Rhode Island K-12
Computer Science Education Standards

Endorsed by the Rhode Island Council on Elementary and Secondary Education
May 2018

CS4RI

Rhode Island
STEAM Center
At Rhode Island College

RIDE Rhode Island Department of Education

Table of Contents

Acknowledgements

The development of the Computer Science Education Standards was a statewide collaboration with contributions from a diverse group of Rhode Island stakeholders. I express my deepest appreciation to the Advisory Committee that gave up one Saturday a month for almost a year to review, evaluate, revise, and write the new standards. Their extraordinary dedication, enthusiasm, and thoughtfulness made the standards a reality. Special thanks go to Chris Allen, Kathi Fisler, Tim Henry, Joe Mazzone, Ilona Miko, and Vic Fay Wolfe for providing team leadership as we progressed. A very heartfelt thank you to Kathi Fisler and Ilona Miko for going above and beyond their responsibilities by devoting extra time and effort through extensive review, editing, discussion, and support.

I acknowledge, with much appreciation, the Review Team that looked at the draft standards with fresh eyes, and provided insightful comments and suggestions. Members of the public also contributed time and attention to reviewing the draft standards, and I thank them for invaluable feedback. I express my gratitude to the experts in the cybersecurity field and specialists in digital/media/instructional literacy who critiqued relevant sections of the standards.

A special note of thanks to Pat Yongpradit (Code.org), Peter McLaren (Next Generation Science Standards Writing Team Member), John Bilotta (Rhode Island Society of Technology Educators), and Tommy Gober and Kevin Nolten (Cyber Innovation Center) for their guidance, support, and encouragement.

The development of the Computer Science Education Standards for Rhode Island would have been impossible without the support of the van Beuren Charitable Foundation, and especially senior program officer Deborah S. Linnell.

Last but not least, many thanks to CS4RI and the Rhode Island Department of Education (RIDE) for helping us achieve our goal of developing comprehensive computer science education standards for Rhode Island.

The excitement and eagerness with which everyone approached this project is testament to the belief that all Rhode Island students deserve the best education for future success.

Carol M. Giuriceo, Ph.D.
Chair, Computer Science Education Standards Advisory Committee

Our Vision for Rhode Island

Our students, including those historically underrepresented, understand the value, influence, and relevance of computer science education. We believe that increased use and mastery of computational thinking through the grade levels builds human capacity, and allows students to become informed users, as well as active creators, of technology. Students shall thoughtfully and ethically approach personal and societal challenges and participate in finding solutions to local, regional, and global issues.

Our educators collaborate within communities of professional practice as computer science becomes the multi-disciplinary bridge across school districts. We believe that an engaged citizenry emerges from a strong focus on essential life and career skills, problem-solving abilities, and lifelong commitment to learning, which positions Rhode Island as a leader in technology and a premier innovative center in the United States.

Introduction

In January 2017, the Rhode Island STEAM Center organized and convened a statewide Computer Science (CS) Education Advisory Committee, with funding from the van Beuren Charitable Foundation, with the goal of creating CS education standards for Rhode Island. The impetus for this work was the current momentum surrounding the implementation of CS education in K-12 through the Computer Science for Rhode Island (CS4RI) initiative and the recent national emphasis on computer science education.

In March 2017, the CS Education Standards Advisory Committee, composed of Rhode Islanders from across the state, met to begin work on developing and aligning with the nationally-recognized K-12 Computer Science Framework (released October 2017), the Computer Science Teachers Association (CSTA) Computer Science Standards (draft standards November 2016; final standards released July 2017), and CS standards work in other states.

The Advisory Committee represented a broad range of expertise. We included elementary, middle, and high school teachers, district coordinators and administrators, higher education faculty, and industry professionals. Some members had computer science expertise; others were pedagogy experts and understood the value and use of academic standards. All served pro bono. Committee meetings occurred one Saturday a month through December 2017. During this time, we reviewed existing standards, evaluated practices, and identified core concepts.

We chose to adapt rather than adopt, the CSTA K-12 Standards because we wanted to create standards that retained the rigorous and collaborative work of the CSTA yet also related to the needs of Rhode Island. Our adaptations include:

- reorganizing the standards into concepts that we believe more accurately describe our focus and create logical progressions without too much overlap
- forming a new Digital Literacy concept focused on the use of computing devices, recognizing its fit alongside the current Information Literacy standards (recently revised) and Media Literacy standards (in development)
- forming a new Cybersecurity concept and recognizing its increasing global relevance, as well as Rhode Island's growing and economically-relevant cybersecurity sector

Throughout the process, we focused on creating pathways that set realistic expectations for all students and can be implemented in a sustainable way in Rhode Island. They do not represent a comprehensive list of all topics within computer science.

Most of all, we kept equity at the forefront of our discussions. We believe that increased use and mastery of computational thinking through the grade levels builds human capacity.

Advisory Committee

- Chris Allen, NBCT, Fourth Grade Teacher, Greenbush Elementary School, West Warwick Public Schools
- Jenny Chan-Remka, Ed.D., Assistant Superintendent, Woonsocket Education Department
- Michelle Conary, Computer Literacy Instructor, Chariho Middle School, Chariho Regional School District
- Jane L. Daly, Assistant Superintendent of Schools, Chariho Regional School District
- Vic Fay-Wolfe, Ph.D., Computer Science, University of Rhode Island
- Kathi Fisler, Ph.D., Research Professor, Computer Science, Brown University/Co-Director, Bootstrap
- Carol M. Giuriceo, Ph.D., Director, Rhode Island STEAM Center @ Rhode Island College
- Lenora E. Goodwin, Consulting Teacher, Teacher Retention and Induction Network (T.R.A.I.N.), Providence Public Schools
- Timothy Henry, Ph.D., Professor, IT Graduate Director, New England Institute of Technology
- Dominic Herard, Mathematics & Computer Science Teacher, Times Squared STEM Academy, Providence
- Verda Jones, Business & Technology Instructor, Shea Senior High School, Pawtucket School District
- Ramarao Koppaka, Staff Vice President, Principal Enterprise Architect, FM Global
- Linda Larsen, Director of Education Outreach & Workforce Development, Southeastern New England Defense Industry Alliance (SENEDIA)
- Bryan Lucas, Computer Science/Literacy Teacher, Chariho Middle School, Chariho Regional School District
- Joe Mazzone, Secretary, CSTA-RI/Career and Technical Education Instructor, William M. Davies Jr. Career and Technical High School, Lincoln
- Ilona Miko, Ph.D., MikoArtScience Consulting
- Ryan Mullen, Coordinator of Teaching & Learning, Warwick Public Schools
- Elizabeth (Liz) Patterson, Computer Science Teacher, Portsmouth High School, Portsmouth School Department
- Janet Prichard, Ph.D., Professor, Information Systems and Analytics, Bryant University
- Cmdr. Joseph E. Santos, Military Professor, U.S. Naval War College, Newport

Reviewers

Review Team

- Amanda Bagley, Second Grade Education Teacher, Greenbush Elementary School, West Warwick Public Schools
- John Bilotta, Executive Director, Rhode Island Society of Technology Educators (RISTE)
- Joe Devine, Partner & CTO, Bridge Technical Talent, LLC
- Howard L. Dooley, Jr., Project Manager, Rhode Island Technology Enhanced Sciences and Computing (RITES +C), University of Rhode Island
- Donald Gregory, Education Specialist, Providence Public Library
- Linda A. Jzyk, Grant Specialist, Rhode Island College Foundation, Former Science and Technology Specialist, Rhode Island Department of Education (RIDE)
- Tom Kowalczyk, Founder, KMRM,LLC
- Theresa Moore, President, T-Time Productions
- Diane Sanna, Assistant Superintendent, Bristol Warren Regional School District
- John Smithers, CEO, Tech Collective
- Holly Walsh, Digital Learning Specialist, Office of College and Career Readiness, Rhode Island Department of Education (RIDE)

Specialists

Cybersecurity

- Jason Albuquerque, C/CISO,CGCIO, Chief Information Security Officer, Carousel Industries
- Brig Gen Kimberly A. Baumann, Ph.D., Assistant Adjutant General, Rhode Island National Guard
- Simon A. Cousins, Principal Client Specialist, FM Global
- Richard Siedzik, Director of Information Security and Planning/ISO, Bryant University

Digital Literacy

- Renee Hobbs, Ph.D., Professor, Department of Communication Studies; Co-Director, Graduate Certificate in Digital Literacy, Harrington School of Communication and Media, University of Rhode Island
- Mary H. Moen, Ph.D., Assistant Professor, Graduate School of Library and Information Studies, University of Rhode Island
- Zoey Wang, Graduate Assistant, Rhode Island STEAM Center @ Rhode Island College

Public Review: Their Feedback

An open invitation was extended to teachers, principals, district administrators, superintendents, industry professionals, and other stakeholders to submit comments on the draft standards. There feedback provided valuable input that greatly enhanced the content of the standards.

Computer Science & Relevance to Rhode Island

In January 2016, the Metropolitan Policy Program at Brookings, along with Battelle Technology Partnership Practice (now TEConomy Partners, LLC) and with support from Monitor Deloitte, Deloitte Consulting LLP released Rhode Island Innovates: A Competitive Strategy for the Ocean State, a detailed economic assessment with recommendations for Rhode Island's economic development. Brookings and its partners engaged in a six-month inquiry with private- and public-sector stakeholders across the state to assess Rhode Island's present situation and competitive position, and to provide an action plan for strategy development.

The report identified CS science as a core competency in Rhode Island with areas of focus in data sciences, robotics, cybersecurity, and algorithms. According to Brookings, a core competency "represent[s] zones of endeavor where a place has the ability to grow. Core competencies indicate where there is a critical mass of expertise and creative activity across product development and process improvements that has the potential to generate new intellectual property and startups . . . core competencies highlight where a state's firms and research institutions have the capacity not only to advance new research discoveries but also to apply them, mobilize talent, and create good jobs." The report indicated that over 3,800 jobs posted online in 2013 in Rhode Island required knowledge of at least one programming language.

Unfortunately, the Brookings inquiry also found that student engagement with CS was low, with many of the state's schools only offering a basic computer literacy class as a graduation requirement, rather than a more rigorous and comprehensive CS course. During the 2014-2015 academic year, only 72 Rhode Island students took the AP CS exam. Acknowledging the need for sustainable solutions, recommendations included incorporating CS into the PK-12 curriculum through changing technology graduation requirements and public/private partnerships that work to implement CS courses and professional development.

To meet this need, the Computer Science for Rhode Island initiative, or CS4RI was created to bring CS learning opportunities to all Rhode Island schools. National and local programs from Microsoft, Project Lead the Way, and Code.org, to the University of Rhode Island and Brown University, are currently serving as content providers, offering professional development and established curricula to schools across the state. The CS4RI initiative brings computer education to the forefront of the discussion, as well as needed resources to jumpstart CS incorporation in K-12 education.

Conversations with educators during the first months of CS4RI implementation indicated that educators would welcome guidelines that assist in the development of computer science pathways. Identifying achievement outcomes for students in different grades would allow educators to feel confident that they were teaching what students need to know. In January 2017, the Rhode Island STEAM Center @ Rhode Island College received funding from the van Beuren Charitable Foundation to develop Computer Science Education Standards for Rhode Island.

Alignment with National Efforts

The Computer Science (CS) Education Standards process began at a critical time in K-12 computer science education.

- In October 2016, the **K-12 Computer Science (CS) Framework** was released. The Framework, a national effort led by the Association for Computing Machinery (ACM), Code.org, Computer Science Teachers Association (CSTA), Cyber Innovation Center (CIC), and the National Math + Science Initiative (Steering Committee) defined conceptual guidelines for states and districts to create a K-12 pathway in CS. Participants in the development of the Framework included writers, advisors, and researchers who represented associations (such as the International Society for Technology Education [ISTE]), industry (such as Microsoft, Google, Apple), states, school districts, higher education, and K-12.
- In July 2017, the Computer Science Teachers Association released their revised K-12 Computer Science Standards, which aligned with the K-12 CS Framework. These standards describe learning objectives designed to provide the foundation for a complete computer science curriculum at the K-12 level.
- Both the Framework and the CSTA Standards are based on current professional research and practice in computer science education.

Organizations and Key Documents Referenced

K-12 Computer Science Framework

Our standards reflect the recommendations of 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. The K-12 Compute Science Framework is endorsed by leading industry and educational organizations, as well as K-12, higher education, and research leaders in the field of computer science education. To find more information, including a full list of supporters, visit k12cs.org.

2017 CSTA K-12 Computer Science Standards

The Advisory Committee used the 2017 CSTA K-12 Computer Science Standards as a foundation for our standards but modifications were made to reflect the education environment in Rhode Island. The CSTA Standards are licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0) license. To find more information, visit https://www.csteachers.org/page/standards.

