–27), Haifa, Israel.

Sherin, B. L. (2001). A comparison of programming languages and algebraic notation as expressive languages for physics. *International Journal of Computers for Mathematical Learning, 6*(1), 1–61. doi: 10.1023/A:1011434026437

Shulman, L. S. (1986). Those who understand: Knowledge growth in teaching. *Educational Researcher 15*(2), 4–14. doi: 10.3102/0013189X015002004

Simon, Fincher, S., Robins, A., Baker, B., Box, I., Cutts, Q., . . . Tutty, J. (2006, January). Predictors of success in a first programming course. In *Proceedings of the Eighth Australasian Computing Education Conference* (pp. 189–196), Hobart, Tasmania, Australia.

Smith, J. L., Lewis, K. L., Hawthorne, L., & Hodges, S. D. (2013). When trying hard isn't natural: Women's belonging with and motivation for male-dominated STEM fields as a function of effort expenditure concerns. *Personality and Social Psychology Bulletin*, *39*(2), 131–143.

Snyder, L. (2010). *Six computational thinking practices*. Retrieved from https://csprinciples.cs.washington.edu/sixpractices.html

Soloway, E. (1986). Learning to program = learning to construct mechanisms and explanations. *Communications of the ACM, 29*(9), 850–858.

Sprague, P., & Schahczenski, C. (2002). Abstraction the key to CS1. *Journal of Computing Sciences in Colleges, 17*(3), 211–218.

Sudol, L. A., Stehlik, M., & Carver, S. (2009). *Mental models of data: A pilot study.* Paper presented at Ninth Baltic Sea Conference on Computing Education Research (Koli Calling 2009), Koli National Park, Finland.

Sullivan, G. (2014, May 29). Google statistics show Silicon Valley has a diversity problem. *The Washington Post.* Retrieved from https://www.washingtonpost.com/news/morning-mix/wp/2014/05/29/most-google-employees-are-white-men-where-are-allthewomen/

Syslo, M. M. (2015). From algorithmic to computational thinking: On the way for computing for all students. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*. Vilnius, Lithuania. doi: 10.1145/2729094.2742582

Taub, R., Armoni, M., & Ben-Ari, M. (2012). CS unplugged and middle-school students' views, attitudes, and intentions regarding CS. *ACM Transactions on Computing Education, 12*(2), Article 8.

Texas Education Agency. (2011). *Technology applications TEKS.* Retrieved from http://tea.texas.gov/Curriculum_and_Instructional_Programs/Curriculum_Standards/TEKS_Texas_Essential_Knowledge_and_Skills_(TEKS)_Review/Technology_Applications_TEKS/

Tucker, A., McCowan, D., Deek, F., Stephenson, C., Jones, J., & Verno, A. (2006). *A model curriculum for K–12 computer science: Report of the ACM K–12 task force curriculum committee* (2nd ed.). New York, NY: Association for Computing Machinery.

Twarek, B. (2015). *Pre-K to 12 computer science scope and sequence.* Retrieved from http://www.csinsf.org/curriculum.html

Valkenburg, P. M., & Peter, J. (2013). The differential susceptibility to media effects model. *Journal of Communication*, *63*, 221–243. doi: 10.1111/jcom.12024

Vekiri, I., & Chronaki, A. (2008). Gender issues in technology use: Perceived social support, computer self-efficacy and value beliefs, and computer use beyond school. *Computers & Education, 51*, 1392–1404.

Watson, W. E., Kumar, K., & Michaelsen, L. K. (1993). Cultural diversity's impact on interaction process and performance: Comparing homogeneous and diverse task groups. *Academy of Management Journal*, *36*(3), 590–602.

Wehmeyer, M. L. (2015). Framing the future self-determination. *Remedial and Special Education*, *36*(1), 20–23.

Wehmeyer, M. L., Shogren, K. A., Palmer, S. B., Williams-Diehm, K. L., Little, T. D., & Boulton, A. (2012). The impact of the self-determined learning model of instruction on student self-determination. *Exceptional Children, 78*(2), 135–153.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2015). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology, 25*(1), 127–147.

Weintrop, D., & Wilensky, U. (2015). Using commutative assessments to compare conceptual understanding in blocks-based and text-based programs. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research* (pp. 101–110), Omaha, NE.

Werner, L., Campe, S., & Denner, J. (2012). Children learning computer science concepts via Alice game-programming. In *Proceedings of the Special Interest Group in Computer Science Education* (pp. 215–220), Raleigh, NC.

