V.C.

CONNECTICUT STATE BOARD OF EDUCATION
Hartford

TO BE PROPOSED:
June 6, 2018

RESOLVED, That the State Board of Education adopts the Computer Science Teacher Association (CSTA) K-12 Standards and the International Society for Technology Education (ISTE) Standards for Students.

Approved by a vote of _________, this sixth day of June, Two Thousand Eighteen.

Signed: ______________________________
Dr. Dianna R. Wentzell, Secretary
State Board of Education

CONNECTICUT STATE BOARD OF EDUCATION
Hartford

TO: State Board of Education

FROM: Dr. Dianna R. Wentzell, Commissioner of Education

DATE: June 6, 2018

SUBJECT: Adoption of the Computer Science Teacher Association (CSTA) K-12 Standards and the International Society for Technology Education (ISTE) Standards for Students

Executive Summary

Introduction
Today's students are part of a world in which technology is evolving rapidly, forging new fields of study, creating new types of jobs, and requiring new sets of skills. Not only must students understand the *use* of digital tools can help solve tomorrow's problems, they must also learn how to *create* those tools. Students need opportunities to improve their learning by effectively leveraging technology and to build an understanding of the principles and practices of computer science. To better assist students in building the relevant knowledge and skills necessary in this digital age, two sets of technology-related standards were recently revised by national experts: the *CSTA K–12 Standards* and the *ISTE Standards for Students*.

The Association for Computing Machinery, Code.org, Computer Science Teachers Association, Cyber Innovation Center, and National Math and Science Initiative collaborated with states and districts to develop a framework that includes overarching, high-level computer science guidance per grade band. In 2016, the K–12 Computer Science Framework was published promoting a vision in which all students engage in computer science and innovation. The CSTA utilized the *K–12 Computer Science Framework* to revise the national K–12 computer science standards. In July 2017, CSTA released the *CSTA K–12 Computer Science Standards* that provide detailed, measureable student performance expectations at particular grade levels in the discipline of computer science.

The *ISTE Standards for Students* were also updated in 2016. These standards provide a framework for amplifying digital age learning, citizenship, and teaching across the content areas. They are designed to empower student voice and ensure that learning is a student-driven process regardless of the discipline being taught.

History/Background
Computer science and digital citizenship continue to be a priority in Connecticut. In 2016, the State Board of Education (Board) adopted the Position Statement on Computer Science Education for All Students K–12. The position statement outlines the responsibilities for various stakeholders to build a high-quality, comprehensive, and culturally-responsive computer science education program for all Connecticut students.

Additionally, in July 2017, the legislature passed Public Act No. 17-67. This act established a Digital Citizenship, Internet Safety, and Media Literacy Advisory Council (Council) to be chaired by the Department of Education. One responsibility of the Council is to provide recommendations to the Board regarding best practices relating to instruction in digital citizenship, Internet safety, and media literacy. The *ISTE Standards for Students* support the work of the Council and enable students to recognize the rights, responsibilities, and opportunities of living, learning and working in an interconnected digital world.

The *CSTA K–12 Computer Science Standards* and the *Connecticut Computer Science Education Implementation Guidelines* were introduced for review and consideration to the Academic Standards and Assessment subcommittee of the Board on December 11, 2017. The members supported the work of the Connecticut Computer Science Standards Committee and provided feedback. Based on this feedback, the documents were revised to include a glossary, implementation models, and a variety of curriculum resources.

Both the *CSTA K–12 Computer Science Standards* and *ISTE Standards for Students* were presented to the Academic Standards and Assessment subcommittee of the Board on February 26, 2018. At this meeting, the members requested a visual representation to show the distinct differences and possible overlap of these two sets of standards as well as a narrative outlining the need for both sets of standards (Appendix A).

These documents were submitted to the Academic Standards and Assessment Subcommittee on April 4, 2018. Upon review of these materials, the members of the committee recommended that both sets of standards, the *CSTA K–12 Computer Science Standards* and the *ISTE Standards for Students*, and related documents go forward for full Board adoption.

Recommendation
The Connecticut State Department of Education (CSDE) presents the *CSTA K–12 Computer Science Standards* and *ISTE Standards for Students* for review and consideration of adoption.

Next-Steps
The CSDE will continue to work with multiple partners to ensure that when the *CSTA K–12 Computer Science Standards* and *ISTE Standards for Students* are adopted, there is accessible professional learning for computer science education and digital learning for all districts. Additionally, the CSDE will continue to work collaboratively with the Commission for Educational Technology to leverage conference presentation opportunities, as well as online resources such as webinars to educate districts on the *CSTA K–12 Computer Science Standards* and *ISTE Standards for Students*.

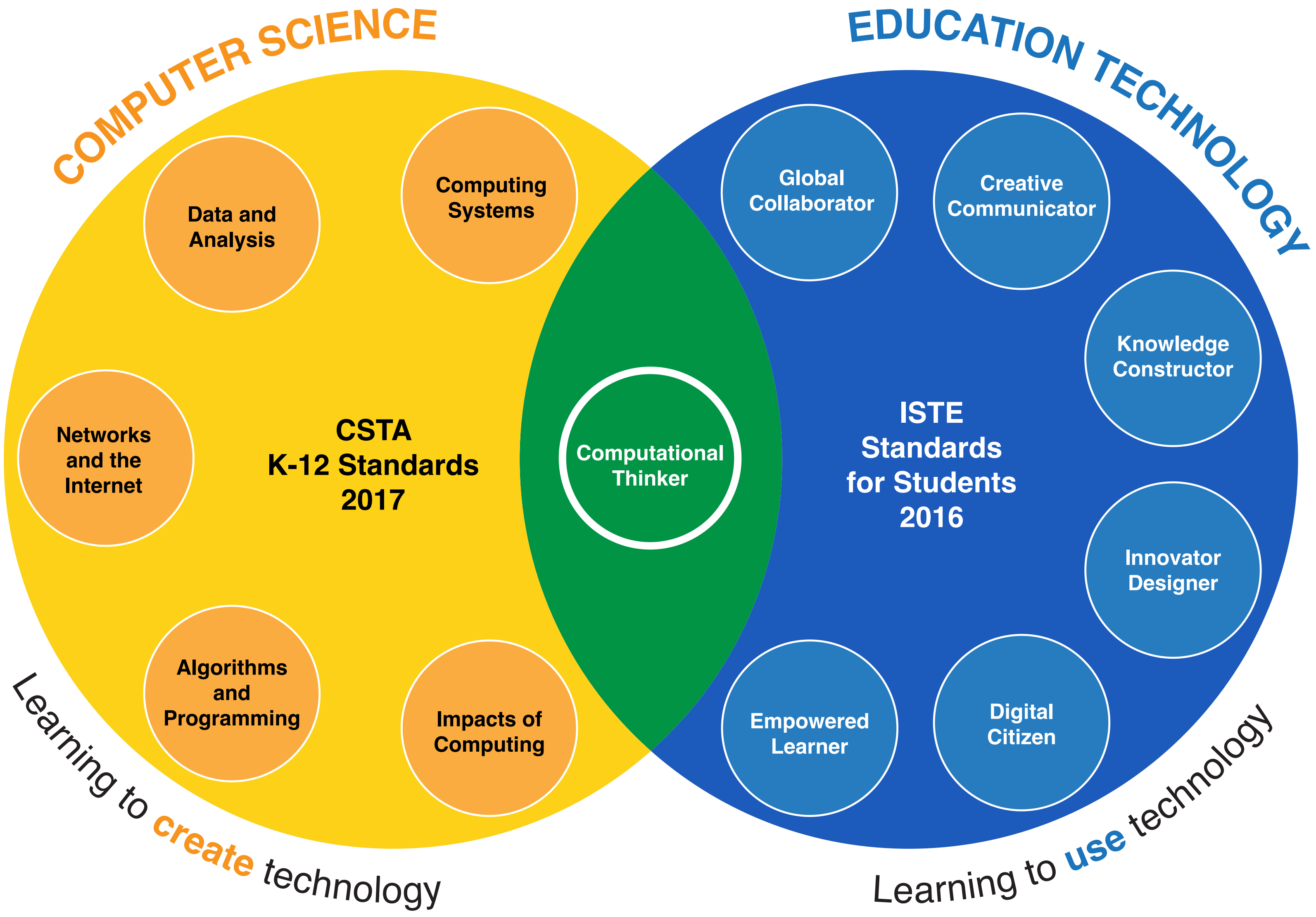
Prepared by: Jennifer Michalek
Education Consultant, Academic Office

Approved by: Melissa K. Wlodarczyk Hickey, Ed.D.
Reading/Literacy Director

Appendix A

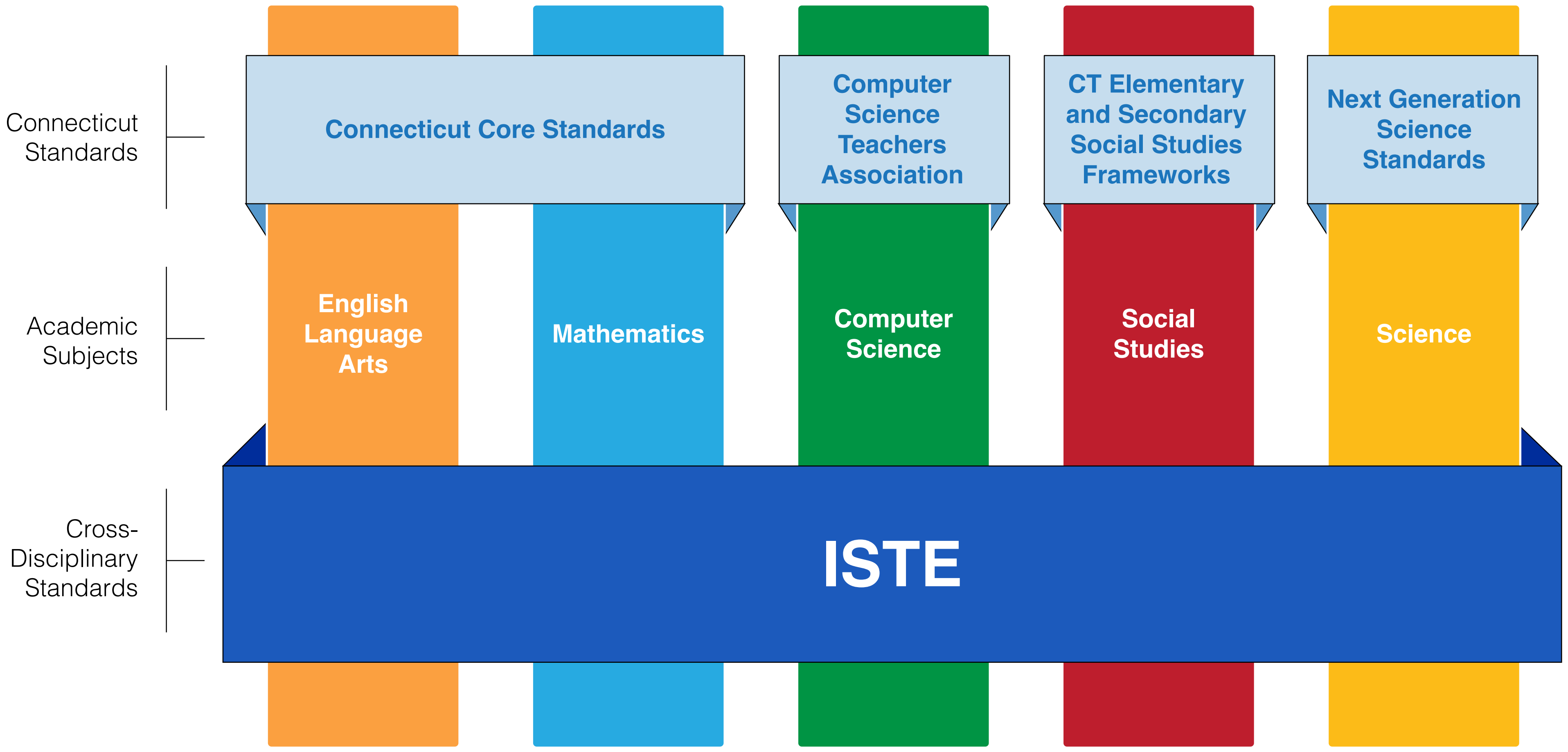
ISTE Standards and Computer Science Standards: Working Together to Best Prepare Today's Students