2016 Massachusetts Digital Literacy and Computer Science (DLCS) Curriculum Framework

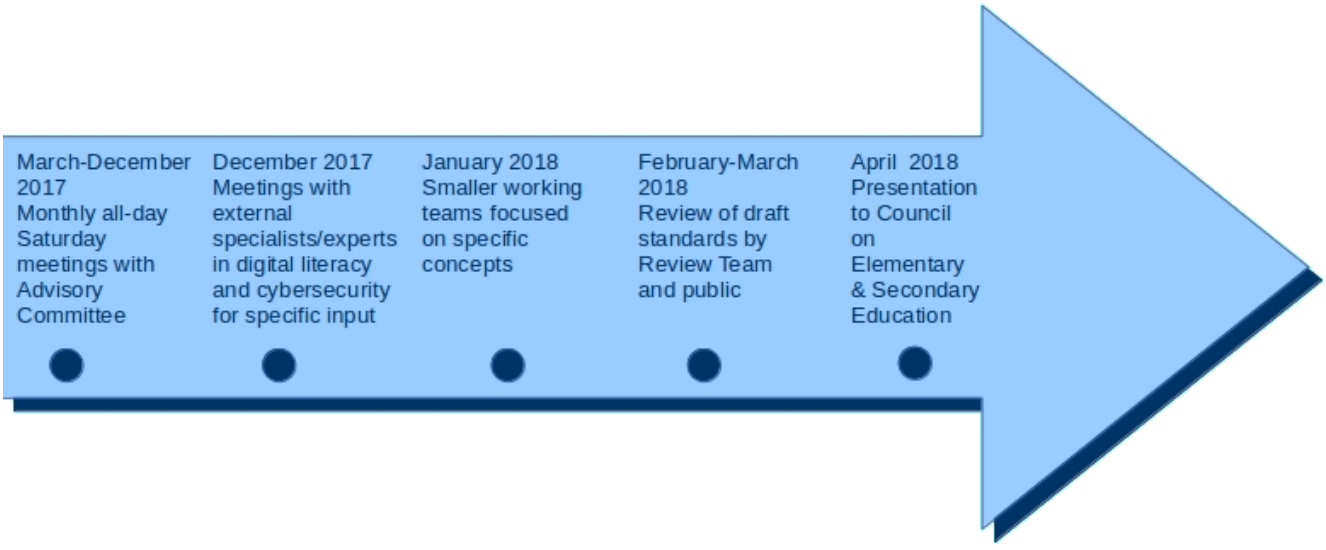
The Advisory Committee reviewed the 2016 Massachusetts Digital Literacy and Computer Science (DLCS) Curriculum Framework developed by the Massachusetts Department of Elementary and Secondary Education with a focus on the Digital Tools and Collaboration strand. To find more information, visit http://www.doe.mass.edu/frameworks/dlcs.pdf.

Process & Timeline

The Computer Science (CS) Education Standards Committee met monthly for day-long sessions from March 2017 to December 2017. Smaller committees convened more frequently during January and February 2018 for targeted discussions. Our process included reviewing the K-12 CS Framework and existing CS standards in other states. The Committee identified working Core Concepts and Sub-Concepts after the initial review. Smaller working groups with members representing different sectors formed to focus on specific concepts. Review of the CSTA CS K-12 Standards began. As work progressed, the Committee decided to combine two of the existing Concepts into one and add two new Concepts. The Committee followed a similar process with the Sub-Concepts.

Other stakeholders also helped to inform the standards. Although the Committee used existing standards in cybersecurity and digital literacy as starting points, we reached out to cybersecurity experts who offered suggestions and recommendations so the standards were comprehensive but not overly technical. Additionally, the Committee was aware of the overlap among digital literacy, media literacy, and instructional literacy, and met with specialists to discuss how to include digital literacy in the CS standards without duplication.

The Review Team, composed of Rhode Islanders from across the state, served as the reviewers for the draft standards. They evaluated the draft standards using a checklist developed by the K-12 CS Framework developers. Criteria included focus/manageability, equity, coherence/progression, clarity/accessibility, and measurability, among others.



Our Guiding Principles

The following Guiding Principles helped establish our aspirational vision and informed the development of K-12 Computer Science education standards for Rhode Island.	
Broaden Participation & Equity	All students regardless of age, race, ethnicity, gender, socioeconomic status, special needs, English proficiency, or any other demographic will have the opportunity to participate in computer science.The content and practices of the standards will be accessible to all.
Stimulate Learning & Curiosity	The standards at all grade levels will connect to appropriate real world challenges as a means to motivate and empower, promote individual growth, and spark a desire for life-long learning.
Build Connections Across Disciplines	Computer science will complement other disciplines and build upon and develop student knowledge.,The standards will connect with practices and concepts from the Common Core State Standards (CCSS) and the Next Generation Science Standards (NGSS) to promote learning across disciplines.
Encourage Workforce/Economic Development	Students will have the skills, practices, and knowledge to participate in a world that is increasingly influenced and shaped by technological advancements.,The standards will help to prepare students who can adapt and prosper under constantly changing conditions.
Support Teachers	The standards will identify focused learning progressions and multi-tier teaching approaches that meet the needs of all learners.
Inform with Current Research	The standards will be based on current professional research and practice in computer science education and pedagogy.

Equity in Computer Science Education

The Rhode Island Computer Science Education Standards Advisory Committee believes that equity and broadening participation must be at the forefront of the computer science initiative to ensure that all Rhode lsland students benefit. We strongly agree with the position identified in the K-12 Computer Science Framework (2016) which states:

> 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. . . . equity, inclusion, and diversity are critical factors in all aspects of computer science.(pp.23, 26)[1]

We constantly returned to this issue throughout the development of the standards. We worked to ensure equity is embedded in the standards themselves, the descriptions, and the accompanying suggested activities. Additionally, standards can be met without computing devices or with a limited amount of available hardware so implementation is possible for all schools.

1. K-12 Computer Science Framework. (2016). Retrieved from http://www.k12cs.org

Computational Thinking

> Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. . . . This kind of thinking will be part of the skill set of, not only other scientists, but of everyone else. Ubiquitous computing is to today as computational thinking is to tomorrow. Ubiquitous computing was yesterday's dream that become today's reality; computational thinking is tomorrow's reality.
>
> – Jeannette Wing, March 2006
> Communications of the ACM, 49(3), 33-35.

Computational thinking is central to the standards and a necessary skill for participation in today's society. It can be applied broadly to solving complex problems in other disciplines and can be taught across the K-12 curriculum.[1]

1. Computational Thinking for a Computational World. (2017). Retrieved from http://digitalpromise.org/wp-content/uploads/2017/12/dp-comp-thinking-v1r5.pdf

Standards

Standards represent pathways that are realistic expectations for all students. They identify the knowledge, practices, and skills in computer science that all students should know and be able to do at each level in their education. They serve as specific performance measures and are used as reference points for planning and teaching, including but not limited to, the development of curriculum frameworks, curricula, lesson plans, instruction, professional development, and assessment.

The standards are written to be aspirational – they represent the concepts and practices that all students need to master. They are designed to inform, encourage, and drive a sustainable computer science education program, and were developed to be cognitively appropriate for each grade band. Careful attention was paid to word choice in the standards to ensure measurability.

Grade Bands

The decision to adopt and use the grade bands identified in the CSTA K-12 Standards document – K-2, 3-5, 6-8, 9-12 – allows for increased flexibility for implementation in schools. Although the CSTA separated grades 9-12 into two levels – 9-10, 11-12 – with the 11-12 level designed for students enrolled in more rigorous courses, we decided that it was appropriate to extend the 9-10 level to 9-12 at this time since our goal focused on standards for ALL students.

Implementation of Standards

The Computer Science (CS) Education Standards are designed for all students K-12 in Rhode Island regardless of career aspirations. They represent the knowledge and skills that all students need to effectively participate and be productive in today's society.

At this time, adoption of the CS Education Standards by school districts is not mandatory. However, the response to the CS4RI initiative in connecting content providers with local schools to reduce barriers and provide quality CS education and professional development has been overwhelmingly positive at all grade levels.

CS4RI will support implementation of the CS Education Standards in school districts through a four-pronged approach:

1. All curricula and professional development offered by CS content providers in the CS4RI matrix will be aligned with the new Rhode Island standards. The new Memorandums of Understanding include this requirement.
2. The *SCRIPT – School CSforALL Resource & Implementation Planning Tool* – will be offered in Summer 2018 to all school districts to serve as a framework and platform to guide district staff in the creation of implementation plans based on the needs and goals of individual districts.
3. CS4RI will be working closely with the Computer Science Teachers Association – Rhode Island to provide resources and support through communities of practice.
4. CS4RI will be developing additional resources and supplementary materials to support CS Education Standards adoption in K-12 education.

Core Concepts and Sub-Concepts

The Core Concepts represent specific areas of disciplinary importance in computer science. The Sub-Concepts represent specific ideas within each concept.

Computational Thinking & Programming	• Algorithms • Variables • Data Structures & Data Types • Control Structures • Modularity • Computational Design
Computing Systems & Networks	• Human-Computer Interaction • Hardware & Software • Troubleshooting • Networks & the Internet
Cybersecurity	• Risks • Safeguards • Response
Data & Analysis	• Collection, Visualization, & Transformation • Inference & Models • Storage
Digital Literacy	• Creation & Use • Searching Digital Information • Understanding Software Tools
Responsible Computing in Society	• Culture • Safety, Law, & Ethics • Social Interactions

Each Core Concept and Sub-Concept is described in more detail on the following pages. The descriptions were adopted from the K-12 CS Framework. Certain changes and additions were made when necessary.

Computational Thinking & Programming

Overview: Computational thinking involves problem solving that requires the logical analysis of data. It serves as a fundamental skill for all students, and can be applied to complex problems across disciplines. These skills empower people to communicate with the world in new ways and solve compelling problems. Creating 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	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, and 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	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.
Data Structure & Data Types	Data structures store and organize data within a computer program. Data types classify data by attributes. In early grades, students learn to model and identify real-world examples of data. As they progress, they organize and create programs to process those data.
Control Structures	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	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.
Computational Design	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 decision, which involve user constraints, efficiency, ethics and testing.

Computing Systems & Networks

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.

Additionally, 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.

Human-Computer Interaction	Many everyday objects contain computational components that both 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 & Software	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	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 & the Internet	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 systems 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: Cybersecurity includes practices, processes, technologies, and other protective measures that are designed to protect against unwanted, unauthorized, or illegal access to or use of data, through onsite or remote devices, programs, and/or networks. As more information becomes digitized, both proactive and adaptive approaches to securing data become essential to meet the frequent and continually-evolving cybersecurity risks.

Risks	Being online or connected to a network has become part of a daily routine in personal, school and work environments. Students have ubiquitous access to information but are also exposed to common threats, scams, and fraud. In the early grades, students learn to identify and detect activity that may be monitoring and compromising their information. As they progress, students learn about social engineering, privacy concerns, and personal responsibility.
Safeguards	Transmitting information securely across networks requires appropriate protection that will mitigate or contain the impact, or even prevent the cybersecurity event. Safeguards include limiting access, using targeted processes and procedures, maintaining security software, and continuous monitoring of activity. In early grades, students learn how to protect their personal information. As they progress, students learn increasingly complex ways and tools used to protect information sent across networks and the trade-offs when selecting and implementing cybersecurity strategies.
Response	Implementing appropriate measures in response to a cybersecurity event requires an awareness of and a suitable reaction to the threat. Responses include pre-event planning, adoption and maintenance, threat containment, structure reporting and communications protocols, root cause analysis, and continuous process improvement. In the early grades, students learn when to report suspicious activity. As they progress, students learn how to take appropriate action on both personal and organizational levels.

Data & 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 are collected, analyzed and stored to better understand our social structures, geography, health and environment, and to make more accurate predictions about them.

Collection, Visualization, & Transformation	Data are collected with both computational and non-computational tools and processes. In early grades, students learn how data about themselves and their environment are collected and used. As they progress, students learn the effects of collecting data with computational and automated tools. Data are 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 those to others.
Inference & Models	Data science is one example where computer science serves many fields. Computer science and other sciences use data to make inferences, theories, or predictions based upon 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.
Storage	Core functions of computers are storing, representing, and retrieving data. In early grades, students learn how data are stored on computers. As they progress, students learn how to evaluate different storage methods, including the tradeoffs associated with those methods.

Digital Literacy

Overview: Digital literacy refers to the ability to leverage software technology to create, share, and modify artifacts, as well as search over digital information. This literacy includes understanding the benefits and implications of software tool use while accessing digital information and collaborating on digital artifacts. Digital literacy is a multifaceted concept that extends beyond skills-based activities and incorporates both cognitive and technical skills.

Creation & Use	Software tools are used to create and edit artifacts as well as locate and retrieve information. In early grades, students learn how to perform common operations using local, networked, or online tools. As they progress, students learn how to collaborate using software tools, and make informed decisions according to purpose and need.
Searching Digital Information	Locating, retrieving, and organizing relevant information includes being able to search for information in different ways. In early grades, students conduct basic and multi-criteria searches to find information in digital resources. As they progress, students learn to expand to multiple formats and databases, and synthesize search results to answer a complex question or solve a problem.
Understanding Software Tools	Humans interact with software tools to perform tasks. In early grades, students begin by understanding what software can do and how to explain this to others. As they progress, they learn about how software tools perform computations to incorporate multiple functions, and how software can be customized depending on who is using it.

Responsible Computing in Society

Overview: Computing affects many aspects of our world in both positive and negative ways, and at local and global scales. Individuals and communities influence computing through both 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 computing technology, including its impact on equity and access to computing.

Culture	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.
Safety, Law, & Ethics	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.
Social Interactions	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.

Practices

We adopted the practices that the K-12 CS Framework developed, and that are used in the CSTA K-12 Standards. These practices describe the behavior and ways of thinking that computationally-literate students use to engage in Core Concepts.