Werner, L., & Jenner, D. (2009). Pair programming in middle school: What does it look like? *Journal of Research on Technology in Education, 42*(1), 29–49.

Werner, L., Denner, J., & Campe, S. (2012, February–March). The fairy performance assessment: Measuring computational thinking in middle school. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (pp. 215–220). doi: 10.1145/2157136.2157200

Werner, L., Denner, J., & Campe, S. (2014). Using computer game programming to teach computational thinking skills. In K. Schrier (Ed.), *Learning, education, and games* (Vol. 1, pp. 37–53). Pittsburgh, PA: ETC Press.

Werner, L., Denner, J., Campe, S., Ortiz, E., DeLay, D., Hartl, A. C., & Laursen, B. (2013). Pair programming for middle school students: Does friendship influence academic outcomes? In *Proceedings of the 44th ACM Technical Symposium on Computer Science Education* (pp. 421–426). doi: 10.1145/2445196.2445322

Whitley, Jr., B. E. (1997). Gender differences in computer-related attitudes and behavior: A meta-analysis. *Computers in Human Behavior, 13*(1), 1–22.

Wille, S. J., & Kim, D. (2015). Factors affecting high school student engagement in introductory computer science classes. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (pp. 675–675), New York, NY. doi: 10.1145/2676723.2691891

Wille, S. J., Pike, M., & Century, J. (2015). Bringing AP Computer Science Principles to students with learning disabilities and/or an ADHD: The hidden underrepresented group. Abstract. http://www.nsf.gov/awardsearch/showAward?AWD_ID=1542963.

Williams, L. A., & Kessler, R. R. (2000). All I really need to know about pair programming I learned in kindergarten. *Communications of the ACM, 43*(5), 108–114.

Williams, L., Kessler, R. R., Cunningham, W., & Jeffries, R. (2000). Strengthening the case for pair programming. *IEEE Software, 17*(4), 19–25.

Wing, J. M. (2006, March). Computational thinking. *Communications of the ACM, 49*(3), 33–35.

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society, 366*(1881), 3717–3725.

Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., & Korb, J. T. (2014). Computational thinking in elementary and secondary teacher education. *ACM Transactions on Computing Education, 14*(1), Article 5, 1–16.

Yardi, S., & Bruckman, A. (2007). What is computing?: Bridging the gap between teenagers' perceptions and graduate students' experiences. In *Proceedings of the Third International Workshop on Computing Education Research* (pp. 39–50). doi: 10.1145/1288580.1288586

Zare-ee, A., & Shekarey, A. (2010). The effects of social, familial, and personal factors on students' course selection in Iranian technical schools. *Procedia: Social and Behavioral Sciences, 9*, 295–298.

Appendix F: Frequently Asked Questions

Q. What is the framework?

A: The framework is a baseline, essential set of computer science concepts and practices. The focus of the framework is to illuminate powerful ideas in K–12 computer science. Each of the core concepts are delineated with expectations at four different grade-band endpoints: Grades 2, 5, 8, and 12. The practices are not delineated by explicit grade bands but instead provide a narrative describing each practice's progression from kindergarten to Grade 12.

Q: Why do we need a framework, and why is it important?

A: Computing is all around us in the modern world, yet many students do not understand how these technologies work. Interest in computer science is increasing, and K–12 education is eager to meet the demand. However, computer science is fairly new to K–12 education, and states, districts, schools, and teachers need guidance for an appropriate K–12 pathway in computer science.

Q: What is the vision of the framework?

A: The purpose is to create a high-level framework of computer science concepts and practices that will empower students to

- be informed citizens who can critically engage in public discussion on computer science-related topics;
- develop as learners, users, and creators of computer science knowledge and artifacts;
- better understand the role of computing in the world around them; and
- learn, perform, and express themselves in other subjects and interests.

Q: What is computer science?

A: Computer science is the study of computers and algorithmic processes, including their principles, design, implementation, and impact on society (Tucker, 2006, p. 2).

Q: Who is the framework for?

A: The K–12 Computer Science Framework is an initial step in a process that will inform state- and district-level decisions to introduce and improve computer science education. The framework was written for a variety of audiences with a wide range of backgrounds in computer science. The primary audiences of the framework are state policymakers and administrators, district policymakers and administrators, standards developers, curriculum developers, professional development providers, researchers, and current and new computer science educators in both formal and informal settings.