CONNECTICUT STATE DEPARTMENT OF EDUCATION



ISTE Standards and Computer Science Standards: Technology for Learning and Careers

CSDE
CONNECTICUT STATE DEPARTMENT OF EDUCATION



Connecticut Standards
Academic Subjects
Cross-Disciplinary Standards
Connecticut Core Standards
Computer Science Teachers Association
CT Elementary and Secondary Social Studies Frameworks
Next Generation Science Standards
English Language Arts
Mathematics
Computer Science
Social Studies
Science
ISTE

CSDE
CONNECTICUT STATE
DEPARTMENT OF EDUCATION

ISTE Standards and Computer Science Standards: Technology for Learning and Careers

The State Board of Education endorses the standards of the International Society for Technology in Education (ISTE) and the Computer Science Teachers Association (CSTA). While the two leverage similar terminology, given that both support current instructional best practices, they differ in significant ways.

The ISTE Standards for Students address how students learn, which includes — but does not depend on — the use of technology. The ISTE Standards define habits of mind and general competencies, which many organizations believe are critical for students to master for life in a digital world, rather than discrete content areas. Educators are encouraged to address the Standards across all academic areas, toward the goal of deepening and scaling learning. Teachers should assess student proficiency in the ISTE Standards as part of general coursework, ideally as part of longitudinal portfolios that demonstrate interdisciplinary mastery of digital learning skills.

The CSTA K–12 Standards provide a comprehensive set of K–12 standards in the academic subject of computer science. They are designed to provide a clear understanding of the principles and practices of computer science as an independent discipline. The standards provide academic coherence between coursework and the rapid growth of computing and technology in the modern world and address the need for an educated workforce that can build and manage technology for the benefit of society.

Schools' adoption of both the ISTE and CSTA Standards will help all students to understand and leverage technology for successful careers, lifelong learning, and citizenship in today's digital world.

	ISTE Standards	**NCSTA Standards**
Standard Type	Habits of Mind, General Competencies	Specific Technical Content
Purpose	Deepen Learning Across All Subjects	Develop Technical Skills
Applications	All Subjects, Standards	Discrete Subject
Used By	All Students, Educators, and Leaders	All Students and Teachers of CS
Assessment Type	Cross-Disciplinary, Integrated	Content Specific (e.g., AP Exam)

CSTEACHERS.ORG

COMPUTER SCIENCE TEACHERS ASSOCIATION

K-12 Computer Science Standards, Revised 2017

K12 COMPUTER SCIENCE

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

About the CSTA K-12 Computer Science Standards

Computer science and the technologies it enables rest at the heart of our economy and the way we live our lives. To be well-educated citizens in a computing-intensive world and to be prepared for careers in the 21st century, our students must have a clear understanding of the principles and practices of computer science. The CSTA K–12 Computer Science Standards delineate a core set of learning objectives designed to provide the foundation for a complete computer science curriculum and its implementation at the K–12 level. To this end, the CSTA Standards:

- Introduce the fundamental concepts of computer science to all students, beginning at the elementary school level.
- Present computer science at the secondary school level in a way that can fulfill a computer science, math, or science graduation credit.
- Encourage schools to offer additional secondary-level computer science courses that will allow interested students to study facets of computer science in more depth and prepare them for entry into the work force or college.
- Increase the availability of rigorous computer science for all students, especially those who are members of underrepresented groups.

The standards have been written by educators to be coherent and comprehensible to teachers, administrators, and policy makers.

Levels 1A, 1B, 2, and 3A are the computer science standards for **ALL students**. The Level 3B standards are intended for students who wish to pursue the study of computer science in high school beyond what is required for all students (specialty or elective courses).

Connection to the *K-12 Computer Science Framework*

The K–12 Computer Science Framework (k12cs.org) provides overarching, high-level guidance per grade bands, while the standards provide detailed, measurable student performance expectations. The Framework was considered as a primary input for the standards development process.

The CSTA Standards Revision Task Force crafted standards by combining concept statements and practices from the Framework. It also used descriptive material from the Framework when writing examples and clarifying statements to accompany the standards.

Concepts

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

Practices

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

Level 1A: Grades K-2 (Ages 5-7)

Computing Systems

Identifier	Standard *and Descriptive Statement*	Subconcept	Practice
1A-CS-01	Select and operate appropriate software to perform a variety of tasks, and recognize that users have different needs and preferences for the technology they use. *People use computing devices to perform a variety of tasks accurately and quickly. Students should be able to select the appropriate app/program to use for tasks they are required to complete. For example, if students are asked to draw a picture, they should be able to open and use a drawing app/program to complete this task, or if they are asked to create a presentation, they should be able to open and use presentation software. In addition, with teacher guidance, students should compare and discuss preferences for software with the same primary functionality. Students could compare different web browsers or word processing, presentation, or drawing programs.*	Devices	1.1
1A-CS-02	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.*	Hardware & Software	7.2
1A-CS-03	Describe basic hardware and software problems using accurate terminology. *Problems with computing systems have different causes. Students at this level do not need to understand those causes, but they should be able to communicate a problem with accurate terminology (e.g., when an app or program is not working as expected, a device will not turn on, the sound does not work, etc.). Ideally, students would be able to use simple troubleshooting strategies, including turning a device off and on to reboot it, closing and reopening an app, turning on speakers, or plugging in headphones. These are, however, not specified in the standard, because these problems may not occur.*	Troubleshooting	6.2, 7.2

Networks and the Internet

1A-NI-04	Explain what passwords are and why we use them, and use strong passwords to protect devices and information from unauthorized access. *Learning to protect one's device or information from unwanted use by others is an essential first step in learning about cybersecurity. Students are not required to use multiple strong passwords. They should appropriately use and protect the passwords they are required to use.*	Cybersecurity	7.3

Data and Analysis

1A-DA-05	Store, copy, search, retrieve, modify, and delete information using a computing device and define the information stored as data. *All information stored and processed by a computing device is referred to as data. Data can be images, text documents, audio files, software programs or apps, video files, etc. As students use software to complete tasks on a computing device, they will be manipulating data.*	Storage	4.2
1A-DA-06	Collect and present the same data in various visual formats. *The collection and use of data about the world around them is a routine part of life and influences how people live. Students could collect data on the weather, such as sunny days versus rainy days, the temperature at the beginning of the school day and end of the school day, or the inches of rain over the course of a storm. Students could count the number of pieces of each color of candy in a bag of candy, such as Skittles or M&Ms. Students could create surveys of things that interest them, such as favorite foods, pets, or TV shows, and collect answers to their surveys from their peers and others. The data collected could then be organized into two or more visualizations, such as a bar graph, pie chart, or pictograph.*	Collection Visualization & Transformation	7.1, 4.4
1A-DA-07	Identify and describe patterns in data visualizations, such as charts or graphs, to make predictions. *Data can be used to make inferences or predictions about the world. Students could analyze a graph or pie chart of the colors in a bag of candy or the averages for colors in multiple bags of candy, identify the patterns for which colors are most and least represented, and then make a prediction as to which colors will have most and least in a new bag of candy. Students could analyze graphs of temperatures taken at the beginning of the school day and end of the school day, identify the patterns of when temperatures rise and fall, and predict if they think the temperature will rise or fall at a particular time of the day, based on the pattern observed.*	Inference & Models	4.1

Algorithms and Programming

1A-AP-08	Model daily processes by creating and following algorithms (sets of step-by-step instructions) to complete tasks. *Composition is the combination of smaller tasks into more complex tasks. Students could create and follow algorithms for making simple foods, brushing their teeth, getting ready for school, participating in clean-up time.*	Algorithms	4.4

1A-AP-09	Model the way programs store and manipulate data by using numbers or other symbols to represent information. *Information in the real world can be represented in computer programs. 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.*	Variables	4.4
1A-AP-10	Develop programs with sequences and simple loops, to express ideas or address a problem. *Programming is used as a tool to create products that reflect a wide range of interests. 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 task desired.* *Loops allow for the repetition of a sequence of code multiple times. For example, in a program to show the life cycle of a butterfly, a loop could be combined with move commands to allow continual but controlled movement of the character.*	Control	5.2
1A-AP-11	Decompose (break down) the steps needed to solve a problem into a precise sequence of instructions. *Decomposition is the act of breaking down tasks into simpler tasks. Students could break down the steps needed to make a peanut butter and jelly sandwich, to brush their teeth, to draw a shape, to move a character across the screen, or to solve a level of a coding app.*	Modularity	3.2
1A-AP-12	Develop plans that describe a program's sequence of events, goals, and expected outcomes. *Creating a plan for what a program will do clarifies the steps that will be needed to create a program and can be used to check if a program is correct. Students could create a planning document, such as a story map, a storyboard, or a sequential graphic organizer, to illustrate what their program will do. Students at this stage may complete the planning process with help from their teachers.*	Program Development	5.1, 7.2
1A-AP-13	Give attribution when using the ideas and creations of others while developing programs. *Using computers comes with a level of responsibility. Students should credit artifacts that were created by others, such as pictures, music, and code. Credit could be given orally, if presenting their work to the class, or in writing or orally, if sharing work on a class blog or website. Proper attribution at this stage does not require a formal citation, such as in a bibliography or works cited document.*	Program Development	7.3