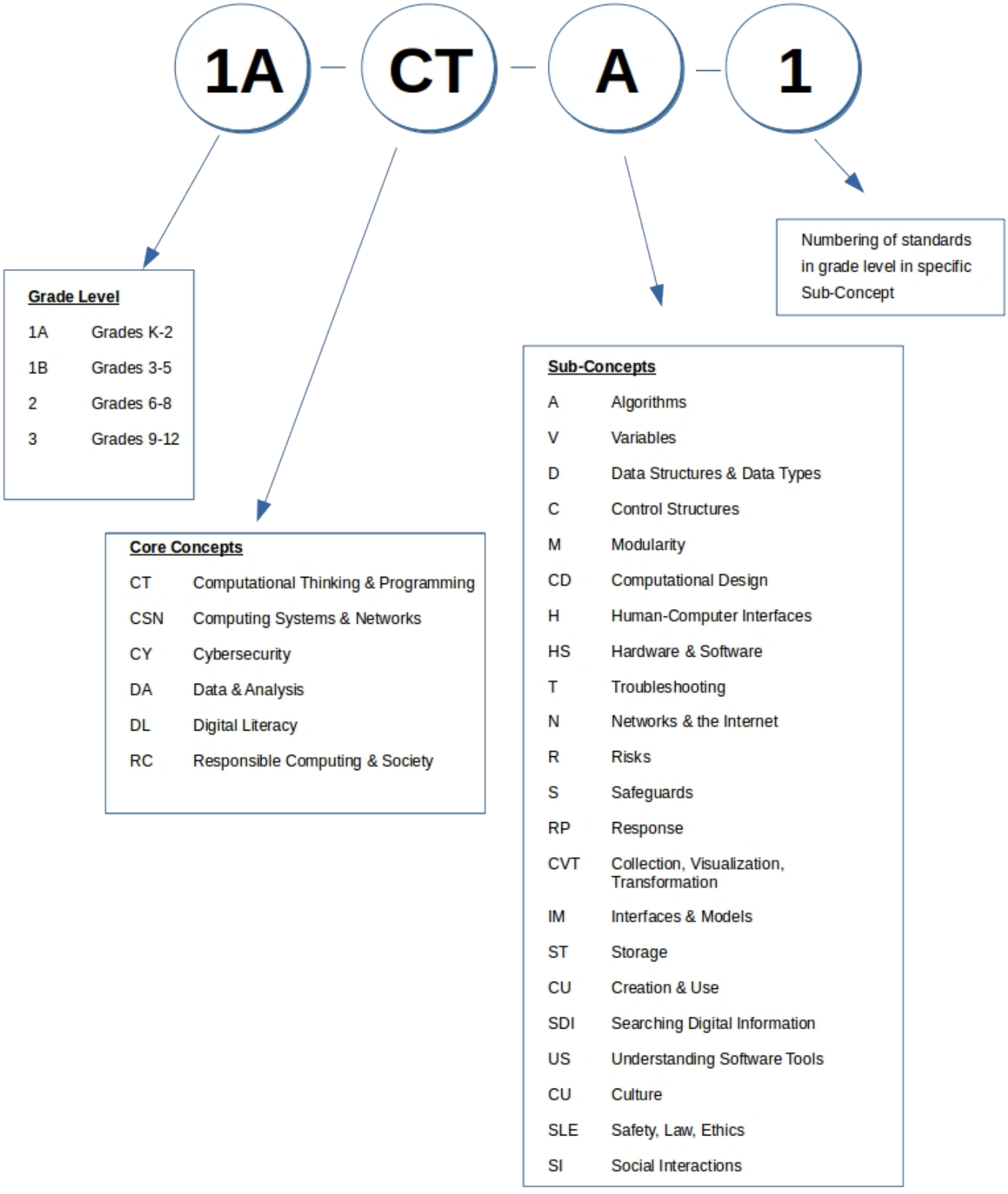
The Advisory Committee added an eighth practice – *Using Technology Appropriately* – which describes the necessary behavior and ways of thinking that support the *Cybersecurity* and *Digital Literacy* Core Concepts.

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. 1.1 Include the unique perspectives of others and reflect on one's own perspectives when designing and developing computational products. 1.2 Address,the needs of diverse users during the design process to produce artifacts with broad accessibility and usability. 1.3 Employ self- and peer-advocacy to address bias in interactions, product design, and development methods.
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. 2.1 Cultivate working relationships with individuals possessing diverse perspectives, skills, and personalities. 2.2 Create team norms, expectations, and equitable workloads to increase efficiency and effectiveness. 2.3 Solicit and incorporate feedback from, and provide constructive feedback to team members and other stakeholders. 2.4 Evaluate and select technological tools that can be used to collaborate on a project.

Practice 3 ***Recognizing & 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. 3.1 Identify complex, interdisciplinary, real-world problems that can be solved computationally. 3.2 Decompose complex real-world problems into manageable sub-problems that could integrate existing solutions or procedures. 3.3 Evaluate whether it is appropriate and feasible to solve a problem computationally.
Practice 4 ***Developing & 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. 4.1 Extract common features from a set of interrelated processes or complex phenomena. 4.2 Evaluate existing technological functionalities and incorporate them into new designs. 4.3 Create modules and develop points of interaction that can apply to multiple situations and reduce complexity. 4.4 Model phenomena and processes and simulate systems to understand and evaluate potential outcomes.
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. 5.1 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. 5.2 Create a computational artifact for practical intent, personal expression, or to address a societal issue. 5.3 Modify an existing artifact to improve or customize it.

Practice 6 ***Testing & 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. 6.1 Systematically test computational artifacts by considering all scenarios and using test cases. 6.2 Identify and fix errors using a systematic process. 6.3 Evaluate and refine a computational artifact multiple times to enhance its performance, reliability, usability, and accessibility.
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. 7.1 Select, organize, and interpret large data sets from multiple sources to support a claim. 7.2 Describe, justify, and document computational processes and solutions using appropriate terminology consistent with the intended audience and purpose. 7.3 Articulate ideas responsibly by observing intellectual property rights and giving appropriate attribution.
Practice 8 ***Using Technology Appropriately***	**Overview:** Today's technology-focused society requires more than just an understanding of how to use technology, but a working knowledge of what is appropriate, responsible, and safe behavior in a digital world. In computer science, understanding extends to the use of: hardware and software; applications such as email; the Internet and smaller home/school/business networks; and the use of onsite and offsite storage. Additionally, appropriate use includes onsite and remote access. 8.1 Follow certain protocols when using technology. 8.2 Identify and address risks and/or unintended consequences associated with technological tools by considering all scenarios. 8.3 Evaluate technological tools systematically through the use of select criteria based on the requirements of the task and the capacity of the system.

Reading the Standards



Grade Band	Identifier	COMPUTER SCIENCE EDUCATION STANDARD	CORE CONCEPT	Sub-Concept	Practice(s)
K-2	1A-CT-A-1	Model daily processes by creating and following algorithms to complete tasks.	**Computational Thinking & Programming**	Algorithms	Developing & Using Abstractions
3-5	1B-CT-A-1	Compare and refine multiple algorithms for the same task and determine which is more appropriate to complete the task.	**Computational Thinking & Programming**	Algorithms	• Recognizing & Defining Computational Problems • Testing & Refining Computational Artifacts
6-8	2-CT-A-1	Use diagrams and/or pseudocode to plan, analyze, solve and/or code complex problems as algorithms.	**Computational Thinking & Programming**	Algorithms	Developing & Using Abstractions
9-12	3-CT-A-1	Create computational artifacts that use algorithms to solve computational problems by leveraging prior knowledge and personal interests.	**Computational Thinking & Programming**	Algorithms	Creating Computational Artifacts
K-2	1A-CT-V-1	Model real-world data and how it is stored.	**Computational Thinking & Programming**	Variables	Creating Computational Artifacts
3-5	1B-CT-V-1	Create programs that use variables	**Computational Thinking & Programming**	Variables	Creating Computational Artifacts
6-8	2-CT-V-1	Create clearly named variables that represent different data. Perform operations on data stored in variables.	**Computational Thinking & Programming**	Variables	Creating Computational Artifacts
9-12	3-CT-V-1	Explain the role of a variable within a program, and the scope in which its name and value can be used.	**Computational Thinking & Programming**	Variables	Developing & Using Abstractions

K-2	1A-CT-D-1	Model real-world objects and/or processes that can be represented by various types of data.	**Computational Thinking & Programming**	Data Structures and Data Types	Developing & Using Abstractions
3-5	1B-CT-D-1	Identify real-world examples of data structures and data types.	**Computational Thinking & Programming**	Data Structures and Data Types	Recognizing & Defining Computational Problems
6-8	2-CT-D-1	Organize data into an appropriate data structure in a program.	**Computational Thinking & Programming**	Data Structures and Data Types	Creating Computational Artifacts
9-12	3-CT-D-1	Create a program that processes a collection of data.	**Computational Thinking & Programming**	Data Structures and Data Types	Creating Computational Artifacts
K-2	1A-CT-C-1	Develop simple programs with sequences and simple repetitions	**Computational Thinking & Programming**	Control Structures	Creating Computational Artifacts
3-5	1B-CT-C-1	Create programs that combine sequences, loops, conditionals, and/or events.	**Computational Thinking & Programming**	Control Structures	Creating Computational Artifacts
6-8	2-CT-C-1	Design programs that combine control structures, including nested loops and compound conditionals.	**Computational Thinking & Programming**	Control Structures	Creating Computational Artifacts
9-12	3-CT-C-1	Create and justify the selection of specific control structures when tradeoffs involve code organization, readability, and program performance and explain the benefits and drawbacks of choices made.	**Computational Thinking & Programming**	Control Structures	Creating Computational Artifacts
K-2	1A-CT-M-1	Decompose a task into a set of smaller tasks.	**Computational Thinking & Programming**	Modularity	Recognizing & Defining Computational Problems

3-5	1B-CT-M-1	Continually decompose problems into smaller subtasks until each subtask is a manageable set of basic operations.	**Computational Thinking & Programming**	Modularity	Recognizing & Defining Computational Problems
3-5	1B-CT-M-2	Create computational artifacts by incorporating existing modules into one's own work to solve a problem.	**Computational Thinking & Programming**	Modularity	• Developing & Using Abstractions • Creating Computational Artifacts
6-8	2-CT-M-1	Decompose computational problems to facilitate the design and implementation of programs.	**Computational Thinking & Programming**	Modularity	• Recognizing & Defining Computational Problems • Creating Computational Artifacts
6-8	2-CT-M-2	Create procedures with parameters to organize code and make it easier to reuse.	**Computational Thinking & Programming**	Modularity	Developing & Using Abstractions
9-12	3-CT-M-1	Identify existing computational artifacts that can be used for the subtasks of a decomposed problem.	**Computational Thinking & Programming**	Modularity	Recognizing & Defining Computational Problems
9-12	3-CT-M-2	Create computational artifacts by incorporating pre-defined procedures, self-defined procedures and external artifacts.	**Computational Thinking & Programming**	Modularity	Creating Computational Artifacts
K-2	1A-CT-CD-1	Develop a plan that describes what a computational artifact should look like and how it should perform.	**Computational Thinking & Programming**	Computational Design	• Creating Computational Artifacts • Communicating About Computing

K-2	1A-CT-CD-2	Identify a task that includes sequences and simple loops	**Computational Thinking & Programming**	Computational Design	Testing & Refining Computational Artifacts
3-5	1B-CT-CD-1	Use an iterative process to plan the development of a computational artifact by including others' perspectives and considering user preferences.	**Computational Thinking & Programming**	Computational Design	• Fostering An Inclusive Computing Culture • Creating Computational Artifacts
3-5	1B-CT-CD-2	Debug errors in an algorithm or program that includes sequences and simple loops.	**Computational Thinking & Programming**	Computational Design	Testing & Refining Computational Artifacts
3-5	1B-CT-CD-3	Describe steps taken and choices made during the process of creating a computational artifact.	**Computational Thinking & Programming**	Computational Design	Communicating About Computing
6-8	2-CT-CD-1	Seek and incorporate feedback from team members and users to refine a solution that meets user needs.	**Computational Thinking & Programming**	Computational Design	• Fostering An Inclusive Computing Culture • Collaborating Around Computing
6-8	2-CT-CD-2	Test and debug a program to ensure it runs as intended.	**Computational Thinking & Programming**	Computational Design	Testing & Refining Computational Artifacts
6-8	2-CT-CD-3	Describe choices made during development of computational artifacts.	**Computational Thinking & Programming**	Computational Design	Communicating About Computing
9-12	3-CT-CD-1	Systematically design and implement computational artifacts for targeted audiences by incorporating feedback from users.	**Computational Thinking & Programming**	Computational Design	Creating Computational Artifacts

9-12	3-CT-CD-2	Systematically test and refine programs using a range of test cases.	**Computational Thinking & Programming**	Computational Design	Testing & Refining Computational Artifacts
9-12	3-CT-CD-3	Document computational artifacts in order to make them easier to follow, test, and debug.	**Computational Thinking & Programming**	Computational Design	Communicating About Computing
K-2	1A-CSN-H-1	Identify the inputs and outputs of a computer system.	**Computing Systems & Networks**	Human-Computer Interfaces	Communicating About Computing
3-5	1B-CSN-H-1	Describe how people interact with the various parts of computing systems to accomplish tasks.	**Computing Systems & Networks**	Human-Computer Interfaces	Communicating About Computing
6-8	2-CSN-H-1	Identify improvements to the design of computing devices, based on an analysis of how users interact with the devices.	**Computing Systems & Networks**	Human-Computer Interfaces	Fostering An Inclusive Computing Culture
9-12	3-CSN-H-1	Analyze a computing system and explain how abstractions simplify the underlying implementation details embedded in everyday objects .	**Computing Systems & Networks**	Human-Computer Interfaces	Developing & Using Abstractions
K-2	1A-CSN-HS-1	Use appropriate terminology in identifying and describing the function of common physical components of computing systems (hardware).	**Computing Systems & Networks**	Hardware and Software	Communicating About Computing
3-5	1B-CSN-HS-1	Model how computer hardware and software work together as a system to accomplish tasks.	**Computing Systems & Networks**	Hardware and Software	Developing & Using Abstractions