Q: Why is computer science important?

A: Computer science underpins many aspects of the modern world. The ubiquity of personal computing in our lives and our exponentially increasing reliance on all things related to technology have changed the fabric of society and day-to-day life. Unfortunately, K–12 students today have limited opportunity to learn about these computer science ideas and practices and to analyze how computing influences their daily lives.

Q: What are the core concepts and practices of the framework?

A:: The core concepts are categories that represent major content areas in the field of computer science. They represent specific areas of disciplinary importance rather than abstract, general ideas. The core practices are the behaviors that computationally literate students use to fully engage with the core concepts of computer science.

Core concepts

1. Computing Systems
2. Networks and the Internet
3. Data and Analysis
4. Algorithms and Programming
5. Impacts of Computing

Core practices

1. Fostering an Inclusive Computing Culture
2. Collaborating Around Computing
3. Recognizing and Defining Computational Problems
4. Developing and Using Abstractions
5. Creating Computational Artifacts
6. Testing and Refining Computational Artifacts
7. Communicating About Computing

Q: Who created the framework?

A: The steering committee for the framework consists of representatives from the following organizations: Association for Computing Machinery, Code.org, Computer Science Teachers Association, Cyber Innovation Center, and National Math + Science Initiative. The writing team was composed of representatives from participating states, school districts, K–12 educators, higher education faculty, and research and nonprofit organizations (you can see their biographies in **Appendix B: Biographies of Writers and Development Staff**). Leading researchers and representatives from organizations in computer science education served as advisors to the writers and the development staff. More than 100 computer science education practitioners and stakeholder organizations served as reviewers, with more than 530 reviews submitted.

Q: Were teachers involved?

A: Many of the writers were current teachers or had been teachers in the past. Their experience spanned kindergarten through Grade 12 and included a variety of subjects outside of computer science. Some of the advisors were teachers, many of the development staff were former teachers, and teachers were included in review periods and in focus groups during the development process.

Q: How were states involved in the development of the framework?

A: For each of the involved states, representatives from the state department of education and board of education attended the stakeholder convenings to provide feedback into the development of the framework. The ten states that participated in the launch of the framework's development were asked to nominate someone from their state to serve on the writing team and to convene a group to review the drafts of the framework in their state.

Q: How was the public involved in the creation of the framework?

A: Three public review periods were held during the development of the framework in 2016. Each lasted two to three weeks, and each was publicized widely to encourage the public to read the draft versions and submit feedback via an online form. The feedback that was received was read by the writers and development team, and common themes were addressed by the writers. You can read more in the **Development Process** chapter and see a summary of the public reviews in **Appendix A: Feedback and Revisions**.

Q: How was research used to inform the development of the framework?

A: The writing team considered the current literature on computer science education from the start of the writing process. After the second draft of the framework was complete, a systematic review of the literature related to the concepts and practices was completed. From this review, concept and practice statements were tweaked to align them with the current research in the field. You can read more in the **The Role of Research in the Development and Future of the Framework**.

Q: What is the relationship between the framework and standards?

A: The framework broadly delineates the concepts students should know and the practices students should exhibit, but it does not provide the level of detail of grade-by-grade standards, course objectives or descriptions, or lesson plans. Instead it serves as a comprehensive guide for the development of standards, curriculum, assessments, teacher education, and extracurricular programs. The framework is not a set of standards. States may use the framework to develop standards that will combine the concepts and practices into performance expectations that are clear, specific, and measurable.

Q: How is the framework different from national standards?

A: The framework statements are **not** standards. They are purposefully not as prescriptive or measurable as performance standards. They do not address individual grade-level granularity; instead, they address grade bands and describe how learning progresses from one grade band to another.

There are far fewer statements in the framework than in a standards document. The focus of the framework is to provide a minimum set of concepts and practices that describe baseline literacy in computer science that all students should have. An explicit goal of the framework is to show significance/application beyond computer science and significance for every citizen, not just computer science students.

The framework is intentionally designed for customization and will be freely available. It describes what students should learn using nontechnical prose that is easy to understand by a wide audience. States and districts should make the final decision on the documents they use when developing their own computer science standards.

The framework distinguishes between concepts and practices. A standards document should integrate these two dimensions into each standard.

You can read more about how the framework can inform the development of standards in the **Guidance for Standards Developers** chapter.

Q: What is the relationship with the Computer Science Teachers Association (CSTA) K–12 standards?