1A-AP-14	Debug (identify and fix) 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 changing the sequence of the steps, following the algorithm in a step-by-step manner, or trial and error to fix problems in algorithms and programs.*	Program Development	6.2
1A-AP-15	Using correct terminology, describe steps taken and choices made during the iterative process of program development. *At this stage, students should be able to talk or write about the goals and expected outcomes of the programs they create and the choices that they made when creating programs. This could be done using coding journals, discussions with a teacher, class presentations, or blogs.*	Program Development	7.2

Impacts of Computing

1A-IC-16	Compare how people 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 to find a book about it. Today, students can view and read information on the Internet about a topic or they can download e-books about it directly to a device. Such information may be available in more than one language and could be read to a student, allowing for great accessibility.*	Culture	7
1A-IC-17	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 could share their work on blogs or in other collaborative spaces online, taking care to avoid sharing information that is inappropriate or that could personally identify them to others. Students could provide feedback to others on their work in a kind and respectful manner and could tell an adult if others are sharing things they should not share or are treating others in an unkind or disrespectful manner on online collaborative spaces.*	Social Interactions	2.1
1A-IC-18	Keep login information private, and log off of devices appropriately. *People use computing technology in ways that can help or hurt themselves or others. Harmful behaviors, such as sharing private* *information and leaving public devices logged in should be recognized and avoided.*	Safety Law & Ethics	7.3

Level 1B: Grades 3-5 (Ages 8-11)

Computing Systems

Identifier	Standard *and Descriptive Statement*	Subconcept	Practice
1B-CS-01	Describe how internal and external parts of computing devices function to form a system. *Computing devices often depend on other devices or components. For example, a robot depends on a physically attached light sensor to detect changes in brightness, whereas the light sensor depends on the robot for power. Keyboard input or a mouse click could cause an action to happen or information to be displayed on a screen; this could only happen because the computer has a processor to evaluate what is happening externally and produce corresponding responses. Students should describe how devices and components interact using correct terminology.*	Devices	7.2
1B-CS-02	Model how computer hardware and software work together as a system to accomplish tasks. *In order for a person to accomplish tasks with a computer, both hardware and software are needed. At this stage, a model should only include the basic elements of a computer system, such as input, output, processor, sensors, and storage. Students could draw a model on paper or in a drawing program, program an animation to demonstrate it, or demonstrate it by acting this out in some way.*	Hardware & Software	4.4
1B-CS-03	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 problems such as the device not responding, no power, no network, app crashing, no sound, or password entry not working. Should errors occur at school, the goal would be that students would use various strategies, such as rebooting the device, checking for power, checking network availability, closing and reopening an app, making sure speakers are turned on or headphones are plugged in, and making sure that the caps lock key is not on, to solve these problems, when possible.*	Troubleshooting	6.2

Networks and the Internet

1B-NI-04	Model how information is broken down into smaller pieces, transmitted as packets through multiple devices over networks and the Internet, and reassembled at the destination. *Information is sent and received over physical or wireless paths. It is broken down into smaller pieces called packets, which are sent independently and reassembled at the destination. Students should demonstrate their understanding of this flow of information by, for instance, drawing a model of the way packets are transmitted, programming an animation to show how packets are transmitted, or demonstrating this through an unplugged activity which has them act it out in some way.*	Network Communication & Organization	4.4
1B-NI-05	Discuss real-world cybersecurity problems and how personal information can be protected. *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. Students could discuss or use a journaling or blogging activity to explain, orally or in writing, about topics that relate to personal cybersecurity issues. Discussion topics could be based on current events related to cybersecurity or topics that are applicable to students, such as the necessity of backing up data to guard against loss, how to create strong passwords and the importance of not sharing passwords, or why we should install and keep anti-virus software updated to protect data and systems.*	Cybersecurity	3.1

Data and Analysis

1B-DA-06	Organize and present collected data visually to highlight relationships and support a claim. *Raw data has little meaning on its own. Data is often sorted or grouped to provide additional clarity. Organizing data can make interpreting and communicating it to others easier. Data points can be clustered by a number of commonalities. The same data could be manipulated in different ways to emphasize particular aspects or parts of the data set. For example, a data set of sports teams could be sorted by wins, points scored, or points allowed, and a data set of weather information could be sorted by high temperatures, low temperatures, or precipitation.*	Collection Visualization & Transformation	7.1
1B-DA-07	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 is represented. Inferences or predictions based on data are less likely to be accurate if the data is not sufficient or if the data is incorrect in some way. Students should be able to refer to data when communicating an idea. For example, in order to explore the relationship between speed, time, and distance, students could operate a robot at uniform speed, and at increasing time intervals to predict how far the robot travels at that speed. In order to make an accurate prediction, one or two attempts of differing times would not be enough. The robot may also collect*	Inference & Models	7.1

	temperature data from a sensor, but that data would not be relevant for the task. Students must also make accurate measurements of the distance the robot travels in order to develop a valid prediction. Students could record the temperature at noon each day as a basis to show that temperatures are higher in certain months of the year. If temperatures are not recorded on non-school days or are recorded incorrectly or at different times of the day, the data would be incomplete and the ideas being communicated could be inaccurate. Students may also record the day of the week on which the data was collected, but this would have no relevance to whether temperatures are higher or lower. In order to have sufficient and accurate data on which to communicate the idea, students might want to use data provided by a governmental weather agency.		

Algorithms and Programming

1B-AP-08	Compare and refine multiple algorithms for the same task and determine which is the most appropriate. *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 and plan multiple algorithms to get from one point to another. They could look at routes suggested by mapping software and change the route to something that would be better, based on which route is shortest or fastest or would avoid a problem. Students might compare algorithms that describe how to get ready for school. Another example might be to write different algorithms to draw a regular polygon and determine which algorithm would be the easiest to modify or repurpose to draw a different polygon.*	Algorithms	6.3, 3.3
1B-AP-09	Create programs that use variables to store and modify data. *Variables are used to store and modify data. At this level, understanding how to use variables is sufficient. For example, students may use mathematical operations to add to the score of a game or subtract from the number of lives available in a game. The use of a variable as a countdown timer is another example.*	Variables	5.2
1B-AP-10	Create programs that include sequences, events, loops, and conditionals. *Control structures specify the order (sequence) 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 action. 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 multiplication fact questions and then uses a conditional to check whether or not the answer that was entered is correct. Loops allow for the repetition of a sequence of code multiple times. For example, in a program that produces an animation about a famous historical character, students could use a loop to have the character walk across the screen as they introduce themselves.*	Control	5.2

1B-AP-11	Decompose (break down) problems into smaller, manageable subproblems to facilitate the program development process. *Decomposition is the act of breaking down tasks into simpler tasks. For example, students could create an animation by separating a story into different scenes. For each scene, they would select a background, place characters, and program actions.*	Modularity	3.2
1B-AP-12	Modify, remix, or incorporate portions of an existing program into one's own work, to develop something new or add more advanced features. *Programs can be broken down into smaller parts, which can be incorporated into new or existing programs. For example, students could modify prewritten code from a single-player game to create a two-player game with slightly different rules, remix and add another scene to an animated story, use code to make a ball bounce from another program in a new basketball game, or modify an image created by another student.*	Modularity	5.3
1B-AP-13	Use an iterative process to plan the development of a program by including others' perspectives and considering user preferences. *Planning is an important part of the iterative process of program development. Students outline key features, time and resource constraints, and user expectations. Students should document the plan as, for example, a storyboard, flowchart, pseudocode, or story map.*	Program Development	1.1, 5.1
1B-AP-14	Observe intellectual property rights and give appropriate attribution when creating or remixing programs. *Intellectual property rights can vary by country but copyright laws give the creator of a work a set of rights that prevents others from copying the work and using it in ways that they may not like. Students should identify instances of remixing, when ideas are borrowed and iterated upon, and credit the original creator. Students should also consider common licenses that place limitations or restrictions on the use of computational artifacts, such as images and music downloaded from the Internet. At this stage, attribution should be written in the format required by the teacher and should always be included on any programs shared online.*	Program Development	5.2, 7.3
1B-AP-15	Test and debug (identify and fix errors) a program or algorithm to ensure it runs as intended. *As students develop programs they should continuously test those programs to see that they do what was expected and fix (debug), any errors. Students should also be able to successfully debug simple errors in programs created by others.*	Program Development	6.1, 6.2

1B-AP-16	Take on varying roles, with teacher guidance, when collaborating with peers during the design, implementation, and review stages of program development. *Collaborative computing is the process of performing a computational task by working in pairs or on teams. Because it involves asking for the contributions and feedback of others, effective collaboration can lead to better outcomes than working independently. Students should take turns in different roles during program development, such as note taker, facilitator, program tester, or "driver" of the computer.*	Program Development	2.2
1B-AP-17	Describe choices made during program development using code comments, presentations, and demonstrations. *People communicate about their code to help others understand and use their programs. Another purpose of communicating one's design choices is to show an understanding of one's work. These explanations could manifest themselves 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.*	Program Development	7.2

Impacts of Computing

1B-IC-18	Discuss 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, including to increase their benefits, decrease their risks, and meet societal needs. Students, with guidance from their teacher, should discuss topics that relate to the history of technology and the changes in the world due to technology. Topics could be based on current news content, such as robotics, wireless Internet, mobile computing devices, GPS systems, wearable computing, or how social media has influenced social and political changes.*	Culture	3.1
1B-IC-19	Brainstorm 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. Anticipating the needs and wants of diverse end users requires students to purposefully consider potential perspectives of users with different backgrounds, ability levels, points of view, and disabilities. For example, students may consider using both speech and text when they wish to convey information in a game. They may also wish to vary the types of programs they create, knowing that not everyone shares their own tastes.*	Culture	1.2

1B-IC-20	Seek diverse perspectives for the purpose of improving computational artifacts. *Computing provides the possibility for collaboration and sharing of ideas and allows the benefit of diverse perspectives. For example, students could seek feedback from other groups in their class or students at another grade level. Or, with guidance from their teacher, they 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.*	Social Interactions	1.1
1B-IC-21	Use public domain or creative commons media, 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, or prohibit use entirely.*	Safety Law & Ethics	7.3