6-8	2-CSN-HS-1	Design projects that combine hardware and software components to collect and use data to perform a function.	**Computing Systems & Networks**	Hardware and Software	Creating Computational Artifacts
9-12	3-CSN-HS-1	Compare levels of abstraction and interactions between application software, system software, and hardware layers.	**Computing Systems & Networks**	Hardware and Software	Developing & Using Abstractions
K-2	1A-CSN-T-1	Describe basic hardware and software problems using appropriate terminology.	**Computing Systems & Networks**	Troubleshooting	• Testing & Refining Computational Artifacts • Communicating About Computing
3-5	1B-CSN-T-1	Determine potential solutions to solve simple hardware and software problems using common troubleshooting strategies.	**Computing Systems & Networks**	Troubleshooting	Testing & Refining Computational Artifacts
6-8	2-CSN-T-1	Identify and fix problems with computing devices and their components using a systematic troubleshooting method or guide.	**Computing Systems & Networks**	Troubleshooting	Testing & Refining Computational Artifacts
9-12	3-CSN-T-1	Develop and communicate troubleshooting strategies others can use to identify and fix errors.	**Computing Systems & Networks**	Troubleshooting	Testing & Refining Computational Artifacts
K-2	1A-CSN-N-1	Describe the Internet as a place to share and find information.	**Computing Systems & Networks**	Networks and the Internet	Communicating About Computing

3-5	1B-CSN-N-1	Model how information is broken down into smaller pieces of data, transmitted as packets through multiple devices over networks and the Internet, and reassembled at the destination.	**Computing Systems & Networks**	Networks and the Internet	Developing & Using Abstractions
6-8	2-CSN-N-1	Model the role of protocols in transmitting data across networks and the Internet.	**Computing Systems & Networks**	Networks and the Internet	Developing & Using Abstractions
9-12	3-CSN-N-1	Identify the various elements of a network and describe how they function and interact to transfer information.	**Computing Systems & Networks**	Networks and the Internet	Communicating About Computing
K-2	1A-CY-R-1	Keep login and personal information private, and log off of devices appropriately.	**Cybersecurity**	Risks	Using Technology Appropriately
3-5	1B-CY-R-1	Describe the risks of sharing personal information, on websites or other public forums.	**Cybersecurity**	Risks	Using Technology Appropriately
3-5	1B-CY-R-2	Describe ways personal information can be obtained digitally.	**Cybersecurity**	Risks	Using Technology Appropriately
3-5	1B-CY-R-3	Describe the risks of others using one's personal resources or devices.	**Cybersecurity**	Risks	Using Technology Appropriately
6-8	2-CY-R-1	Describe tradeoffs between allowing information to be public and keeping information private and secure.	**Cybersecurity**	Risks	Using Technology Appropriately

6-8	2-CY-R-2	Describe social engineering attacks and the potential risks associated with them.	**Cybersecurity**	Risks	Using Technology Appropriately
6-8	2-CY-R-3	Describe risks of using free and open services.	**Cybersecurity**	Risks	Using Technology Appropriately
9-12	3-CY-R-1	Explain the privacy concerns related to the collection and generation of data through automated processes that may not be evident to users.	**Cybersecurity**	Risks	Using Technology Appropriately
9-12	3-CY-R-2	Analyze an existing or proposed application to identify the potential ways it could be used to obtain sensitive information.	**Cybersecurity**	Risks	• Recognizing & Defining Computational Problems • Using Technology Appropriately
9-12	3-CY-R-3	Explain how the digital security of an organization may be affected by the actions of its employees.	**Cybersecurity**	Risks	Using Technology Appropriately
K-2	1A-CY-S-1	Recognize basic digital security features.	**Cybersecurity**	Safeguards	Using Technology Appropriately
3-5	1B-CY-S-1	Explain individual actions that protect personal electronic information and devices.	**Cybersecurity**	Safeguards	Using Technology Appropriately
6-8	2-CY-S-1	Explain physical and digital security measures that protect electronic information.	**Cybersecurity**	Safeguards	Using Technology Appropriately
6-8	2-CY-S-2	Demonstrate how multiple methods of encryption provide secure transmission of information.	**Cybersecurity**	Safeguards	Using Technology Appropriately

9-12	3-CY-S-1	Recommend security measures to address various scenarios based on factors such as efficiency, feasibility, and ethical impacts.	**Cybersecurity**	Safeguards	Using Technology Appropriately
9-12	3-CY-S-2	Explain tradeoffs when selecting and implementing cybersecurity recommendations.	**Cybersecurity**	Safeguards	Using Technology Appropriately
K-2	1A-CY-RP-1	Identify situations with applications and devices that should be reported to a responsible adult.	**Cybersecurity**	Response	Using Technology Appropriately
3-5	1B-CY-RP-1	Identify and describe unusual data or behaviors of applications and devices that should be reported to a responsible adult.	**Cybersecurity**	Response	Using Technology Appropriately
6-8	2-CY-RP-1	Describe which actions to take and not to take when an application or device reports a problem or behaves unexpectedly.	**Cybersecurity**	Response	Using Technology Appropriately
9-12	3-CY-RP-1	Describe the appropriate actions to take in response to detected security breaches.	**Cybersecurity**	Response	Using Technology Appropriately
K-2	1A-DA-CVT-1	Collect and present the same data in multiple formats.	**Data & Analysis**	Collection, Visualization, Transformation	• Developing & Using Abstractions • Communicating About Computing

3-5	1B-DA-CVT-1	Organize and present collected data to highlight relationships and support a claim.	**Data & Analysis**	Collection, Visualization, Transformation	Developing & Using Abstractions Communicating About Computing
6-8	2-DA-CVT-1	Collect data using computational tools or online sources and transform the data to make it more useful and reliable.	**Data & Analysis**	Collection, Visualization, Transformation	Testing & Refining Computational Artifacts
9-12	3-DA-CVT-1	Select appropriate data-collection tools and presentation techniques for different types of data.	**Data & Analysis**	Collection, Visualization, Transformation	• Developing & Using Abstraction • Communicating About Computing
K-2	1A-DA-IM-1	Identify and describe patterns in data presentations, such as charts or graphs, to make predictions.	**Data & Analysis**	Inferences and Models	Developing & Using Abstractions
3-5	1B-DA-IM-1	Use data to highlight or propose cause-and-effect relationships, predict outcomes, or communicate an idea.	**Data & Analysis**	Inferences and Models	• Creating Computational Artifacts • Communicating About Computing
6-8	2-DA-IM-1	Create and refine computational models based on generated or gathered data.	**Data & Analysis**	Inferences and Models	• Developing & Using Abstraction • Creating Computational Artifacts • Testing & Refining Computational Artifacts

6-8	2-DA-IM-2	Discuss potential visible biases that could exist in a dataset and how these biases could affect analysis conclusions.	**Data & Analysis**	Inferences and Models	• Fostering An Inclusive Computing Culture • Communicating About Computing
9-12	3-DA-IM-1	Create computational models that represent the relationships among different elements of data collected from a phenomenon or process.	**Data & Analysis**	Inferences and Models	• Developing & Using Abstraction • Creating Computational Artifacts
9-12	3-DA-IM-2	Discuss potential hidden biases that could be introduced while collecting a dataset and how these biases could affect analysis conclusions.	**Data & Analysis**	Inferences and Models	• Fostering An Inclusive Computing Culture • Communicating About Computing
9-12	3-DA-IM-3	Evaluate the ability of models and simulations to test and support the refinement of hypotheses.	**Data & Analysis**	Inferences and Models	• Developing & Using Abstraction • Testing & Refining Computational Artifacts
K-2	1A-DA-ST-1	Identify data as information that is stored by software.	**Data & Analysis**	Storage	Developing & Using Abstractions
3-5	1B-DA-ST-1	Store, copy, search, retrieve, modify, and delete data using a computing device.	**Data & Analysis**	Storage	• Collaborating Around Computing • Recognizing & Defining Computational Problems
6-8	2-DA-ST-1	Store, retrieve, and share data to collaborate, using a cloud-based system.	**Data & Analysis**	Storage	• Collaborating Around Computing • Creating Computational Artifacts

6-8	2-DA-ST-2	Describe various low-level data transformations and identify which result in a loss of information	**Data & Analysis**	Storage	Developing & Using Abstractions
9-12	3-DA-ST-1	Explain tradeoffs between storing data locally or in central, cloud-based systems.	**Data & Analysis**	Storage	• Collaborating Around Computing • Creating Computational Artifacts
9-12	3-DA-ST-2	Translate data for various real-world phenomena, such as characters, numbers, and images, into bits.	**Data & Analysis**	Storage	Developing & Using Abstractions
K-2	1A-DL-CU-1	Use software tools to create simple digital artifacts	**Digital Literacy**	Creation and Use	Using Technology Appropriately
3-5	1B-DL-CU-1	Use software tools to create and share multimedia artifacts	**Digital Literacy**	Creation and Use	Using Technology Appropriately
6-8	2-DL-CU-1	Use software tools to create artifacts that engage users over time	**Digital Literacy**	Creation and Use	Using Technology Appropriately
9-12	3-DL-CU-1	Select appropriate software tools or resources to create a complex artifact or solve a problem.	**Digital Literacy**	Creation and Use	Using Technology Appropriately
K-2	1A-DL-SDI-1	Conduct basic digital searches.	**Digital Literacy**	Searching Digital Information	Using Technology Appropriately
3-5	1B-DL-SDI-1	Conduct and refine multi-criteria searches over digital information.	**Digital Literacy**	Searching Digital Information	Using Technology Appropriately
6-8	2-DL-SDI-1	Conduct searches over multiple types of digital information.	**Digital Literacy**	Searching Digital Information	Using Technology Appropriately

9-12	3-DL-SDI-1	Decompose a complex problem into multiple questions, identify which can be explored through digital sources, and synthesize query results using a variety of software tools.	**Digital Literacy**	Searching Digital Information	Using Technology Appropriately
K-2	1A-DL-US-1	Describe basic differences between humans and computers for performing computational tasks.	**Digital Literacy**	Understanding Software Tools	Using Technology Appropriately
3-5	1B-DL-US-1	Describe the different high-level tasks that are common to software tools that students use.	**Digital Literacy**	Understanding Software Tools	Using Technology Appropriately
6-8	2-DL-US-1	Describe the different formats of software components that support common tasks in software tools.	**Digital Literacy**	Understanding Software Tools	Using Technology Appropriately
9-12	3-DL-US-1	Describe different kinds of computations that software tools perform to tailor a system to individual users.	**Digital Literacy**	Understanding Software Tools	Using Technology Appropriately
K-2	1A-RC-CU-1	Compare and contrast how individuals live and work before and after the implementation or adoption of new computing technology.	**Responsible Computing & Society**	Culture	Recognizing & Defining Computational Problems
3-5	1B-RC-CU-1	Compare and contrast computing technologies that have changed the world, and express how those technologies influence, and are influenced by, cultural practices.	**Responsible Computing & Society**	Culture	Recognizing & Defining Computational Problems
3-5	1B-RC-CU-2	Identify ways to improve the accessibility and usability of technology products for the diverse needs and wants of users.	**Responsible Computing & Society**	Culture	Fostering An Inclusive Computing Culture

6-8	2-RC-CU-1	Compare and contrast tradeoffs associated with computing technologies that affect people's everyday activities and career options.	**Responsible Computing & Society**	Culture	Communicating About Computing
6-8	2-RC-CU-2	Discuss issues of bias and accessibility in the design of existing technologies.	**Responsible Computing & Society**	Culture	Fostering An Inclusive Computing Culture
9-12	3-RC-CU-1	Evaluate the ways computing impacts personal, ethical, social, economic, and cultural practices.	**Responsible Computing & Society**	Culture	Fostering An Inclusive Computing Culture
9-12	3-RC-CU-2	Design and analyze computational artifacts to reduce bias and equity deficits.	**Responsible Computing & Society**	Culture	Fostering An Inclusive Computing Culture Testing & Refining Computational Artifacts
9-12	3-RC-CU-3	Evaluate the impact of equity, access, and influence on the distribution of computing resources in a global society.	**Responsible Computing & Society**	Culture	Fostering An Inclusive Computing Culture
K-2	1A-RC-SLE-1	Discuss ownership and attribution of digital artifacts.	**Responsible Computing & Society**	Safety, Law & Ethics	Communicating About Computing
3-5	1B-RC-SLE-1	Incorporate public domain or creative commons media into a digital artifact, and refrain from copying or using material created by others without permission.	**Responsible Computing & Society**	Safety, Law & Ethics	Communicating About Computing
6-8	2-RC-SLE-1	Discuss how laws control use and access to intellectual property, and mandate broad access to information technologies.	**Responsible Computing & Society**	Safety, Law & Ethics	Communicating About Computing