A: The K–12 Computer Science Framework served as one of many inputs into the interim 2016 revision of the CSTA K–12 computer science standards to ensure alignment and allow for the computer science education community to speak with a coherent voice about what K–12 students should know and be able to do. The co-chairs of the CSTA standards revision task force served as advisors to the framework, and half of the CSTA standards writers (including all of the CSTA lead writers) served as writers of the framework.

Q: Are future revisions of the framework planned?

A: It is anticipated that the framework will be revised in the future.

Q: How does the framework address computer literacy and digital citizenship?

A: The framework defines K–12 computer science, which is different from digital citizenship and computer literacy. Digital citizenship is defined as the norms of appropriate, responsible behavior with regard to the use of technology (Massachusetts Department of Elementary and Secondary Education, 2016). Computer literacy focuses on the use of existing technologies and computer programs like word processing and spreadsheets (National Center for Women & Information Technology).

Computer science, on the other hand, is about "the ability to create and adapt new technologies" and involves analyzing how computers work and how they affect us (National Center for Women & Information Technology, n.d., para.3). The K–12 Computer Science Framework is meant to complement the instruction of computer literacy and digital citizenship. Instruction in all three areas is important for all students.

Q: How does the framework address computational thinking?

A: Computational thinking refers to the thought processes involved in expressing solutions as computational steps or algorithms that can be carried out by a computer (Cuny, Snyder, & Wing, 2010; Aho, 2011; Lee, 2016). It is delineated in four of the seven computer science practices in the framework. Computer science offers a unique opportunity for students to develop computational thinking, but the opportunity to apply computational thinking extends beyond the context of computer science. Recent revisions to the 2016 International Society for Technology in Education Standards for Students are aligned with this definition of computational thinking. These documents support the shared vision that computational thinking is important for all students in all classes.

Q: How is the framework implemented in schools?

A: The framework can be used in a variety of ways. It can inform curriculum development, standards development, K–12 pathways, teacher preparation and professional development, and classroom assessment, among others. It can be implemented as standalone courses or integrated into other subject areas, particularly at the elementary and middle school levels. You can read more in the **Implementation Guidance** chapter.

Q: Where can I find the framework?

A: You can download a copy of the entire framework (all chapters, including guidance material) as a PDF file at k12cs.org. Just want the concepts and practices? Click on "Framework Statements" in the main menu to access the dropdown menu and select one of the three views. From each page, you can download a PDF file of the concepts and practices.

Q: If I want help using the framework, whom should I contact?

A: You can reach out to the development staff with your questions using the form found at k12cs.org.

References

Aho, A. V. (2011, January). Computation and computational thinking. *ACM Ubiquity, 1*, 1–8. .

Cuny, J., Snyder, L., & Wing, J. M. (2010). *Computational thinking: A definition*. Unpublished manuscript.

Lee, I. (2016). Reclaiming the roots of CT. *CSTA Voice: The Voice of K–12 Computer Science Education and Its Educators, 12*(1), 3–4. Retrieved from http://www.csteachers.org/resource/resmgr/Voice/csta_voice_03_2016.pdf

Massachusetts Department of Elementary and Secondary Education. (2016, June). *2016 Massachusetts digital literacy and computer science (DLCS) curriculum framework*. Malden, MA: Author. Retrieved from http://www.doe.mass.edu/frameworks/dlcs.pdf

National Center for Women & Information Technology. (n.d.). *Moving beyond computer literacy: Why schools should teach computer science.* Retrieved from https://www.ncwit.org/resources/moving-beyond-computer-literacy-why-schools-should-teach-computer-science/moving-beyond

Tucker, A., McCowan, D., Deek, F., Stephenson, C., Jones, J., & Verno, A. (2006). *A model curriculum for K–12 computer science: Report of the ACM K–12 task force curriculum committee* (2nd ed.). New York, NY: Association for Computing Machinery.

Photo Credits

Thank you to the students and teachers of the school districts who invited photographers to their classrooms and allowed their photographs to be included in this document.

Lincoln High School, Tacoma, Washington: cover, pages 4, 21, 23, 39, 41, 42, 46, 54, 85, 87, 143, 151, 168, 176, 202, and 219.

John Muir Elementary School, Seattle, Washington: pages 12, 24, 31, 55, 57, 65, 67, 69, 70, 134, 181, 183, 199, 201, 208, 211, and 215.

Mount View Elementary School, White Center, Washington: pages 7, 9, 16, 53, 123, 125, 134, 136, 145, 147, 205, and 229.