Level 2: Grades 6-8 (Ages 11-14)

Computing Systems

Identifier	**Standard** *and Descriptive Statement*	Subconcept	Practice
2-CS-01	Recommend 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 devices, including both hardware and software. Students should make recommendations for existing devices (e.g., a laptop, phone, or tablet) or design their own components or interface (e.g., create their own controllers). Teachers can guide students to consider usability through several lenses, including accessibility, ergonomics, and learnability. For example, assistive devices provide capabilities such as scanning written information and converting it to speech.*	Devices	3.3
2-CS-02	Design projects that combine hardware and software components to collect and exchange data. *Collecting and exchanging data involves input, output, storage, and processing. When possible, students should select the hardware and software components for their project designs by considering factors such as functionality, cost, size, speed, accessibility, and aesthetics. For example, components for a mobile app could include accelerometer, GPS, and speech recognition. The choice of a device that connects wirelessly through a Bluetooth connection versus a physical USB connection involves a tradeoff between mobility and the need for an additional power source for the wireless device.*	Hardware & Software	5.1

2-CS-03	Systematically identify and fix problems with computing devices and their components. *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. Just as pilots use checklists to troubleshoot problems with aircraft systems, students should use a similar, structured process to troubleshoot problems with computing systems and ensure that potential solutions are not overlooked. Examples of troubleshooting strategies include following a troubleshooting flow diagram, making changes to software to see if hardware will work, checking connections and settings, and swapping in working components.*	Troubleshooting	6.2

Networks and the Internet

2-NI-04	Model the role of protocols in transmitting data across networks and the Internet. *Protocols are rules that define how messages between computers are sent. They determine how quickly and securely information is transmitted across networks and the Internet, as well as how to handle errors in transmission. Students should model how data is sent using protocols to choose the fastest path, to deal with missing information, and to deliver sensitive data securely. For example, students could devise a plan for resending lost information or for interpreting a picture that has missing pieces. The priority at this grade level is understanding the purpose of protocols and how they enable secure and errorless communication. Knowledge of the details of how specific protocols work is not expected.*	Network Communication & Organization	4.4
2-NI-05	Explain how physical and digital security measures protect electronic information. *Information that is stored online is vulnerable to unwanted access. Examples of 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. Examples of digital security measures include secure router admin passwords, firewalls that limit access to private networks, and the use of a protocol such as HTTPS to ensure secure data transmission.*	Cybersecurity	7.2
2-NI-06	Apply multiple methods of encryption to model the 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 they should understand the different levels of complexity used to hide or secure information. For example, students could secure messages using methods such as Caesar cyphers 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.*	Cybersecurity	4.4

Data and Analysis

2-DA-07	Represent data using multiple encoding schemes. *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). Students should represent the same data in multiple ways. For example, students could represent the same color using binary, RGB values, hex codes (low-level representations), as well as forms understandable by people, including words, symbols, and digital displays of the color (high-level representations).*	Storage	4
2-DA-08	Collect data using computational tools and transform the data to make it more useful and reliable. *As students continue to build on their ability to organize and present data visually to support a claim, they will need to understand when and how to transform data for this purpose. Students should transform data to remove errors, highlight or expose relationships, and/or make it easier for computers to process. The cleaning of data is an important transformation for ensuring consistent format and reducing noise and errors (e.g., removing irrelevant responses in a survey). An example of a transformation that highlights a relationship is representing males and females as percentages of a whole instead of as individual counts.*	Collection Visualization & Transformation	6.3
2-DA-09	Refine computational models based on the data they have generated. *A model may be a programmed simulation of events or a representation of how various data is related. In order to refine a model, students need to consider which data points are relevant, how data points relate to each other, and if the data is accurate. 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). Additionally, students could refine game mechanics based on test outcomes in order to make the game more balanced or fair.*	Inference & Models	5.3, 4.4

Algorithms and Programming

2-AP-10	Use flowcharts and/or pseudocode to address complex problems as algorithms. *Complex problems are problems that would be difficult for students to solve computationally. Students should use pseudocode and/or flowcharts to organize and sequence an algorithm that addresses a complex problem, even though they may not actually program the solutions. For example, students might express an algorithm that produces a recommendation for purchasing sneakers based on inputs such as size, colors, brand, comfort, and cost. Testing the algorithm with a wide range of inputs and users allows students to refine their recommendation algorithm and to identify other inputs they may have initially excluded.*	Algorithms	4.4, 4.1

2-AP-11	Create clearly named variables that represent different data types and perform operations on their values. *A variable is like a container with a name, in which the contents may change, but the name (identifier) does not. 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. Examples of operations include adding points to the score, combining user input with words to make a sentence, changing the size of a picture, or adding a name to a list of people.*	Variables	5.1, 5.2
2-AP-12	Design and iteratively develop programs that combine control structures, including nested loops and compound conditionals. *Control structures can be combined in many ways. Nested loops are loops placed within loops. 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 use a compound conditional within a loop to unlock a door only if a character has a key AND is touching the door.*	Control	5.1, 5.2
2-AP-13	Decompose problems and subproblems into parts to facilitate the design, implementation, and review of programs. *Students should break down problems into subproblems, which can be further broken down to smaller parts. 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. For example, animations can be decomposed into multiple scenes, which can be developed independently.*	Modularity	3.2
2-AP-14	Create procedures with parameters to organize code and make it easier to reuse. *Students should create procedures and/or functions that are used multiple times within a program to repeat groups of instructions. These procedures can be generalized by defining parameters that create different outputs for a wide range of inputs. For example, a procedure to draw a circle involves many instructions, but all of them can be invoked with one instruction, such as "drawCircle." By adding a radius parameter, the user can easily draw circles of different sizes.*	Modularity	4.1, 4.3

2-AP-15	Seek and incorporate feedback from team members and users to refine a solution that meets user needs. *Development teams that employ user-centered design create solutions (e.g., programs and devices) that can have a large societal impact, such as an app that allows people with speech difficulties to translate hard-to-understand pronunciation into understandable language. Students should begin to seek diverse perspectives throughout the design process to improve their computational artifacts. Considerations of the end-user may include usability, accessibility, age-appropriate content, respectful language, user perspective, pronoun use, color contrast, and ease of use.*	Program Development	2.3, 1.1
2-AP-16	Incorporate existing code, media, and libraries into original programs, and give attribution. *Building on the work of others enables students to produce more interesting and powerful creations. Students should use portions of code, algorithms, and/or digital media in their own programs and websites. At this level, they may also import libraries and connect to web application program interfaces (APIs). For example, when creating a side-scrolling game, students may incorporate portions of code that create a realistic jump movement from another person's game, and they may also import Creative Commons-licensed images to use in the background. Students should give attribution to the original creators to acknowledge their contributions.*	Program Development	4.2, 5.2, 7.3
2-AP-17	Systematically test and refine programs using a range of test cases. *Use cases and test cases are created and analyzed to better meet the needs of users and to evaluate whether programs function as intended. At this level, testing should become a deliberate process that is more iterative, systematic, and proactive than at lower levels. Students should begin to test programs by considering potential errors, such as what will happen if a user enters invalid input (e.g., negative numbers and 0 instead of positive numbers).*	Program Development	6.1
2-AP-18	Distribute tasks and maintain a project timeline when collaboratively developing computational artifacts. *Collaboration is a common and crucial practice in programming development. Often, many individuals and groups work on the interdependent parts of a project together. Students should assume pre-defined roles within their teams and manage the project workflow using structured timelines. With teacher guidance, they will begin to create collective goals, expectations, and equitable workloads. For example, students may divide the design stage of a game into planning the storyboard, flowchart, and different parts of the game mechanics. They can then distribute tasks and roles among members of the team and assign deadlines.*	Program Development	2.2

2-AP-19	Document programs 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 explains their artifacts and how they function. For example, students could provide a project overview and clear user instructions. They should also incorporate comments in their product and communicate their process using design documents, flowcharts, and presentations.*	Program Development	7.2

Impacts of Computing

2-IC-20	Compare tradeoffs associated with computing technologies that affect people's everyday activities and career options. *Advancements in computer technology are neither wholly positive nor negative. However, the ways that people use computing technologies have tradeoffs. Students should consider current events related to broad ideas, including privacy, communication, and automation. For example, driverless cars can increase convenience and reduce accidents, but they are also susceptible to hacking. The emerging industry will reduce the number of taxi and shared-ride drivers, but will create more software engineering and cybersecurity jobs.*	Culture	7.2
2-IC-21	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., apps, games, and devices) with the teacher's guidance. For example, facial recognition software 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. When discussing accessibility, students may notice that allowing a user to change font sizes and colors will not only make an interface usable for people with low vision but also benefits users in various situations, such as in bright daylight or a dark room.*	Culture	1.2
2-IC-22	Collaborate with many contributors through strategies such as crowdsourcing or surveys when creating a computational artifact. *Crowdsourcing is gathering services, ideas, or content from a large group of people, especially from the online community. It can be done at the local level (e.g., classroom or school) or global level (e.g., age-appropriate online communities, like Scratch and Minecraft). For example, a group of students could combine animations to create a digital community mosaic. They could also solicit feedback from many people though use of online communities and electronic surveys.*	Social Interactions	2.4, 5.2

2-IC-23	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. For example, it allows artists and designers to display their talents and reach a broad audience. However, security attacks often start with personal information that is publicly available online. Social engineering is based on tricking people into revealing sensitive information and can be thwarted by being wary of attacks, such as phishing and spoofing.*	Safety Law & Ethics	7.2

Level 3A: Grades 9-10 (Ages 14-16)

Computing Systems

Identifier	**Standard** *and Descriptive Statement*	Subconcept	Practice
3A-CS-01	Explain how abstractions hide the underlying implementation details of computing systems embedded in everyday objects. *Computing devices are often integrated with other systems, including biological, mechanical, and social systems. A medical device can be embedded inside a person to monitor and regulate his or her health, a hearing aid (a type of assistive device) can filter out certain frequencies and magnify others, a monitoring device installed in a motor vehicle can track a person's driving patterns and habits, and a facial recognition device can be integrated into a security system to identify a person. The creation of integrated or embedded systems is not an expectation at this level. Students might 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) it includes, and explain how the implementation details are hidden from the user.*	Devices	4.1
3A-CS-02	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. Multiple layers of software are built upon the hardware and interact with the layers above and below them to reduce complexity. System software manages a computing device's resources so that software can interact with hardware. For example, text editing software interacts with the operating system to receive input from the keyboard, convert the input to bits for storage, and interpret the bits as readable text to display on the monitor. System software is used on many different types of devices, such as smart TVs, assistive devices, virtual components, cloud components, and drones. For example, students may explore the progression from voltage to binary signal to logic gates to adders and so on. Knowledge of specific, advanced terms for computer architecture, such as BIOS, kernel, or bus, is not expected at this level.*	Hardware & Software	4.1