9-12	3-RC-SLE-1	Evaluate the impact of intellectual property laws on the use of digital information	**Responsible Computing & Society**	Safety, Law & Ethics	Communicating About Computing
9-12	3-RC-SLE-2	Evaluate the social and economic implications of privacy and free speech in the context of safety, law, or ethics.	**Responsible Computing & Society**	Safety, Law & Ethics	Communicating About Computing
K-2	1A-RC-SI-1	Work respectfully and responsibly with others online.	**Responsible Computing & Society**	Social Interactions	Collaborating Around Computing
3-5	1B-RC-SI-1	Seek diverse perspectives for the purpose of improving computational artifacts.	**Responsible Computing & Society**	Social Interactions	Fostering An Inclusive Computing Culture
6-8	2-RC-SI-1	Collaborate and strategize with many online contributors when creating a computational or digital artifact.	**Responsible Computing & Society**	Social Interactions	• Collaborating Around Computing • Creating Computational Artifacts
9-12	3-RC-SI-1	Use tools and methods for collaboration on a project to increase connectivity between people in different cultures and career fields.	**Responsible Computing & Society**	Social Interactions	Collaborating Around Computing

Appendix A

COMPUTATIONAL THINKING & PROGRAMMING				
	Grades K-2	**Grades 3-5**	**Grades 6-8**	**Grades 9-12**
Algorithms	**Model daily processes by creating and following algorithms to complete tasks.** *An algorithm is a set of step-by-step instructions. Students should be able to break down simple actions into discrete component steps to achieve a goal. For example, students could create and follow algorithms for preparing a simple snack, brushing their teeth, getting ready for school, or contributing during clean-up time.* **Practice(s): 4.4**	**Compare and refine multiple algorithms for the same task and determine which is more appropriate to complete the task.** *Different algorithms can achieve the same result, though sometimes one algorithm might be most appropriate for a specific situation. Students should be able to look at different ways to solve the same task and decide which would be the best solution. For example, students could use a map or floor tiles within a school setting and plan multiple algorithms to get from one point to another.* **Practice(s): 3.3, 6.3**	**Use diagrams and/or pseudocode to plan, analyze, solve and/or code complex problems as algorithms.** *Complex problems are problems that do not have simple solutions. Pseudocode allows students to create an outline of the processes of a computer program through informal description. Students should be able to use pseudocode and/or flowcharts to organize and sequence an algorithm that addresses a complex problem, even though they may not actually program the solution. For example, students could express an algorithm, either as written description or as a flowchart that describes getting dressed in the morning and includes conditions such as outdoor temperature, precipitation, and the day of the week (school day or weekend).* **Practice(s): 4.1, 4.4**	**Create computational artifacts that use algorithms to solve computational problems by leveraging prior knowledge and personal interests.** *A computational artifact is something created by humans that works on a computing device. Examples include computer programs, digital images, and animations. The process of developing computational artifacts embraces both creative expression and the exploration of ideas to solve computational problems. Students should develop computational artifacts that demonstrate the performance, reusability, and ease of implementation of an algorithm. For example, students could create a computer program, website, or application that is based on a game that they have played, or based on a social issue that they care about.* **Practice(s): 5.2**
Variables	**Model real-world data and how it is stored.** *Students should be able to model physical data and how it is stored. For example, students could use a collection of bins/boxes labeled with names of the students in the class to store papers for those students, and discuss how boxes are named and how papers can be placed in the bins for storage and later removed to be read.* **Practice(s): 5.2**	**Create programs that use variables.** *Variables are used to name and store data. Students should understand how to use variables in programs. For example, students could use a variable to store the number of steps necessary for an onscreen character to move.* **Practice(s): 5.2**	**Create clearly named variables that represent different data. Perform operations on data stored in variables.** *Programmers create variables to store data values of selected types. When planning and developing programs, students should decide when and how to declare and name new variables. Students should use naming conventions to improve program readability. Students should use operations to change the value stored in variables. For example, students could add points to a score, combine user input with words to make a sentence, change the size of a picture, or add a name to a list of people.* **Practice(s): 5.1, 5.2**	**Explain the role of a variable within a program, and the scope in which its name and value can be used.** *Scoping rules determine where a variable's name and value may be referenced within a program. Students should be able to explain the purpose for which they created variables in a program, indicate where the variable can be accessed in the program, and provide reasoning about why the scope of a variable is appropriate (or not) for its role in the overall program. For example, students could explain that local variables can be used to store data that is only used within a procedure.* **Practice(s): 4.1**

Data Structures & Data Types	**Model real-world objects and/or processes that can be represented by various types of data.** *Students should be able to represent physical objects and their attributes as written data. For example, students could use thumbs up/down as representations of yes/no, use arrows when writing algorithms to represent direction, or encode and decode words using numbers, pictographs, or other symbols to represent letters or words.* **Practice(s): 4.4**	**Identify real-world examples of data structures and data types.** *Data structures hold multiple pieces of data about one thing. Examples of data structures are a list of planets and their diameters, or a phone contact with the first name, last name, and phone number of a person. Each piece of data has a data type, such as a diameter being a number, and a name being a string of characters. Students should be able to describe how data is grouped for an association with an entity in either the real world or in a computational artifact, and the type of that data. For example, students could describe the data structures and data types in an online game that has several characters.* **Practices(s): 3.1**	**Organize data into an appropriate data structure in a program.** *Students should be able to identify the components of data in a given computational problem, determine the type of each component, and propose a structural organization for those data. For example, students could represent characters in a game with a data type that has a name, picture, and position, and could represent the collection of characters on the screen as a program list of that data type.* **Practice(s): 5.1**	**Create a program that processes a collection of data.** *Programs often process collections of data, such as the collection of song titles in a playlist, or a collection of all sprites on the screen. Students should be able to organize multiple data items of the same type into a program data structure (such as a list or array) and write a program that computes a result about that collection. For example, students could check to see if any two sprites in a program list or array of sprites have collided on the screen.* **Practice(s): 5.2**
Control Structures	**Develop simple programs with sequences and simple repetitions.** *Programming control structures specify the order in which instructions are executed within a program. Sequences are the order of instructions in a program. For example, if dialogue is not sequenced correctly when programming a simple animated story, the story will not make sense. If the commands to program a robot are not in the correct order, the robot will not complete the desired task. Repetition constructs (which vary across languages but include loops) allow for performing operations multiple times. For example, students could model repetitions for handwashing by chanting "rub your hands, rub your hands....rub your hands," then replacing that with "rub your hands ten times."* **Practice(s): 5.2**	**Create programs that combine sequences, repetitions, conditionals, and/or events.** *Control structures specify the order in which instructions are executed within a program and can be combined to support the creation of more complex programs. Events allow portions of a program to run based on a specific interaction with the program, such as the user clicking the mouse. Students should be able to create a program using events, conditionals, and repetitions. For example, students could write a program to explain the water cycle and when a specific component is clicked (event), the program would show information about that part of the water cycle. Conditionals allow for the execution of a portion of code in a program when a certain condition is true. For example, students could write a math game that asks a multiplication question and then uses a conditional to check whether or not the answer that was entered is correct. An example of a combined program would be a math quiz program that loops through multiple questions each with a conditional to check for the right answer.* **Practice(s): 5.2**	**Design programs that combine control structures, including nested repetitions and compound conditionals.** *Control structures can be combined in many ways. Nested repetitions consist of operations that are repeated within other repeated operations. Compound conditionals combine two or more conditions in a logical relationship (e.g., using AND, OR, and NOT), and nesting conditionals within one another allows the result of one conditional to lead to another. For example, when programming an interactive story, students could repeatedly check whether a character has a key and is touching the door before unlocking the door.* **Practice(s): 5.1, 5.2**	**Create and justify the selection of specific control structures when tradeoffs involve code organization, readability, and program performance and explain the benefits and drawbacks of choices made.** *A programmer has choices about what control structures to use. Students should be able to choose good control structures and describe why they chose them. Readability refers to how clear the program is to other programmers. The discussion of performance is limited to a theoretical understanding of execution time and storage requirements; a quantitative analysis is not expected. For example, the students could explain why using a repetition construct is preferable to writing (nearly) identical code multiple times. Another example is that students could program with series of if-else statements for a complex condition, and describe how this differs in execution and readability than a similar series of if statements (without "else" clauses) for that complex condition.* **Practice(s): 5.2**

Modularity	**Decompose a task into a set of smaller tasks.** *Decomposition is the act of breaking down tasks into simpler tasks. Students should be able to list steps for a particular task. For example, students could describe the tasks needed to get ready to go home from school.* **Practice(s): 3.2**	**Continually decompose problems into smaller subtasks until each subtask is a manageable set of basic operations.** *Students should break problems into hierarchies of subtasks, each of which in turn decomposes into other subtasks or individual steps. For example, students could plan a party by separating the task into subtasks such as inviting guests, getting party favors, planning games, and preparing food. The inviting guest subtask could be broken into its own subtasks of determining a guest list, writing invitations, and sending invitations - where each of these subtasks is a manageable set of basic operations a person knows how to do such as write and send an email with the invitation.* **Practice(s): 3.2**	**Decompose computational problems to facilitate the design and implementation of programs.** *Decomposition facilitates aspects of program development by allowing students to focus on one piece at a time (e.g., getting input from the user, processing the data, and displaying the result to the user). Decomposition also enables different students to work on different parts at the same time. Students should be able to decompose a computational problem into subtasks that facilitate the use of appropriate programming language constructs that reflect the subtasks of the problem solution. For example, students could match subtasks in their problem decomposition to procedures that they will program in a programming language.* **Practice(s): 3.3, 5.1**	**Identify existing computational artifacts that can be used for the subtasks of a decomposed problem.** *Students should be able to take a problem that they decomposed to subtasks and identify existing computational solutions to the subtask. For example, students could find a library of procedures to do data visualization and use those procedures to display data in their program.* **Practice(s): 3.2**
	No K-12 standard.	**Create computational artifacts by incorporating existing modules into one's own work to solve a problem.** *Students should be able to combine existing computational artifacts into their own computational artifact. For example, students could combine pictures and text into a meme picture or create a program using the existing instructions and built-in functions of a programming language/environment.* **Practice(s): 4.2, 5.3**	**Create procedures with parameters to organize code and make it easier to reuse.** *Procedures and/or functions can be used multiple times within a program to repeat groups of instructions.Students should be able to name the procedures appropriately to match their function and be able to define parameters that create different outputs for a wide range of inputs. For example, all student could create a procedure to draw circles of different sizes by adding a radius parameter.* **Practice(s): 4.1, 4.3**	**Create computational artifacts by incorporating pre-defined procedures, self-defined procedures and external artifacts.** *Computational artifacts can be created by combining and modifying existing external artifacts or by developing new artifacts. Interacting modules, each with a specific role but coordinated for a common overall purpose, allow for better management of complex tasks. Students should be able identify existing external modules that they can use, create modules that don't have a good existing solution, and combine these modules to create a computational artifact. For example, students could create an original web site and use open-source JavaScript libraries to expand its functionality. As another example, students could create their own modules to clean and process specific data, and then use existing modules from an external library to display* **Practice(s): 5.2, 5.3**

Computational Design	**Develop a plan that describes what a computational artifact should look like and how it should perform.** *Creating a plan for what an artifact should like and do clarifies the steps that will be needed to create it and can be used to check if it is correct. Students should be able to* look*complete a planning process with the teacher's assistance. For example, students could create a planning document such as a story map, a storyboard, or a sequential graphic organizer to illustrate the program.* **Practice(s): 5.1, 7.2**	**Use an iterative process to plan the development of a computational artifact by including others' perspectives and considering user preferences.** *Planning is an important part of the iterative process of program development. Students should be able to outline key features, time and resource constraints, and user expectations. For example, students could document a plan using storyboards, flowcharts, pseudocode, or user-interface sketches.* **Practice(s): 1.1, 5.1**	**Seek and incorporate feedback from team members and users to refine a solution that meets user needs.** *Development teams employ user-centered design to create solutions (e.g., programs and devices) that support the needs of end users, such as an application that allows people with speech difficulties to translate hard-to-understand pronunciation into understandable language. For example, students could begin to seek diverse perspectives throughout the design process to improve their computational artifacts by focusing on usability, accessibility, age-appropriate content, respectful language, user perspective, pronoun use, color contrast, and ease of use.* **Practice(s): 1.1, 2.3**	**Systematically design and implement computational artifacts for targeted audiences by incorporating feedback from users**. *Students should be able to follow a systematic process that includes feedback from broad audiences on their computational artifact. For example, students could create a user satisfaction survey, identify distribution methods that could yield feedback from a diverse audience on the usability and effectiveness of their website and document the process of incorporating feedback.* **Practice(s): 5.1**
	Identify a task that includes sequences and simple loops. *Students should be able to implement a simple algorithm, determine if it is incorrect, and fix errors. For example one student could direct another student from a start location to an end location by holding up arrows. If the second student fails to get to the end location, the directing student should determine where the mistake occurred, and correct the mistake.* **Practice(s): 6.2**	**Debug errors in an algorithm or program that includes sequences and simple loops.** *Algorithms or programs may not always work correctly. Students should be able to use various strategies, such as following the algorithm in a step-by-step manner, changing the sequence of the steps, and/or using trial and error to fix problems in algorithms and programs. For example, students could check their onscreen characters to see if they are colliding because of too many repetitions.* **Practice(s): 6.2**	**Test and debug a program to ensure it runs as intended.** *As programs are developed, they should be continuously tested to ensure they run as intended. If not, errors should be identified and fixed. Students should also be able to successfully debug simple errors in programs created by others. For example, students could review programs intentionally created with errors to identify the problem and determine the fix.* **Practice(s): 6.1, 6.2**	**Systematically test and refine programs using a range of test cases.** *Test cases and use cases are created and analyzed to better meet the needs of users and to evaluate whether programs function as intended. Students should be able to recognize that testing is a deliberate process that is iterative, systematic, and proactive. For example, students could begin to test programs by considering potential errors, such as what will happen if a user enters invalid inputs (e.g., negative numbers and 0 instead of positive numbers).* **Practice(s): 6.1**