We also thank the organizations who graciously shared photographs of students engaging in computer science, as well as the students, parents, and teachers in the images.

DevTech Research Group: pages 194, 196, 272, and 273.

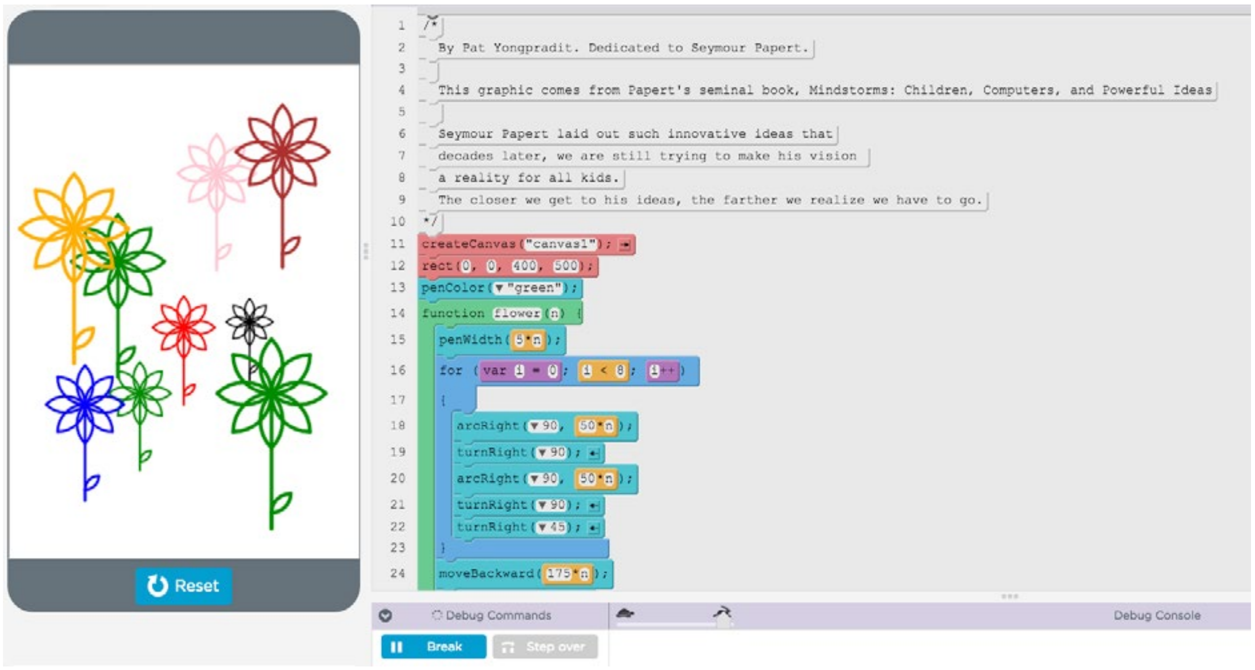
AccessCS10K from the University of Washington: page 33.

WCTE Cookeville, TN: pages 187 and 270.

WHRO Norfolk, VA: pages 187 and 270.

On the cover

The program displayed on the monitor was inspired by an anecdote in a book that has influenced many computer science educators, Seymour Papert's *Mindstorms* (1980). In this anecdote, Papert describes a hypothetical situation in which students write a program to draw a garden of flowers. The anecdote illustrates how playing with a few powerful ideas can lead to beautiful results. Papert's powerful ideas inspired a generation of educators who continue to work to make his vision a reality. These flowers have grown in the fertile ground that Papert prepared. Seymour Papert passed away on July 31, 2016. The closer we get to his ideas, the farther we realize we have to go.


```
/*
  By Pat Yongpradit. Dedicated to Seymour Papert.

  This graphic comes from Papert's seminal book, Mindstorms: Children, Computers, and Powerful Ideas

  Seymour Papert laid out such innovative ideas that
  decades later, we are still trying to make his vision
  a reality for all kids.
  The closer we get to his ideas, the farther we realize we have to go.
*/
createCanvas("canvas1");
rect(0, 0, 400, 500);
penColor("green");
function flower(n) {
  penWidth(5*n);
  for (var i = 0; i < 8; i++)
  {
    arcRight(90, 50*n);
    turnRight(90);
    arcRight(90, 50*n);
    turnRight(90);
    turnRight(45);
  }
  moveBackward(175*n);
```