3A-CS-03	Develop guidelines that convey systematic troubleshooting strategies that 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. Examples of complex troubleshooting strategies include resolving connectivity problems, adjusting system configurations and settings, ensuring hardware and software compatibility, and transferring data from one device to another. Students could create a flow chart, a job aid for a help desk employee, or an expert system.*	Troubleshooting	6.2

Networks and the Internet

3A-NI-04	Evaluate the scalability and reliability of networks, by describing the relationship between routers, switches, servers, topology, and addressing. *Each device is assigned an address that uniquely identifies it on the network. Routers function by comparing IP addresses to determine the pathways packets should take to reach their destination. Switches function by comparing MAC addresses to determine which computers or network segments will receive frames. Students could use online network simulators to experiment with these factors.*	Network Communication & Organization	4.1
3A-NI-05	Give examples to illustrate how sensitive data can be affected by malware and other attacks. *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. Potential security problems, such as denial-of-service attacks, ransomware, viruses, worms, spyware, and phishing, present threats to sensitive data. Students might reflect on case studies or current events in which governments or organizations experienced data leaks or data loss as a result of these types of attacks.*	Network Communication & Organization	7.2
3A-NI-06	Recommend security measures to address various scenarios based on factors such as 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 the feasibility of using computational tools to solve given problems or subproblems, such as through a cost-benefit analysis. Eventually, students should include more factors in their evaluations, such as how efficiency affects feasibility or whether a proposed approach raises ethical concerns.*	Cybersecurity	3.3

3A-NI-07	Compare various security measures, considering tradeoffs between the usability and security of a computing system. *Security measures may include physical security tokens, two-factor authentication, and biometric verification, but choosing security measures involves tradeoffs between the usability and security of the system. The needs of users and the sensitivity of data determine the level of security implemented. Students might discuss computer security policies in place at the local level that present a tradeoff between usability and security, such as a web filter that prevents access to many educational sites but keeps the campus network safe.*	Network Communication & Organization	6.3
3A-NI-08	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. Students could debate issues from the perspective of diverse audiences, including individuals, corporations, privacy advocates, security experts, and government.*	Cybersecurity	7.2

Data and Analysis

3A-DA-09	Translate between different bit representations of real-world phenomena, such as characters, numbers, and images. *For example, convert hexadecimal color codes to decimal percentages, ASCII/Unicode representation, and logic gates.*	Storage	4.1
3A-DA-10	Evaluate the tradeoffs in how data elements are organized and where data is stored. *People make choices about how data elements are organized and where data is stored. These choices affect cost, speed, reliability, accessibility, privacy, and integrity. Students should evaluate whether a chosen solution is most appropriate for a particular problem. Students might consider the cost, speed, reliability, accessibility, privacy, and integrity tradeoffs between storing photo data on a mobile device versus in the cloud.*	Storage	3.3

3A-DA-11	Create interactive data visualizations using software tools to help others better understand real-world phenomena. *People transform, generalize, simplify, and present large data sets in different ways to influence how other people interpret and understand the underlying information. Examples include visualization, aggregation, rearrangement, and application of mathematical operations. People use software tools or programming to create powerful, interactive data visualizations and perform a range of mathematical operations to transform and analyze data. Students should model phenomena as systems, with rules governing the interactions within the system and evaluate these models against real-world observations. For example, flocking behaviors, queueing, or life cycles. Google Fusion Tables can provide access to data visualization online.*	Collection Visualization & Transformation	4.4
3A-DA-12	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 phenomenon 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 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. Students should analyze and evaluate these models against real-world observations. For example, students might create a simple producer–consumer ecosystem model using a programming tool. Eventually, they could progress to creating more complex and realistic interactions between species, such as predation, competition, or symbiosis, and evaluate the model based on data gathered from nature.*	Inference & Models	4.4

Algorithms and Programming

3A-AP-13	Create prototypes that use algorithms to solve computational problems by leveraging prior student knowledge and personal interests. *A prototype is a computational artifact that demonstrates the core functionality of a product or process. Prototypes are useful for getting early feedback in the design process, and can yield insight into the feasibility of a product. 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. Students should develop artifacts in response to a task or a computational problem that demonstrate the performance, reusability, and ease of implementation of an algorithm.*	Algorithms	5.2

3A-AP-14	Use lists to simplify solutions, generalizing computational problems instead of repeatedly using simple variables. *Students should be able to identify common features in multiple segments of code and substitute a single segment that uses lists (arrays) to account for the differences.*	Variables	4.1
3A-AP-15	Justify the selection of specific control structures when tradeoffs involve implementation, readability, and program performance, and explain the benefits and drawbacks of choices made. *Implementation includes the choice of programming language, which affects the time and effort required to create a program. Readability refers to how clear the program is to other programmers and can be improved through documentation. The discussion of performance is limited to a theoretical understanding of execution time and storage requirements; a quantitative analysis is not expected. Control structures at this level may include conditional statements, loops, event handlers, and recursion. For example, students might compare the readability and program performance of iterative and recursive implementations of procedures that calculate the Fibonacci sequence.*	Control	5.2
3A-AP-16	Design and iteratively develop computational artifacts for practical intent, personal expression, or to address a societal issue by using events to initiate instructions. *In this context, relevant computational artifacts include programs, mobile apps, or web apps. Events can be user-initiated, such as a button press, or system-initiated, such as a timer firing. At previous levels, students have learned to create and call procedures. Here, students design procedures that are called by events. Students might create a mobile app that updates a list of nearby points of interest when the device detects that its location has been changed.*	Control	5.2
3A-AP-17	Decompose problems into smaller components through systematic analysis, using constructs such as procedures, modules, and/or objects. *At this level, students should decompose complex problems into manageable subproblems that could potentially be solved with programs or procedures that already exist. For example, students could create an app to solve a community problem by connecting to an online database through an application programming interface (API).*	Control	3.2

3A-AP-18	Create artifacts by using procedures within a program, combinations of data and procedures, or independent but interrelated programs. *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. Complex programs are designed as systems of interacting modules, each with a specific role, coordinating for a common overall purpose. Modules allow for better management of complex tasks. The focus at this level is understanding a program as a system with relationships between modules. The choice of implementation, such as programming language or paradigm, may vary. Students could incorporate computer vision libraries to increase the capabilities of a robot or leverage open-source JavaScript libraries to expand the functionality of a web application.*	Modularity	5.2
3A-AP-19	Systematically design and develop programs for broad audiences by incorporating feedback from users. *Examples of programs could include games, utilities, and mobile applications. Students at lower levels collect feedback and revise programs. At this level, students should do so through a systematic process that includes feedback from broad audiences. Students might create a user satisfaction survey and brainstorm distribution methods that could yield feedback from a diverse audience, documenting the process they took to incorporate selected feedback in product revisions.*	Modularity	5.1
3A-AP-20	Evaluate licenses that limit or restrict use of computational artifacts when using resources such as libraries. *Examples of software licenses include copyright, freeware, and the many open-source licensing schemes. At previous levels, students adhered to licensing schemes. At this level, they should consider licensing implications for their own work, especially when incorporating libraries and other resources. Students might consider two software libraries that address a similar need, justifying their choice based on the library that has the least restrictive license.*	Program Development	7.3
3A-AP-21	Evaluate and refine computational artifacts to make them more usable and accessible. *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 should respond to the changing needs and expectations of end users and improve the performance, reliability, usability, and accessibility of artifacts. For example, students could incorporate feedback from a variety of end users to help guide the size and placement of menus and buttons in a user interface.*	Program Development	6.3

3A-AP-22	Design and develop computational artifacts working in team roles using collaborative tools. *Collaborative tools could be as complex as source code version control system or as simple as a collaborative word processor. Team roles in pair programming are driver and navigator but could be more specialized in larger teams. As programs grow more complex, the choice of resources that aid program development becomes increasingly important and should be made by the students. Students might work as a team to develop a mobile application that addresses a problem relevant to the school or community, selecting appropriate tools to establish and manage the project timeline; design, share, and revise graphical user interface elements; and track planned, in-progress, and completed components.*	Program Development	2.4
3A-AP-23	Document design decisions using text, graphics, presentations, and/or demonstrations in the development of complex programs. *Complex programs are designed as systems of interacting modules, each with a specific role, coordinating for a common overall purpose. These modules can be procedures within a program; combinations of data and procedures; or independent, but interrelated, programs. The development of complex programs is aided by resources such as libraries and tools to edit and manage parts of the program.*	Program Development	7.2

Impacts of Computing

3A-IC-24	Evaluate the ways computing impacts personal, ethical, social, economic, and cultural practices. *Computing may improve, harm, or maintain practices. Equity deficits, such as minimal exposure to computing, access to education, and training opportunities, are related to larger, 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 access to broadband or who have various disabilities. Students should also begin to identify potential bias during the design process to maximize accessibility in product design.*	Culture	1.2
3A-IC-25	Test and refine computational artifacts to reduce bias and equity deficits. *Biases could 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.*	Culture	1.2

3A-IC-26	Demonstrate ways a given algorithm applies to problems across disciplines. *Computation can share features with disciplines such as art and music by algorithmically translating human intention into an artifact. Students should be able to identify real-world problems that span multiple disciplines, such as increasing bike safety with new helmet technology, and that can be solved computationally.*	Culture	3.1
3A-IC-27	Use tools and methods for collaboration on a project to increase connectivity of people in different cultures and career fields. *Many aspects of society, especially careers, have been affected by the degree of communication afforded by computing. The increased connectivity between people in different cultures and in different career fields has changed the nature and content of many careers. Students should explore different collaborative tools and methods used to solicit input from team members, classmates, and others, such as participation in online forums or local communities. For example, students could compare ways different social media tools could help a team become more cohesive.*	Social Interactions	2.4
3A-IC-28	Explain the beneficial and harmful effects that intellectual property laws can have on innovation. *Laws govern many aspects of computing, such as privacy, data, property, information, and identity. These laws can have beneficial and harmful effects, such as expediting or delaying advancements in computing and protecting or infringing upon people's rights. International differences in laws and ethics have implications for computing. For examples, laws that mandate the blocking of some file-sharing websites may reduce online piracy but can restrict the right to access information. Firewalls can be used to block harmful viruses and malware but can also be used for media censorship. Students should be aware of intellectual property laws and be able to explain how they are used to protect the interests of innovators and how patent trolls abuse the laws for financial gain.*	Safety Law & Ethics	7.3
3A-IC-29	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. This automated and nonevident collection can raise privacy concerns, such as social media sites mining an account even when the user is not online. Other examples include surveillance video used in a store to track customers for security or information about purchase habits or the monitoring of road traffic to change signals in real time to improve road efficiency without drivers being aware. Methods and devices for collecting data can differ by the amount of storage required, level of detail collected, and sampling rates.*	Safety Law & Ethics	7.2