Computational Design	**No K-12 standard.**	**Describe steps taken and choices made during the process of creating a computational artifact.** *Students should be able to talk or write, using appropriate terminology, about the goals and expected outcomes of the computational artifacts that they create and the choices that they made. For example, students could describe their work using a notebook of designs, coding journals, discussions with a teacher, class presentations, or blogs.* **Practice(s): 7.2**	**Describe choices made during development of computational artifacts.** *People communicate about their code to help others understand and use their programs. Students should be able to explain their design choices to demonstrate an understanding of their work. For example, students could include these explanations as in-line code comments for collaborators and assessors, or as part of a summative presentation, such as a code walk- through or coding journal.* **Practice(s): 7.2**	**Document computational artifacts in order to make them easier to follow, test, and debug.** *Documentation allows creators and others to more easily use and understand a program. Students should provide documentation for end users that explain their artifacts, how they are used, and why they act the way they do. For example, students could provide a project overview, design rationale and clear user instructions. They should communicate their process using design documents, flowcharts, and presentations.* **Practice(s): 7.2**

COMPUTING SYSTEMS AND NETWORKS				
	Grades K-2	**Grades 3-5**	**Grades 6-8**	**Grades 9-12**
Human-Computer Interaction	**Identify the inputs and outputs of a computer system.** *There are many ways to exchange information with a computer. Students should be able to identify the components of a computer system that help people input information and the parts that produce a desired output. For example, students could use a digital keyboard to input letters and symbols or press play on a video player to start a video, or drag a digital object from one location to another.* **Practice(s): 7.2**	**Describe how people interact with the various parts of computing systems to accomplish tasks.** *Computing devices often depend on human interactions to trigger particular actions. A keyboard input or a mouse click may cause a change in information displayed on a screen. Students should understand that computers are programmed to produce certain outputs based on particular inputs, and should be able to describe how people and devices interact, using appropriate terminology. For example, students could describe all of the actions needed to play their favorite video game or to use their favorite app.* **Practice(s): 7.2**	**Identify improvements to the design of computing devices, based on an analysis of how users interact with the devices.** *The study of human–computer interaction (HCI) can improve the design of both hardware and software of devices. For example, an assistive device may include a microphone (hardware sensor) that converts spoken words to written text. Students should be able to understand that devices can be designed for a variety of purposes, including accessibility, ergonomics, and learnability. For example,students could make recommendations for improvements to existing devices (e.g., a laptop, smartphone, or tablet), software (applications), or they could design their own component or software interface (e.g., create their own controllers).* **Practice(s): 1.1**	**Analyze a computing system and explain how abstractions simplify the underlying implementation details embedded in everyday objects.** *Computing devices are often integrated with other systems, including biological, mechanical, and social systems. Examples are: a medical device can be embedded inside a person to monitor and regulate; an assistive listening device can filter out certain frequencies and amplify others; a monitoring device inside 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. Students should be able to describe (but not create) an integrated or embedded systems. For example, students could select an embedded device such as a car stereo, identify the types of data (radio station presets, volume level) and procedures (increase volume, store/recall saved station, mute button), and explain how the implementation details are hidden from the user.* **Practice(s): 4.1**
Hardware & Software	**Use appropriate terminology in identifying and describing the function of common physical components of computing systems (hardware).** *A computing system is composed of hardware and software. Hardware consists of physical components. Students should be able to identify and describe the function of external hardware, such as desktop computers, laptop computers, tablet devices, monitors, keyboards, mice, and printers. For example, students could label and match components with their descriptions.* **Practice(s): 7.2**	**Model how computer hardware and software work together as a system to accomplish tasks.** *Both hardware and software are needed to accomplish tasks with a computer. Students should recognize the basic elements of a computer system, including input, output, processor, sensors, and storage. For example, students could draw a model (on paper or digitally), program an animation of the model, or describe it through body movements or role playing.* **Practice(s): 4.4**	**Design projects that combine hardware and software components to collect and use data to perform a function.** *Collecting and exchanging data involves input, output, storage, and processing. Students should be able to select the hardware and software components for their project designs by considering factors such as functionality, cost, size, speed, accessibility, and aesthetics. For example, students could design a mobile application that includes accelerometer, GPS, and speech recognition* **Practice(s): 5.1**	**Compare levels of abstraction and interactions between application software, system software, and hardware layers.** *At its most basic level, a computer is composed of physical hardware and electrical impulses with multiple layers of software built upon the hardware. 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. Students should be able to recognize that system software is used on many different types of devices, such as smart TVs, assistive devices, virtual components, cloud components, and drones. For example, students could explain the progression from voltage to binary signal to logic gates to adders and so on.* **Practice(s): 4.1**

Troubleshooting	**Describe basic hardware and software problems using appropriate terminology.** *Problems with computing systems have different causes. Students should be able to communicate a problem with appropriate terminology, although they do not need to understand the causes. For example, students could notify a teacher when an application or program is not working as expected, such as when a device will not turn on or there is no sound.* **Practice(s): 6.2, 7.2**	**Determine potential solutions to solve simple hardware and software problems using common troubleshooting strategies.** *Although computing systems may vary, common troubleshooting strategies can be used on all of them. Students should be able to identify solutions to basic problems, such as the device not responding, no power, no network connection, application crashing, no sound, or password entry not working. For example, when errors occur, students could use various strategies, such as rebooting the device, checking for power, checking network availability, closing and reopening an application, making sure speakers are turned on or headphones are plugged in, and making sure that the caps lock key is not on, in an attempt to solve these problems.* **Practice(s): 6.2**	**Identify and fix problems with computing devices and their components using a systematic troubleshooting method or guide**. *Since 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. Students should be able to use a structured process – similar to the checklist used by aircraft pilots – to troubleshoot problems with computing systems, and ensure that potential solutions are not overlooked. For example, students could follow a troubleshooting flow diagram, make changes to software to see if hardware will work, check connections and settings, or change working components.* **Practice(s): 6.2**	**Develop and communicate troubleshooting strategies others can use to identify and fix errors.** *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. Students should be able to identify complex troubleshooting strategies, which include resolving connectivity problems, adjusting system configurations and settings, ensuring hardware and software compatibility, and transferring data from one device to another. For example, students could create a flowchart, a job aid for a help desk employee, or an expert system (artificial intelligence).* **Practice(s): 6.2**
Networks & the Internet	**Describe the Internet is a place to share and find information.** The *Internet transmits information between computers. Students should understand that information accessed on the Internet can be stored and shared on computers around the world. Students should be able to identify the value of a network like the Internet to find information and access other services. For example, students could look for information that comes from a remote location.* **Practice(s): 7.2**	**Model how information is broken down into smaller pieces of data, transmitted as packets through multiple devices over networks and the Internet, and reassembled at the destination.** *Data are sent and received over physical cables and wires or wireless paths. They are broken down into smaller pieces (packets) which are sent independently and reassembled at the destination. Students should be able to recognize different types of networks for specific purposes (i.e. the school's local network versus the Internet). For example, students could demonstrate their understanding of this flow of data by drawing a model of the way packets are transmitted, programming an animation to show how packets are transmitted, or demonstrating this through body movements or role playing activities.* **Practice(s): 4.4**	**Model the role of protocols in transmitting data across networks and the Internet.** *Protocols are rules that define how messages between computers are sent. Students should understand the purpose of protocols and how they enable secure and errorless communication as well as model protocols that enable the fastest path, deal with missing information, and deliver sensitive data securely. For example, students could devise a plan for resending lost information or for interpreting a picture that has missing pieces.* **Practice(s): 4.4**	**Identify the various elements of a network and describe how they function and interact to transfer information**. *Large-scale coordination occurs among many different machines across multiple paths every time a web page is opened or an image is viewed online. Students should be able to explain the path of communication from their device to a website and back using the network topology (servers, routers, switches, DNS, ISP, etc.). For example, students could use online network simulators to experiment with these factors. experiment with these factors.* **Practice(s): 7.2**

CYBERSECURITY				
	Grades K-2	**Grades 3-5**	**Grades 6-8**	**Grades 9-12**
Risks	**Keep login and personal information private, and log off of devices appropriately.** *Computing technology can help or hurt people. Students should recognize and avoid harmful behaviors, such as sharing private information and staying logged on public devices. For example, students could identify what information might be private about a photograph, and demonstrate that they know how to log out of any accounts they use for classroom work.* **Practice(s): 8.1**	**Describe the risks of sharing personal information, on websites or other public forums**. *Security attacks often start with information that is publicly available online. Students should be able to recognize that sharing information about family members and friends can put them at risk. For example, students could create a list of potential risks of sharing persona information in real life, where personal information includes identifying information such as birthdates as well as information about current locations.* **Practice(s): 8.2**	**Describe tradeoffs between allowing information to be public and keeping information private and secure.** *Sharing information online can help establish, maintain, and strengthen connections between people. It allows artists and designers, for example, to display their work and reach a broad audience. People must decide which information to share and which to protect. Students should recognize that different situations require different safeguards and that not everyone will agree on what to share. For example, students could list the pros and cons of sharing pictures and information about their activities on social media or a school directory.* **Practice(s): 8.2**	**Explain the privacy concerns related to the collection and generation of data through automated processes that may not be evident to users.** *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. Students should be able to identify privacy concerns related to the automated and non-evident collection and recognize that all information will be part of big data. For example, students could analyze the benefits and consequences of 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 purchasing habits, or the monitoring of road traffic to change signals in real-time to improve road efficiency without drivers being aware.* **Practice(s): 8.3**
	No K-2 standard.	**Describe ways personal information can be obtained digitally.** *Websites use various methods to gather information about individuals, families, or friends. Information may survive online even after the owner deletes it. Students should be able to recognize ways that data are gathered online. For example, students could list ways websites gather data such as by asking questions, selling products and tracking people's website visits.* **Practice(s): 8.2**	**Describe social engineering attacks and the potential risks associated with them.** *Social engineering is based on "tricking" people into revealing sensitive information. It can be thwarted by being wary of attacks, such as phishing and spoofing. Students should be able to recognize that attacks can come through links in email, ads on websites, and questions from other people. For example, students could describe a potential phishing attack.* **Practice(s): 8.2**	**Analyze an existing or proposed application to identify the potential ways it could be used to obtain sensitive information.** *Applications gather and store information about users and their behaviors. The risks an application poses depends on what information it gathers, where that information is stored, and who can access that information. Risks may concern reputational, financial, or legal issues. Students should be able to differentiate between methods and devices for collecting data by the amount of storage required, level of detail collected, and sampling rates.* **Practice(s): 3.1, 8.2, 8.3**

Risks	**No K-2 standard.**	**Describe the risks of others using one's personal resources or devices.** *When access to a smartphone, network, or account is shared, information about the "owner" can be exposed. Students should be able to discuss potential consequences to giving friends and acquaintances access to their devices or accounts. For example, students could list inappropriate actions that someone could take if they had access to a digital folder containing another student's classwork.* **Practice(s): 8.1, 8.2**	**Describe risks of using free and open services**. *Free services often carry less security than paid services. Students should be able to identify situations in which they might be using free services and the corresponding risks. Examples include using the risk of information theft over open networks in restaurants, and the risk of malware installation from sites for streaming licensed entertainment (e.g., movies or sporting events). Additionally, certain smartphone or laptop applications may request permissions that may compromise personal information. For example, students could identify certain phone or laptop applications that request permissions that may compromise personal information.* **Practice(s): 8.2**	**Explain how the digital security of an organization may be affected by the actions of its employees.** *Organizations store sensitive, confidential and proprietary information in their computing systems. Employees have a responsibility to help protect these systems and their data. Students should understand how an employee is an integral part of an organization's digital security and how an individual's digital inattentiveness can have serious consequences. For example, students could identify and describe certain contexts within industry, military, healthcare, energy, and government organizations where consequences would be serious.* **Practice(s): 8.1**
Safeguards	**Recognize basic digital security features.** *Devices and software use several mechanisms to safeguard data. For example, devices have passcodes. Software tools have password-protected accounts so that one user cannot see another user's data. Websites display a lock icon when they are using certain common security measures. Students should be able to recognize whether a device or software tool offers basic data protection based on passwords, accounts, and padlock icons. For example, students could create passwords that include numbers, upper & lower case letters & special symbols.* **Practice(s): 8.1**	**Explain individual actions that protect personal electronic information and devices.** *Just as we protect our personal property offline, we also need to protect our devices and the information stored on them. Information can be protected using various security measures. These measures can be physical and/or digital. Digital protection should be based on strong personal authentication (e.g., passwords). Students should be able to identify these measures and describe how to use them. For example, students could describe what makes a password strong and how to safeguard their passwords, describe what anti-virus software does and how to keep it updated, and whether various applications that they use backup data automatically in the cloud.* **Practice(s): 8.1**	**Explain physical and digital security measures that protect electronic information.** *Information that is stored online is vulnerable to unwanted access. Physical security measures to protect data include keeping passwords hidden, locking doors, making backup copies on external storage devices, and erasing a storage device before it is reused. Digital security measures include secure router admin passwords, using two-factor authentication, firewalls that limit access to private networks, installing software updates, using (and not disabling) malware detectors, and the use of a protocol such as HTTPS to ensure secure data transmission. Students should be able to differentiate between physical and digital security measures. For example, students could create a list of security measures for the school and discuss with the onsite IT professional.* **Practice(s): 8.2, 8.3**	**Recommend security measures to address various scenarios based on factors such efficiency, feasibility, and ethical impacts.** *Security measures may include physical security tokens, two-factor authentication, and biometric verification. 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. Students should systematically evaluate and continually re-assess the feasibility of using computational tools to solve given security problems or sub problems. For example, students could use a cost-benefit analysis to evaluate (eventually including more factors in their evaluations) such as how efficiency affects feasibility or whether a proposed approach raises ethical concerns.* **Practice(s): 8.3**