3A-IC-30	Evaluate the social and economic implications of privacy in the context of safety, law, or ethics. *Laws govern many aspects of computing, such as privacy, data, property, information, and identity. International differences in laws and ethics have implications for computing. Students might 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 wellbeing of a community.*	Safety Law & Ethics	7.3

Level 3B: Grades 11-12 (Ages 16-18)

Computing Systems

Identifier	**Standard** *and Descriptive Statement*	**Subconcept**	**Practice**
3B-CS-01	Categorize the roles of operating system software. *Examples of roles could include memory management, data storage/retrieval, processes management, and access control.*	Hardware & Software	7.2
3B-CS-02	Illustrate ways computing systems implement logic, input, and output through hardware components. *Examples of components could include logic gates and IO pins.*	Troubleshooting	7.2

Networks and the Internet

3B-NI-03	Describe the issues that impact network functionality (e.g., bandwidth, load, delay, topology). *Recommend use of free online network simulators to explore how these issues impact network functionality.*	Network Communication & Organization	7.2
3B-NI-04	Compare ways software developers protect devices and information from unauthorized access. *Examples of security concerns to consider: encryption and authentication strategies, secure coding, and safeguarding keys.*	Cybersecurity	7.2

Data and Analysis

3B-DA-05	Use data analysis tools and techniques to identify patterns in data representing complex systems. *For example, identify trends in a dataset representing social media interactions, movie reviews, or shopping patterns.*	Collection Visualization & Transformation	4.1
3B-DA-06	Select data collection tools and techniques to generate data sets that support a claim or communicate information.	Collection Visualization & Transformation	7.2
3B-DA-07	Evaluate the ability of models and simulations to test and support the refinement of hypotheses.	Inference & Models	4.4

Algorithms and Programming

3B-AP-08	Describe how artificial intelligence drives many software and physical systems. *Examples include digital ad delivery, self-driving cars, and credit card fraud detection.*	Algorithms	7.2
3B-AP-09	Implement an artificial intelligence algorithm to play a game against a human opponent or solve a problem. *Games do not have to be complex. Simple guessing games, Tic-Tac-Toe, or simple robot commands will be sufficient.*	Algorithms	5.3
3B-AP-10	Use and adapt classic algorithms to solve computational problems. *Examples could include sorting and searching.*	Algorithms	4.2
3B-AP-11	Evaluate algorithms in terms of their efficiency, correctness, and clarity. *Examples could include sorting and searching.*	Algorithms	4.2
3B-AP-12	Compare and contrast fundamental data structures and their uses. *Examples could include strings, lists, arrays, stacks, and queues.*	Variables	4.2
3B-AP-13	Illustrate the flow of execution of a recursive algorithm.	Control	3.2

3B-AP-14	Construct solutions to problems using student-created components, such as procedures, modules and/or objects. *Object-oriented programming is optional at this level. Problems can be assigned or student-selected.*	Modularity	5.2
3B-AP-15	Analyze a large-scale computational problem and identify generalizable patterns that can be applied to a solution. *As students encounter complex, real-world problems that span multiple disciplines or social systems, they should decompose complex problems into manageable subproblems that could potentially be solved with programs or procedures that already exist. For example, students could create an app to solve a community problem by connecting to an online database through an application programming interface (API).*	Modularity	4.1
3B-AP-16	Demonstrate code reuse by creating programming solutions using libraries and APIs. *Libraries and APIs can be student-created or common graphics libraries or maps APIs, for example.*	Modularity	5.3
3B-AP-17	Plan and develop programs for broad audiences using a software life cycle process. *Processes could include agile, spiral, or waterfall.*	Program Development	5.1
3B-AP-18	Explain security issues that might lead to compromised computer programs. *For example, common issues include lack of bounds checking, poor input validation, and circular references.*	Program Development	7.2
3B-AP-19	Develop programs for multiple computing platforms. *Example platforms could include: computer desktop, web, or mobile.*	Program Development	5.2
3B-AP-20	Use version control systems, integrated development environments (IDEs), and collaborative tools and practices (code documentation) in a group software project. *Group software projects can be assigned or student-selected.*	Program Development	2.4
3B-AP-21	Develop and use a series of test cases to verify that a program performs according to its design specifications. *At this level, students are expected to select their own test cases.*	Program Development	6.1
3B-AP-22	Modify an existing program to add additional functionality and discuss intended and unintended implications (e.g., breaking other functionality). *For instance, changes made to a method or function signature could break invocations of that method elsewhere in a system.*	Program Development	5.3

3B-AP-23	Evaluate key qualities of a program through a process such as a code review. *Examples of qualities could include correctness, usability, readability, efficiency, portability and scalability.*	Program Development	6.3
3B-AP-24	Compare multiple programming languages and discuss how their features make them suitable for solving different types of problems. *Examples of features include blocks versus text, indentation versus curly braces, and high-level versus low-level.*	Program Development	7.2

Impacts of Computing

3B-IC-25	Evaluate computational artifacts to maximize their beneficial effects and minimize harmful effects on society.	Culture	6.1, 1.2
3B-IC-26	Evaluate the impact of equity, access, and influence on the distribution of computing resources in a global society.	Culture	1.2
3B-IC-27	Predict how computational innovations that have revolutionized aspects of our culture might evolve. *Areas to consider might include education, healthcare, art/entertainment, and energy.*	Culture	7.2
3B-IC-28	Debate laws and regulations that impact the development and use of software.	Safety Law & Ethics	3.3, 7.3

Glossary

The glossary includes definitions of terms used in the standards. These terms are defined for readers of the standards and are not necessarily intended to be the definitions or terms that are seen by students.

Term	Definition
abstraction	**(process):** The process of reducing complexity by focusing on the main idea. By hiding details irrelevant to the question at hand and bringing together related and useful details, abstraction reduces complexity and allows one to focus on the problem. **(product):** A new representation of a thing, a system, or a problem that helpfully reframes a problem by hiding details irrelevant to the question at hand. [MDESE, 2016]
accessibility	The design of products, devices, services, or environments for people who experience disabilities. Accessibility standards that are generally accepted by professional groups include the Web Content Accessibility Guidelines (WCAG) 2.0 and Accessible Rich Internet Applications (ARIA) standards. [Wikipedia]
algorithm	A step-by-step process to complete a task.
analog	The defining characteristic of data that is represented in a continuous, physical way. Whereas digital data is a set of individual symbols, analog data is stored in physical media, such as the surface grooves on a vinyl record, the magnetic tape of a VCR cassette, or other nondigital media. [Techopedia]
app	A type of application software designed to run on a mobile device, such as a smartphone or tablet computer. Also known as a *mobile application*. [Techopedia]
artifact	Anything created by a human. *See* computational artifact *for the definition used in computer science.*

Term	Definition
audience	Expected end users of a computational artifact or system.
accessibility	The design of products, devices, services, or environments for people who experience disabilities. Accessibility standards that are generally accepted by professional groups include the Web Content Accessibility Guidelines (WCAG) 2.0 and Accessible Rich Internet Applications (ARIA) standards. [Wikipedia]
authentication	The verification of the identity of a person or process. [FOLDOC]
automate; automation	**automate:** To link disparate systems and software so that they become self-acting or self-regulating. [Ross, 2016] **automation:** The process of automating.
Boolean	A type of data or expression with two possible values: *true* and *false*. [FOLDOC]
bug	An error in a software program. It may cause a program to unexpectedly quit or behave in an unintended manner. [Tech Terms] The process of finding and correcting errors (bugs) is called debugging. [Wikipedia]
code	Any set of instructions expressed in a programming language. [MDESE, 2016]
comment	A programmer-readable annotation in the code of a computer program added to make the code easier to understand. Comments are generally ignored by machines. [Wikipedia]

Term	Definition
complexity	The minimum amount of resources, such as memory, time, or messages, needed to solve a problem or execute an algorithm. [NIST/DADS]
component	An element of a larger group. Usually, a component provides a particular service or group of related services. [Tech Terms, TechTarget]
computational	Relating to computers or computing methods.
computational artifact	Anything created by a human using a computational thinking process and a computing device. A computational artifact can be, but is not limited to, a program, image, audio, video, presentation, or web page file. [College Board, 2016]
computational thinking	The human ability to formulate problems so that their solutions can be represented as computational steps or algorithms to be executed by a computer. [Lee, 2016]
computer	A machine or device that performs processes, calculations, and operations based on instructions provided by a software or hardware program. [Techopedia]
computer science	The study of computers and algorithmic processes, including their principles, their hardware and software designs, their implementation, and their impact on society. [ACM, 2006]
computing	Any goal-oriented activity requiring, benefiting from, or creating algorithmic processes. [MDESE, 2016]

Term	Definition
computing device	A physical device that uses hardware and software to receive, process, and output information. Computers, mobile phones, and computer chips inside appliances are all examples of computing devices.
computing system	A collection of one or more computers or computing devices, together with their hardware and software, integrated for the purpose of accomplishing shared tasks. Although a computing system can be limited to a single computer or computing device, it more commonly refers to a collection of multiple connected computers, computing devices, and hardware.
conditional	A feature of a programming language that performs different computations or actions depending on whether a programmer-specified Boolean condition evaluates to *true* or *false*. [MDESE, 2016] *(A* conditional *could refer to a* conditional statement, conditional expression, *or* conditional construct.*)*
configuration	**(process):** Defining the options that are provided when installing or modifying hardware and software or the process of creating the configuration (product). [TechTarget] **(product):** The specific hardware and software details that tell exactly what the system is made up of, especially in terms of devices attached, capacity, or capability. [TechTarget]
connection	A physical or wireless attachment between multiple computing systems, computers, or computing devices.
connectivity	A program's or device's ability to link with other programs and devices. [Webopedia]

Term	Definition
control; **control structure**	**control:** *(in general)* The power to direct the course of actions. *(in programming)* The use of elements of programming code to direct which actions take place and the order in which they take place. **control structure:** A programming (code) structure that implements control. Conditionals and loops are examples of control structures.
culture; **cultural practices**	**culture:** A human institution manifested in the learned behavior of people, including their specific belief systems, language(s), social relations, technologies, institutions, organizations, and systems for using and developing resources. [NCSS, 2013] **cultural practices:** The displays and behaviors of a culture.
cybersecurity	The protection against access to, or alteration of, computing resources through the use of technology, processes, and training. [TechTarget]
data	Information that is collected and used for reference or analysis. Data can be digital or nondigital and can be in many forms, including numbers, text, show of hands, images, sounds, or video. [CAS, 2013; Tech Terms]
data structure	A particular way to store and organize data within a computer program to suit a specific purpose so that it can be accessed and worked with in appropriate ways. [TechTarget]
data type	A classification of data that is distinguished by its attributes and the types of operations that can be performed on it. Some common data types are integer, string, Boolean (*true* or *false*), and floating-point.

Term	Definition
debugging	The process of finding and correcting errors (bugs) in programs. [MDESE, 2016]
decompose; decomposition	**decompose:** To break down into components. **decomposition:** Breaking down a problem or system into components. [MDESE, 2016]
device	A unit of physical hardware that provides one or more computing functions within a computing system. It can provide input to the computer, accept output, or both. [Techopedia]
digital	A characteristic of electronic technology that uses discrete values, generally 0 and 1, to generate, store, and process data. [Techopedia]
digital citizenship	The norms of appropriate, responsible behavior with regard to the use of technology. [MDESE, 2016]
efficiency	A measure of the amount of resources an algorithm uses to find an answer. It is usually expressed in terms of the theoretical computations, the memory used, the number of messages passed, the number of disk accesses, etc. [NIST/DADS]
encapsulation	The technique of combining data and the procedures that act on it to create a type. [FOLDOC]
encryption	The conversion of electronic data into another form, called ciphertext, which cannot be easily understood by anyone except authorized parties. [TechTarget]

Term	Definition
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]

Term	Definition
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.*

Term	Definition
modularity	The characteristic of a software/web application that has been divided *(decomposed)* into smaller modules. An application might have several procedures that are called from inside its main procedure. Existing procedures could be reused by recombining them in a new application. [Techopedia]
module	A software component or part of a program that contains one or more procedures. One or more independently developed modules make up a program. [Techopedia]
network	A group of computing devices (personal computers, phones, servers, switches, routers, etc.) connected by cables or wireless media for the exchange of information and resources.
operation	An action, resulting from a single instruction, that changes the state of data. [Free Dictionary]
packet	The unit of data sent over a network. [Tech Terms]
parameter	A special kind of variable used in a procedure to refer to one of the pieces of data received as input by the procedure. [MDESE, 2016]
piracy	The illegal copying, distribution, or use of software. [TechTarget]
procedure	An independent code module that fulfills some concrete task and is referenced within a larger body of program code. The fundamental role of a procedure is to offer a single point of reference for some small goal or task that the developer or

Term	Definition
	programmer can trigger by invoking the procedure itself. [Techopedia] In this framework, *procedure* is used as a general term that may refer to an actual procedure or a method, function, or module of any other name by which modules are known in other programming languages.
process	A series of actions or steps taken to achieve a particular outcome. [Oxford]
program; programming	**program** *(n)*: A set of instructions that the computer executes to achieve a particular objective. [MDESE, 2016] **program** *(v)*: To produce a program by programming. **programming:** The craft of analyzing problems and designing, writing, testing, and maintaining programs to solve them. [MDESE, 2016]
protocol	The special set of rules used by endpoints in a telecommunication connection when they communicate. Protocols specify interactions between the communicating entities. [TechTarget]
prototype	An early approximation of a final product or information system, often built for demonstration purposes. [TechTarget, Techopedia]
redundancy	A system design in which a component is duplicated, so if it fails, there will be a backup. [TechTarget]

Term	Definition
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.

Term	Definition
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

Term	Definition
	the user. A physical topology is the way they are actually interconnected with wires and cables. [PCMag]
troubleshooting	A systematic approach to problem solving that is often used to find and resolve a problem, error, or fault within software or a computing system. [Techopedia, TechTarget]
user	*See the definition for* end user.
variable	A symbolic name that is used to keep track of a value that can change while a program is running. Variables are not just used for numbers; they can also hold text, including whole sentences (*strings*) or logical values (*true* or *false*). A variable has a data type and is associated with a data storage location; its value is normally changed during the course of program execution. [CAS, 2013; Techopedia] *Note: This definition differs from that used in math.*

References

Some definitions came directly from these sources, while others were excerpted or adapted to include content relevant to this 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 committee* (2nd ed.). New York, NY: Association for Computing Machinery.
CAS, 2013	**Computing At School's Computing in the National Curriculum: A Guide for Primary Teachers** Computing At School. (2013). *Computing in the national curriculum: A guide for primary*

	teachers. Belford, UK: Newnorth Print. 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. 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. 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. 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. 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/software/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/

Progression of Computer Science Teachers Association (CSTA) K-12 Computer Science Standards, Revised 2017

Con cept	Subconcept	Level 1A (Ages 5-7) *By the end of Grade 2, students will be able to...*	Level 1B (Ages 8-11) *By the end of Grade 5, students will be able to...*	Level 2 (Ages 11-14) *By the end of Grade 8, students will be able to...*	Level 3A (Ages 14-16) *By the end of Grade 10, students will be able to...*
Computing Systems	Devices	**1A-CS-01** Select and operate appropriate software to perform a variety of tasks, and recognize that users have different needs and preferences for the technology they use. *(P1.1)*	**1B-CS-01** Describe how internal and external parts of computing devices function to form a system. *(P7.2)*	**2-CS-01** Recommend improvements to the design of computing devices, based on an analysis of how users interact with the devices. *(P3.3)*	**3A-CS-01** Explain how abstractions hide the underlying implementation details of computing systems embedded in everyday objects. *(P4.1)*
	Hardware & Software	**1A-CS-02** Use appropriate terminology in identifying and describing the function of common physical components of computing systems (hardware). *(P7.2)*	**1B-CS-02** Model how computer hardware and software work together as a system to accomplish tasks. *(P4.4)*	**2-CS-02** Design projects that combine hardware and software components to collect and exchange data. *(P5.1)*	**3A-CS-02** Compare levels of abstraction and interactions between application software, system software, and hardware layers. *(P4.1)*
	Troubleshooting	**1A-CS-03** Describe basic hardware and software problems using accurate terminology. *(P6.2, P7.2)*	**1B-CS-03** Determine potential solutions to solve simple hardware and software problems using common troubleshooting strategies. *(P6.2)*	**2-CS-03** Systematically identify and fix problems with computing devices and their components. *(P6.2)*	**3A-CS-03** Develop guidelines that convey systematic troubleshooting strategies that others can use to identify and fix errors. *(P6.2)*
Networks & The Internet	Network Communication & Organization		**1B-NI-04** Model how information is broken down into smaller pieces, transmitted as packets through multiple devices over networks and the Internet, and reassembled at the destination. *(P4.4)*	**2-NI-04** Model the role of protocols in transmitting data across networks and the Internet. *(P4.4)*	**3A-NI-04** Evaluate the scalability and reliability of networks, by describing the relationship between routers, switches, servers, topology, and addressing. *(P4.1)*
	Cybersecurity	**1A-NI-04** Explain what passwords are and why we use them, and use strong passwords to protect devices and information from unauthorized access. *(P7.3)*	**1B-NI-05** Discuss real-world cybersecurity problems and how personal information can be protected. *(P3.1)*	**2-NI-05** Explain how physical and digital security measures protect electronic information. *(P7.2)*	**3A-NI-05** Give examples to illustrate how sensitive data can be affected by malware and other attacks. *(P7.2)*
				2-NI-06 Apply multiple methods of encryption to model the secure transmission of information. *(P4.4)*	**3A-NI-06** Recommend security measures to address various scenarios based on factors such as efficiency, feasibility, and ethical impacts. *(P3.3)*
					3A-NI-07 Compare various security measures, considering tradeoffs between the usability and security of a computing system. *(P6.3)*
					3A-NI-08 Explain tradeoffs when selecting and implementing cybersecurity recommendations. *(P7.2)*
Data & Analysis	Storage	**1A-DA-05** Store, copy, search, retrieve, modify, and delete information using a computing device and define the information stored as data. *(P4.2)*	*Continuation of standard 1A-DA-05*	**2-DA-07** Represent data using multiple encoding schemes. *(P4.0)*	**3A-DA-09** Translate between different bit representations of real-world phenomena, such as characters, numbers, and images. *(P4.1)*
					3A-DA-10 Evaluate the tradeoffs in how data elements are organized and where data is stored. *(P3.3)*
	Collection, Visualization, & Transformation	**1A-DA-06** Collect and present the same data in various visual formats. *(P7.1, P4.4)*	**1B-DA-06** Organize and present collected data visually to highlight relationships and support a claim. *(P7.1)*	**2-DA-08** Collect data using computational tools and transform the data to make it more useful and reliable. *(P6.3)*	**3A-DA-11** Create interactive data visualizations using software tools to help others better understand real-world phenomena. *(P4.4)*
	Inference & Models	**1A-DA-07** Identify and describe patterns in data visualizations, such as charts or graphs, to make predictions. *(P4.1)*	**1B-DA-07** Use data to highlight or propose cause-and-effect relationships, predict outcomes, or communicate an idea. *(P7.1)*	**2-DA-09** Refine computational models based on the data they have generated. *(P5.3, P4.4)*	**3A-DA-12** Create computational models that represent the relationships among different elements of data collected from a phenomenon or process. *(P4.4)*
Algorithms & Programming	Algorithms	**1A-AP-08** Model daily processes by creating and following algorithms (sets of step-by-step instructions) to complete tasks. *(P4.4)*	**1B-AP-08** Compare and refine multiple algorithms for the same task and determine which is the most appropriate. *(P6.3, P3.3)*	**2-AP-10** Use flowcharts and/or pseudocode to address complex problems as algorithms. *(P4.4, P4.1)*	**3A-AP-13** Create prototypes that use algorithms to solve computational problems by leveraging prior student knowledge and personal interests. *(P5.2)*
	Variables	**1A-AP-09** Model the way programs store and manipulate data by using numbers or other symbols to represent information. *(P4.4)*	**1B-AP-09** Create programs that use variables to store and modify data. *(P5.2)*	**2-AP-11** Create clearly named variables that represent different data types and perform operations on their values. *(P5.1, P5.2)*	**3A-AP-14** Use lists to simplify solutions, generalizing computational problems instead of repeatedly using simple variables. *(P4.1)*
	Control	**1A-AP-10** Develop programs with sequences and simple loops, to express ideas or address a problem. *(P5.2)*	**1B-AP-10** Create programs that include sequences, events, loops, and conditionals. *(P5.2)*	**2-AP-12** Design and iteratively develop programs that combine control structures, including nested loops and compound conditionals. *(P5.1, P5.2)*	**3A-AP-15** Justify the selection of specific control structures when tradeoffs involve implementation, readability, and program performance, and explain the benefits and drawbacks of choices made. *(P5.2)*
					3A-AP-16 Design and iteratively develop computational artifacts for practical intent, personal expression, or to address a societal issue by using events to initiate instructions. *(P5.2)*
	Practices	*P1. Fostering an Inclusive Computing Culture* *P2. Collaborating Around Computing*	*P3. Recognizing and Defining Computational Problems* *P4. Developing and Using Abstractions*	*P5. Creating Computational Artifacts* *P6. Testing and Refining Computational Artifacts*	*P7. Communicating About Computing*