Safeguards	**No K-2 standard**	**No 3-5 standard.**	**Demonstrate how multiple methods of encryption provide secure transmission of information.** *Encryption can be as simple as letter substitution or as complicated as modern methods used to secure networks and the Internet. Students should encode and decode messages using a variety of encryption methods, and should understand the different levels of complexity used to hide or secure information. For example, students could secure messages using methods such as Caesar ciphers or steganography (i.e., hiding messages inside a picture or other data). They can also model more complicated methods, such as public key encryption,through unplugged activities.* **Practice(s): 8.2**	**Explain tradeoffs when selecting and implementing cybersecurity recommendations.** *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. Every security measure involves tradeoffs between the accessibility and security of the system. Students should be able to describe, justify, and document choices they make using terminology appropriate for the intended audience and purpose. For example, students could debate issues from the perspective of diverse audiences, including individuals, corporations, privacy advocates, security experts, and government.* **Practice(s): 8.3**
Response	**Identify situations applications and devices that should be reported to a responsible adult.** *Losing a device or accidentally sharing a password exposes people and their accounts to information theft. Students should understand which kinds of losses to report to a parent, teacher, or other trusted adult. (Instructions should follow any appropriate district or school policies that are intended for young students.)* **Practice(s): 8.1**	**Identify and describe unusual data or behaviors of applications and devices that should be reported to a responsible adult.** *Devices or applications can behave in unexpected ways when a security incident occurs. An unusual screen might open asking for a password, phone number, or permission to install another program. Email attachments can contain malicious software. Students should be able to recognize simple forms of unusual behavior in common applications, including data or links that might constitute a risk, and understand which behaviors to report to a parent, teacher, or other trusted adult. (Instructions should follow any appropriate district or school policies that are intended for young students.) For example, students could review sample email messages and identify features of each that suggest suspicious behavior.* **Practice(s): 8.1**	**Describe which actions to take and not to take when an application or device reports a problem or behaves unexpectedly.** *Users can take defensive actions when devices, applications, or online acquaintances behave in unexpected ways. Disconnecting a device from a network, changing passwords, and removing or blocking people on social media applications are each appropriate in some situations. Students should be able to recognize basic situations in which to take or not take various defensive actions. For example, students could describe benefits and risks to each action (such as the loss of forensic data if a device is completely turned off, or that requests to change passwords should only be followed when they come from an appropriate authority). Students should be able to describe both basic forms of social engineering attacks and how to decide which links or documents to open.* **Practice(s): 8.**2	**Describe the appropriate actions to take in response to detected security breaches.** *Employees may observe activity such as malware scans, modifications to documents, or unusual access to documents by other employees when an organization suffers a security breach. Breaches also occur on a personal level with credit cards or social media accounts. Students should be able to recognize different kinds of breaches that they might detect as employees or in their personal lives. For example, students could discuss sample organizational response protocols, sample state and federal policies on data loss and incidence response, and explain their responsibilities to organizations in responding to breaches. In terms of personal information, students could discuss what sorts of information individuals should gather in advance as evidence of data theft, loss, or fabrication and identify the appropriate people (such as banks, the police, or acquaintances) to whom to report incidents.* **Practice(s): 8.3**

DATA & ANALYSIS				
	Grades K-2	**Grades 3-5**	**Grades 6-8**	**Grades 9-12**
Collection, Visualization, & Transformation	**Collect and present the same data in multiple formats.** *The collection and use of data about the world around us is a routine part of life and influences how people live. Different presentations of data highlight different aspects of the data. Students should be able to collect and tally simple data, then present it two or more ways using different representations. For example, students could count the number of blocks by color, or survey classmates on their favorite foods, then present these data with a bar graph and pictograph.* **Practice(s): 4.4, 7.2**	**Organize and present collected data to highlight relationships and support a claim.** *Raw data are sets of observations that have little meaning on their own. Data are often sorted or grouped to focus on particular questions. Organizing data can make interpreting and communicating them to others easier. Students should sort the same data set in multiple ways, presenting each in an appropriate format. For example, a data set of sports teams could be sorted by wins, points scored, or points allowed, with a bar graph presented for each sorting.* **Practice(s): 4.1, 7.1**	**Collect data using computational tools or online sources and transform the data to make it more useful and reliable.** *For analysis to be reliable, data need to be in a consistent format, free of critical errors or noise, and organized to make key trends visible. Students should be able to collect data from devices or online sources and identify noise, errors, or inconsistencies that could exist in these data. For example, an audio sensor meant to record applause volume may collect extraneous sound during the first few seconds of positioning the sensor, and these parts of recorded sound data should be excluded in analysis. Students could download a spreadsheet containing teacher-crafted dataset from a website and check for missing or duplicate entries.* **Practice(s): 6.2, 6.3**	**Select appropriate data-collection tools and presentation techniques for different types of data.** *Different kinds of data are best collected and presented using different methods and formats. Students should be able to distinguish between discrete and dynamic events, choose appropriate methods or tools for gathering each type of data, and choose appropriate representations for aggregating or presenting such data. For example, a tally counter could be used to compare the number of people attending a concert on different days, with data presented in a bar chart, whereas a light sensor could be used to detect change in illumination during the course of a cloudy versus sunny day, with data plotted as two trends along the same time axis.* **Practice(s): 4.1, 7.2**
Inference and Models	**Identify and describe patterns in data, presentations such as charts or graphs, to make predictions** *Data can be used to make inferences or predictions about the world. Students should be able to interpret simple graphs or charts. For example, students could analyze a chart representing what classmates ate for breakfast for several days, then make a prediction about how many student will eat a particular breakfast on another day.* **Practice(s): 4.1**	**Use data to highlight or propose cause-and-effect relationships, predict outcomes, or communicate an idea.** *The accuracy of data analysis is related to how realistically data are represented. Inferences or predictions based on data are less likely to be accurate if data are not sufficient, or if the data are incorrect in some way. Students should be able to refer to data when forming hypotheses and communicating an idea. They should recognize when data are in sufficient quantity, or relevant. For example, students could record the temperature at noon each day as a basis to show that temperatures are higher in certain months of the year, but these data would be insufficient measurements if only taken once a month. Data about precipitation would not be relevant to this prediction about temperature.* **Practice(s): 5.1, 7.1**	**Create and refine computational models based on generated or gathered data.** *A computational model may be a programmed simulation of events or a mathematical representation of how different objects relate. Students should be able to create and refine a model by considering which data points are relevant, how data points relate to each other, and if the data are accurate. Models can be created in simulation tools, spreadsheets, or on paper. For example, students may make a prediction about how far a ball will travel based on a table of data related to the height and angle of a track. The students could then test and refine their model by comparing predicted versus actual results and considering whether other factors are relevant (e.g., size and mass of the ball).* **Practice(s): 4.4, 5.3, 6.1**	**Create computational models that represent the relationships among different elements of data collected from a phenomenon or process.** *Computational models make predictions about processes or phenomena based on selected data and features.The amount, quality, and diversity of data and the features chosen can affect the quality of a model and therefore our ability to understand a system. Predictions or inferences are tested to validate models. Students should model phenomena as systems, with rules governing the interactions within the system, then analyze and evaluate these models against real-world observations. For example, students could create a simple producer–consumer ecosystem model, or a traffic-pattern prediction model, using a programming or simulation tool.* **Practice(s): 4.4, 5.1, 5.2**

Inferences & Models	**No K-2 standard.**	**No 3-5 standard.**	**Discuss potential visible biases that could exist in a dataset and how these biases could affect analysis conclusions.** *Datasets may not be representative of the population they are being used to study. Students should identify potential biases in the components of a dataset and discuss whether those biases could adversely affect specific conclusions drawn from the dataset. For example, students could analyze survey data to see if key populations are underrepresented.* **Practice(s): 1.3 , 7.1**	**Discuss potential hidden biases that could be introduced while collecting a dataset and how these biases could affect analysis conclusions.** *Datasets may have hidden biases based on how they were collected. Students should identify potential biases that are not directly reflected in the dataset, but that could exist based on how the data were collected. Students should also discuss whether those biases could adversely affect specific conclusions drawn from the dataset. For example, students could analyze data about public works funding for one neighborhood and discuss whether the analysis should predict funding for another neighborhood in the same city.* **Practice: 1.1, 1.3, 7.1**
	No K-2 standard.	**No 3-5 standard.**	**No 6-8 standard.**	**Evaluate the ability of models and simulations to test and support the refinement of hypotheses.** *Models must be evaluated to determine whether they are appropriate and reliable. A model may omit information that is essential to answering a question, or yield results that seem inconsistent with expectations or data from other sources. Students should learn how to test models on data for which results are known, articulate questions they could ask to validate a model, give detailed explanations of where a model might be inaccurate, and make suggestions on how to modify existing models to improve their reliability.For example, students could analyze the accuracy of a weather model if predictions of patterns of rain are inconsistent with recent historical data. Inconsistencies with real data could be used to modify the model, or refine a hypothesis about why rain data is inconsistent with recent historical data.* **Practice(s): 4.4, 6.3**

Storage	**Identify data as information that is stored by software.** *All information stored and processed by a computing device is referred to as data. Data can exist in different forms, such as images, text documents,audio files, video files, software programs, or applications. Storing data digitally makes it easier to create and share copies of the data. Students should be able to identify what kinds of information is stored by various software tools that they use and give examples of when they might want to share data with others. For example, a document stores text and pictures, while sharing the document lets teachers mark comments on it.* **Practice(s): 4.2**	**Store, copy, search, retrieve, modify, and delete data using a computing device.** *Data can be stored, copied, searched, retrieved, modified, and deleted, and can be stored either on local or remote devices. Students should be able to perform each of these operations within the relevant software applications that they use, and understand when a digital tool is manipulating data. Students should also understand that their privileges to manipulate data may depend on who owns the data. For example, students can talk about how they carry out these operations on email or text messages, and who is allowed to execute these operations on their own accounts.* **Practice(s): 2.4, 3.2**	**Store, retrieve, and share data to collaborate, using a cloud-based system.** *Cloud-based applications enable multiple people to update data from multiple locations. This enables backup systems to protect data as well as collaboration on documents and processes. Students should create and modify information using cloud-based systems. For example, students could contribute to a dataset recorded in a shared spreadsheet, take an online class for certification, or contribute regional images to a national database of images.* **Practice(s): 2.4, 5.3**	**Explain tradeoffs between storing data locally or in central, cloud-based systems.** *Local and cloud-based storage of data have different affordances with respect to accessibility, cost, speed, security, and integrity. Students should be able to discuss these tradeoffs in the context of decisions about managing specific forms of data. For example, students could discuss benefits and limitations to both local and cloud-based storage of medical records by a doctor's office.* **Practice(s): 2.4, 5.1**
	No K-2 standard.	**No 3-5 standard.**	**Describe various low-level data transformations and identify which result in a loss of information** *Data representations occur at multiple levels of abstraction, from the physical storage of bits to the arrangement of information into organized formats (e.g. tables). Some data representations compress data to save space, which results in a loss of information. Students should be able to represent the same data in multiple ways, discussing whether each representation loses any information. For example, students could represent the same color using both RGB values, hex codes, and greyscale values, discussing potential loss of hue. Students could experiment with different audio-compression formats, listening for when they can detect differences in sound quality.* **Practice: 4.1, 4.3**	**Translate data for various real-world phenomena, such as characters, numbers, and images, into bits.** *Computers ultimately store data as sequences of binary digits (bits) of 0s and 1s. Letter characters, numbers, and images can be represented as numbers, which in turn can be converted to bits. Students should understand why computers use bit-level representations, and should be able to convert multiple types of basic data into bit-level form. For example, students could translate a 4-digit number into bits, a two-word phrase into ASCII or Unicode, and several image pixels into sequences of hexadecimal color codes* **Practice(s): 4.1**

DIGITAL LITERACY				
	Grades K-2	**Grades 3-5**	**Grades 6-8**	**Grades 9-12**
Creation & Use	**Use software tools to create simple digital artifacts.** *People perform many common operations when creating artifacts with software tools. Students should be able to operate and interact with software tools to create an artifact. For example, students could write a text document that includes an image.* **Practice(s): 8.1**	**Use software tools to create and share multimedia artifacts.** *When creating digital artifacts, people use local, networked and online tools. Students should be able to create, manipulate, and publish multimedia artifacts using software tools from multiple environments. For example, students might create a video with a local tool, then upload the video to an online folder or blog for sharing with friends and family.* **Practice(s): 8.1**	**Use software tools to create artifacts that engage users over time** *Many artifacts are meant to be viewed either in multiple stages (like slideshows) or over multiple visits (like blogs). Students should understand how different kinds of artifacts can engage users over time, and create appropriate artifacts that do so. Students should also describe challenges in keeping such artifacts up to date. For example, students could create a multimedia slideshow showing the history of a local issue, or create a blog with updates on a topic of interest, debating policies for updating older blog posts in the presence of permalinks. .* **Practice(s): 8.1**	**Select appropriate software tools or resources to create a complex artifact or solve a problem.** *People use multiple kinds of software tools when creating interactive artifacts or solving problems. Students should select software tools, artifact styles, and resources based on their efficiency and effectiveness for a given purpose, project or assignment, and be able to justify their choices. For example, students might combine data from citizen science databases, news archives and image databases to create a website that presents presents dynamically-updating information.* **Practice(s): 8.1, 8.3**
Searching Digital Information	**Conduct basic digital searches.** *Digital data repositories often provide ways for users to search for information that interests them. Students should be able to conduct basic keyword searches to find information in digital resources. For example, students could search for books on specific topics in a library catalog or on the Internet.* **Practice(s): 8.1**	**Conduct and refine multi-criteria searches over digital information.** *Queries over digital information often consider multiple constraints. Students should be able to perform searches that involve multiple criteria, whether through search tools with separate fields or by using "and" or "or" operations within queries. Students should also be able to refine results of one query with additional constraints. For example, students could search for local events that occur within a specific date range in a website with local event listings, then restrict the results to music concerts.* **Practice(s): 8.1**	**Conduct searches over multiple types of digital information.** *Digital information comes in forms other than textual documents. Students should be able to search for other forms of data, such as images or audio files, paying attention to Copyright and Fair Use on discovered resources. For example, students could search for graphics in a specific file format to include in a presentation, while establishing that the licensing on the graphic allows such use.* **Practice(s): 8.1, 8.2**	**Decompose a complex problem into multiple questions, identify which can be explored through digital sources, and synthesize query results using a variety of software tools.** *Realistic problems are answered by combining the results of several more focused questions, some of which can be answered through digital information. Students should be able to break down a problem into focused questions, query digital sources when appropriate, synthesize results into an answer to the original problem, while properly identifying and citing sources. For example, students could ask how weather might affect crime statistics for a chosen city or region, search for geo-tagged crime incident data and weather data, then synthesize the data to find and present an answer to an information problem.* **Practice(s): 8.1, 8.3**

Understanding Software Tools	**Describe basic differences between humans and computers for performing computational tasks.** *Both humans and digital devices can solve problems. Students should recognize that computers and digital technology assist humans in performing tasks or making digital computations, and that humans have opinions, while digital devices do not. For example, students could use a sort function to organize a database of images by color value (tool), then discuss which one of the images is their favorite (human)* **Practice(s): 8.1**	**Describe the different high-level tasks that are common to software tools that students use.** *Many software tools are built around a common collection of tasks, such as gathering input from users, storing data, presenting data, protecting data, performing some computation over data, or connecting to other resources or services to perform a computation. Students should be able to describe these tasks and their associated data for at least two different kinds of software that they use on a regular basis. For example, students could describe the data managed by an online document-editing program and how the program controls access to documents. Students could explain how a software tool works to others, such as the meaning behind common icons such as the gearwheel or a padlock in a browser search field or operating system.* **Practice(s): 8.1, 8.3**	**Describe the different formats of software components that support common tasks in software tools** *Software tools are built out of standard components such as databases, file systems, network connections and sensors. Students should be able to explain what various components do in general, and how these components are used in specific software systems that are relevant to the student. For example, students should be able to explain that a photo-sharing website has a database with information on users, a file system of photographs, and uses the network to transfer photos taken on a user's smartphone to the application.* **Practice(s): 8.1, 8.3**	**Describe different kinds of computations that software tools perform to tailor a system to individual users.** *Data is at the heart of software tools: tools manage data, but also use data and computations over data to alter how different users experience a system. Students should be able to explain how software tools use data to customize software to individual users. For example, students might explain what user data and searches are used to determine which advertisements are displayed on a news or entertainment website, and why different users see different advertisements or search results for the same search.* **Practice(s): 8.1, 8.3**

RESPONSIBLE COMPUTING IN SOCIETY				
	Grades K-2	**Grades 3-5**	**Grades 6-8**	**Grades 9-12**
Culture	**Compare and contrast how individuals live and work before and after the implementation or adoption of new computing technology.** *Computing technology has positively and negatively changed the way people live and work. In the past, if students wanted to read about a topic, they needed access to a library or any repository to find resources. Today, students can find information on the Internet about a topic or they can download e-books about it directly to a device. Students should be able to describe the type of information found on the Internet. For example, students could go to the library and with teacher or librarian assistance, find books on a particular topic and then find and compare similar information online.* **Practice(s): 3.1**	**Compare and contrast computing technologies that have changed the world, and express how those technologies influence, and are influenced by, cultural practices.** *New computing technology is created and existing technologies are modified for many reasons, such as increased benefits (e.g., Internet search recommendations), decreased risks (e.g., autonomous vehicles), and social efficiencies (smartphone applications). With guidance from their teacher, students should be able to discuss topics that relate to the history of technology and the changes around them that are driven by technology. For example, students could discuss current events that affect them such as robotics, wireless Internet, mobile computing devices, GPS systems, wearable computing, or how social media has influenced social and political movements, changed or affected the practices of cultural traditions and customs.* **Practice(s): 3.1**	**Compare and contrast tradeoffs associated with computing technologies that affect people's everyday activities and career options.** *Advancements in computer technology are neither entirely positive nor negative. However, the ways that people use computing technologies have tradeoffs. Students should consider cultural impacts, including privacy, free speech, communication, and automation. For example, students could identify tradeoffs with driverless cars - they can increase convenience and reduce accidents, but they are also susceptible to hacking.* **Practice(s): 7.2**	**Evaluate the ways computing impacts personal, ethical, social, economic, and cultural practices.** *Computing may change (improve or harm) or maintain societal practices. Minimal exposure to computing, limited access to education, and lack of training opportunities magnify systemic problems in society. Students should be able to evaluate the accessibility of a product to a broad group of end users, such as people who lack ubiquitous access to the Internet or who have disabilities. For example, students could identify potential bias during the development and design process to maximize accessibility in website or digital product implementation.* **Practice(s): 1.2**
	No K-2 standard.	**Identify ways to improve the accessibility and usability of technology products for the diverse needs and wants of users.** *The development and modification of computing technology are driven by people's needs and wants and can affect groups differently. Students should be able to identify the needs and wants of diverse end users and purposefully consider potential perspectives of users with different backgrounds, ability levels, points of view, and disabilities. For example, students could consider using both speech and text when they convey information in a game. They could also vary the options or styles in programs they create, knowing that not everyone shares their own tastes.* **Practice(s): 1.2**	**Discuss issues of bias and accessibility in the design of existing technologies.** *Students should test and discuss the usability of various technology tools (e.g., applications, games, and devices) with the teacher's guidance. Facial recognition software, for example, that works better for lighter skin tones was likely developed with a homogeneous testing group and could be improved by sampling a more diverse population. Students should be able to recognize that increasing usability benefits others in addition to the targeted group. For example, students could discuss the different types of users who would benefit from being able to adjust font sizes and color contrast ratios.* **Practice(s): 1.2**	**Design and analyze computational artifacts to reduce bias and equity deficits.** *Biases include incorrect assumptions developers have made about their user base. Equity deficits include minimal exposure to computing, access to education, and training opportunities. Students should begin to identify potential bias during the design process to maximize accessibility in product design and become aware of professionally accepted accessibility standards to evaluate computational artifacts for accessibility.* **Practice(s): 1.2, 6.3**

Culture	**No K-2 standard.**	**No 3-5 standard**.	**No 6-8 standard.**	**Evaluate the impact of equity, access, and influence on the distribution of computing resources in a global society.** *Resources, such as computers, mobile devices and network connectivity require money to acquire and training to use effectively. The lack of financial resources and educators skilled in computing in many regions of the world, such as inner cities and rural areas, places individuals at a disadvantage in a society that values technology. Students should be able to discuss the ways this digital divide places individuals at a disadvantage. For example, students could research how better access to information and/or resources affects a population.* **Practice(s) : 1.2**
Safety, Law, & Ethics	**Discuss ownership and attribution of digital artifacts.** *Most digital artifacts have owners. Students should understand the importance of giving credit to media creators/owners when using their work. For example, students could create a text document with digital images and identify the online source of their images.* **Practice(s): 7.3**	**Incorporate public domain or creative commons media into a digital artifact, and refrain from copying or using material created by others without permission.** *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. Students should consider the licenses on computational artifacts that they wish to use. For example, the license on a downloaded image or audio file may have restrictions that prohibit modification, require attribution, forbid commercial use, or prohibit use entirely.* **Practice(s): 7.3**	**Discuss how laws control use and access to intellectual property, and mandate broad access to information technologies.** *Copyright laws and licensing protect owners of intellectual property. Additionally, laws help ensure that people with various disabilities can access computing technologies. Students should be able to explain the potential consequences of violating intellectual property laws or licensing agreements, and sending inappropriate content. For example, students could discuss how creative commons licenses for images, restrictions on taking photos in museums, or how laws drive school policies on appropriate content sharing between students.* **Practice(s): 7.3**	**Evaluate the impact of intellectual property laws on the use of digital information.** *Intellectual property laws can have mixed effects. They are meant to protect creation and invention, but can also restrict access and usage of digital objects that permit leveraging an invention for wider use. The same tools that are used to enforce copyright laws can be used to censor media. For example, laws enacted to reduce online piracy can restrict file sharing in ways that limit public access to information. Students should be aware of intellectual property laws and their impact on using or publishing digital artifacts, as well as commercial endeavors. For example, students could research and explain how music streaming applications compensate artists, or how patents are meant to protect the interests of innovators, yet can be abused in litigation focused on financial gain.* **Practice(s): 7.3**

Safety, Law, & Ethics	**No K-2 standard.**	**No 3-5 standard.**	**No 6-8 standard.**	**Evaluate the social and economic implications of privacy and free speech in the context of safety, law, or ethics.** *Laws govern many aspects of computing, such as privacy, data, property, information access, and digital identity. International differences in laws and ethics have implications for computing. Students should understand how privacy laws impact education, workplace and recreation. For example, students could review case studies or current events which present an ethical dilemma when an individual's right to privacy is at odds with the safety, security, or well-being of a community.* **Practice(s): 7.3**
Social Interactions	**Work respectfully and responsibly with others online.** *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. Students should be able to provide feedback to others on their work in a kind and respectful manner and tell an adult if others are sharing things they should not share or are treating others in an unkind or disrespectful manner. For example, as students share their work in blogs or other online collaborative spaces (such as the local school department domain), they will avoid sharing information that is inappropriate or would violate theirs or another's privacy.* **Practice(s): 2.1**	**Seek diverse perspectives for the purpose of improving computational artifacts.** *Computing facilitates collaboration and sharing of ideas. Students should benefit from diverse perspectives facilitated by digital collaboration. For example, students could do mutual reviews of each other's projects, or seek feedback from other student groups outside their classroom (at another grade level, or in another school). Specifically, and with guidance from their teacher, students could use video conferencing tools or other online collaborative spaces, such as blogs, wikis, forums, or website comments, to gather feedback from individuals and groups about programming projects or other digital objects they create.* **Practice(s): 1.1**	**Collaborate and strategize with many online contributors when creating a computational or digital artifact.** *Crowdsourcing is gathering services, ideas, or content from a large group of people, especially from an online community. It can be done at the local level (e.g., classroom or school) or global level (e.g., age-appropriate online communities for artifact creators). Students should develop an understanding of soliciting feedback from a wider audience. For example, a group of students could collect and combine images or animations from their community and create a digital mosaic. They could also solicit feedback from many people by sharing their digital artifacts with specific online communities and electronic surveys, then make improvements.* **Practice(s): 2.4, 5.2**	**Use tools and methods for collaboration on a project to increase connectivity between people in different cultures and career fields.** *Human social structures that support education, work and communities have been affected by the ease of communication facilitated by computing. The increased connectivity between people in different cultures and in different career fields has impacted the variety and types of careers that are possible. Students should be able to explore different collaborative tools and methods used to solicit input from team members, classmates, and others, such as participation in online forums or compiling survey data from local communities. For example, students could compare ways different social media tools could help a team to research examples and solicit input that helps solve a community problem.* **Practice(s): 2.4**

Appendix B- Glossary

The glossary includes definitions of terms used in the statements in the standards. Unless indicated, these definitions were *adopted directly* from the K-12 Computer Science Framework. As noted 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 presented to students.

*denotes revision of definition by Rhode Island Computer Science Education Standards Committee

**denotes addition of definition by Rhode Island Computer Science Education Standards Committee

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 non-digital 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]

application **	A combination of software components or programs that enable users to perform tasks, such as interact with digital artifacts, databases, and other users.
artifact *	Something made by a human. See computational artifact for the definition used in computer science.
audience	Expected end users of a computational artifact or system.
authentication	The verification of the identity of a person or process. [FOLDOC]
automate; automation	**Automate**: To link disparate systems and software so that they become self-acting or self-regulation [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 *	An artifact that performs computation over digital information.
computational thinking *	The human ability to solve problems, design systems, and understand human behavior, by drawing on the concepts fundamental to computer science. [Lee,2006]
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 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 non-digital 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 artifact **	An artifact that is stored in a digital format.
digital citizenship	The norms of appropriate, responsible behavior with regard to the use of technology. [MDESE, 2016]
digital collaboration**	Any activity that involves the sharing or modifying of artifacts by multiple users.
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, changes the state of data. [Free Dictionary]
operating system **	Software that enables a user to interact with and organize files and applications in a single machine: distinct from application.
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]
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 can be executed by a computer **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.
software tool **	Software that enables creation of digital artifacts, storage of data, and data formatting.
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 **	Anyone interacting with a digital artifact, software tool, program or application.
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.*

Appendix C- Resources

Below are the resources used by the K-12 Computer Science Framework. As noted in the Framework, some definitions came directly from the sources listed in the glossary, while others were excerpted or adapted to include content relevant to the framework.

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 commit- tee* (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
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/