Progression of Computer Science Teachers Association (CSTA) K-12 Computer Science Standards, Revised 2017

Concept	Subconcept	Level 1A (Ages 5-7) *By the end of Grade 2, students will be able to...*	Level 1B (Ages 8-11) *By the end of Grade 5, students will be able to...*	Level 2 (Ages 11-14) *By the end of Grade 8, students will be able to...*	Level 3A (Ages 14-16) *By the end of Grade 10, students will be able to...*
Algorithms & Programming *(continued)*	Modularity	**1A-AP-11** Decompose (break down) the steps needed to solve a problem into a precise sequence of instructions. *(P3.2)*	**1B-AP-11** Decompose (break down) problems into smaller, manageable subproblems to facilitate the program development process. *(P3.2)*	**2-AP-13** Decompose problems and subproblems into parts to facilitate the design, implementation, and review of programs. *(P3.2)*	**3A-AP-17** Decompose problems into smaller components through systematic analysis, using constructs such as procedures, modules, and/or objects. *(P3.2)*
			1B-AP-12 Modify, remix, or incorporate portions of an existing program into one's own work, to develop something new or add more advanced features. *(P5.3)*	**2-AP-14** Create procedures with parameters to organize code and make it easier to reuse. *(P4.1, P4.3)*	**3A-AP-18** Create artifacts by using procedures within a program, combinations of data and procedures, or independent but interrelated programs. *(P5.2)*
	Program Development	**1A-AP-12** Develop plans that describe a program's sequence of events, goals, and expected outcomes. *(P5.1, P7.2)*	**1B-AP-13** Use an iterative process to plan the development of a program by including others' perspectives and considering user preferences. *(P1.1, P5.1)*	**2-AP-15** Seek and incorporate feedback from team members and users to refine a solution that meets user needs. *(P2.3, P1.1)*	**3A-AP-19** Systematically design and develop programs for broad audiences by incorporating feedback from users. *(P5.1)*
		1A-AP-13 Give attribution when using the ideas and creations of others while developing programs. *(P7.3)*	**1B-AP-14** Observe intellectual property rights and give appropriate attribution when creating or remixing programs. *(P7.3)*	**2-AP-16** Incorporate existing code, media, and libraries into original programs, and give attribution. *(P4.2, P5.2, P7.3)*	**3A-AP-20** Evaluate licenses that limit or restrict use of computational artifacts when using resources such as libraries. *(P7.3)*
		1A-AP-14 Debug (identify and fix) errors in an algorithm or program that includes sequences and simple loops. *(P6.2)*	**1B-AP-15** Test and debug (identify and fix errors) a program or algorithm to ensure it runs as intended. *(P6.1, P6.2)*	**2-AP-17** Systematically test and refine programs using a range of test cases. *(P6.1)*	**3A-AP-21** Evaluate and refine computational artifacts to make them more usable and accessible. *(P6.3)*
			1B-AP-16 Take on varying roles, with teacher guidance, when collaborating with peers during the design, implementation, and review stages of program development. *(P2.2)*	**2-AP-18** Distribute tasks and maintain a project timeline when collaboratively developing computational artifacts. *(P2.2)*	**3A-AP-22** Design and develop computational artifacts working in team roles using collaborative tools. *(P2.4)*
		1A-AP-15 Using correct terminology, describe steps taken and choices made during the iterative process of program development. *(P7.2)*	**1B-AP-17** Describe choices made during program development using code comments, presentations, and demonstrations. *(P7.2)*	**2-AP-19** Document programs in order to make them easier to follow, test, and debug. *(P7.2)*	**3A-AP-23** Document design decisions using text, graphics, presentations, and/or demonstrations in the development of complex programs. *(P7.2)*
Impacts of Computing	Culture	**1A-IC-16** Compare how people live and work before and after the implementation or adoption of new computing technology. *(P7.0)*	**1B-IC-18** Discuss computing technologies that have changed the world, and express how those technologies influence, and are influenced by, cultural practices. *(P7.1)*	**2-IC-20** Compare tradeoffs associated with computing technologies that affect people's everyday activities and career options. *(P7.2)*	**3A-IC-24** Evaluate the ways computing impacts personal, ethical, social, economic, and cultural practices. *(P1.2)*
			1B-IC-19 Brainstorm ways to improve the accessibility and usability of technology products for the diverse needs and wants of users. *(P1.2)*	**2-IC-21** Discuss issues of bias and accessibility in the design of existing technologies. *(P1.2)*	**3A-IC-25** Test and refine computational artifacts to reduce bias and equity deficits. *(P1.2)*
					3A-IC-26 Demonstrate ways a given algorithm applies to problems across disciplines. *(P3.1)*
	Social Interactions	**1A-IC-17** Work respectfully and responsibly with others online. *(P2.1)*	**1B-IC-20** Seek diverse perspectives for the purpose of improving computational artifacts. *(P1.1)*	**2-IC-22** Collaborate with many contributors through strategies such as crowdsourcing or surveys when creating a computational artifact. *(P2.4, P5.2)*	**3A-IC-27** Use tools and methods for collaboration on a project to increase connectivity of people in different cultures and career fields. *(P2.4)*
	Safety, Law, & Ethics		**1B-IC-21** Use public domain or creative commons media, and refrain from copying or using material created by others without permission. *(P7.3)*		**3A-IC-28** Explain the beneficial and harmful effects that intellectual property laws can have on innovation. *(P7.3)*
		1A-IC-18 Keep login information private, and log off of devices appropriately. *(P7.3)*		**2-IC-23** Describe tradeoffs between allowing information to be public and keeping information private and secure. *(P7.2)*	**3A-IC-29** Explain the privacy concerns related to the collection and generation of data through automated processes that may not be evident to users. *(P7.2)*
					3A-IC-30 Evaluate the social and economic implications of privacy in the context of safety, law, or ethics. *(P7.3)*
	Practices	*P1. Fostering an Inclusive Computing Culture* *P2. Collaborating Around Computing*	*P3. Recognizing and Defining Computational Problems* *P4. Developing and Using Abstractions*	*P5. Creating Computational Artifacts* *P6. Testing and Refining Computational Artifacts*	*P7. Communicating About Computing*

Connecticut Computer Science Implementation Guidelines

2018

CONNECTICUT STATE
DEPARTMENT OF EDUCATION

Connecticut State Department of Education

Contents

Connecticut Computer Science Standards Workgroup

Introduction

The Connecticut State Board of Education (Board) believes that computer science is a key to developing and integrating 21st Century Skills (e.g., technology, communication, collaboration, critical thinking, problem solving, innovation, creativity, persistence). The Board further believes that all Connecticut public schools must provide challenging and rigorous programs of study in computer science across all grade levels. This implementation guidance document articulates the lens through which to view the standards and provides guidance for implementation across the State of Connecticut.

Background

In 2011, the Computer Science Teachers Association (CSTA) developed the first K–12 computer science standards. As computer science continued to influence technology and the world, it became necessary to review the *2011 CSTA K–12 Standards*. While the review of the *2011 CSTA K–12 Standards* was in progress, CSTA joined forces with other computer science organizations (i.e., Association for Computing Machinery, Code.org, Cyber Innovation Center, and National Math + Science Initiative) to develop a K–12 Computer Science Framework. The *K–12 Computer Science Framework* identifies the core areas of computer science necessary in grades K–12. Using the *K–12 Computer Science Framework*, a team of computer science professionals composed of teachers, administrators, and members of industry updated the *2011 CSTA K–12 Standards*. Both the framework and the updated standards underwent three public review period for scrutiny and feedback from anyone with interest or experience in K–12 computer science education. The feedback from each review period was read and addressed by the development team. The updated K–12 computer science standards were released in July 2017.

The 2017 CSTA K–12 Computer Science Standards

The *2017 CSTA K–12 Computer Science Standards* delineate a core set of learning objectives designed to provide the foundation for a complete computer science curriculum. They have been widely received by the computer science education and business communities, as well as policy developers. The standards are currently being used to define computer science in many states across the United States.

The *2017 CSTA K–12 Computer Science Standards*:

- introduce the fundamental concepts of computer science to all students, beginning at the elementary school level;
- present computer science at the secondary school level in a way that can fulfill a computer science, math, or science graduation credit;
- encourage schools to offer additional secondary-level computer science courses that will allow interested students to study facets of computer science at a deeper level, and prepare these students for entry into the workforce or college; and
- increase the availability of rigorous computer science courses for all students, especially those who are members of underrepresented groups.

Computer Science Standards in Connecticut

In September 2017, the Connecticut Department of Education (CSDE) convened a group of educators charged with putting forward computer science standards for Board approval. These educators were divided into grade level teams: K–5, 6–8, and 9–12. Each team independently reviewed the *2017 CSTA K–12 Computer Science Standards*. The review by each team concluded that these 2017 standards aligned to the beliefs contained with the Board's previously adopted *Position Statement on Computer Science Education for All Students K–12*. It was recommended by the teams that feedback from stakeholders be elicited on the standards and that the standards be brought forth to the Board for adoption.

In Connecticut a survey about the standards was disseminated to a variety of stakeholders in January 2018. This survey provided stakeholders the opportunity to give their feedback in regards to the standards. The survey was made publicly available and responses were collected over a six week period. Respondents included teachers, administrators, parents, higher education and business and industry. The results of the survey were favorable for adopting the standards in Connecticut.

Defining Computer Science

Computer science is defined as "the study of computers and algorithmic processes, including their principles, their hardware and software designs, their [implementation], and their impact on society" (Tucker et. al, 2003, p. 6). Thus, computer science is the foundation for all computing. Computer science builds on computer literacy, educational technology, digital citizenship, and information technology. These aspects of computing are distinguished from computer science because they are focused on using computer technologies rather than understanding why computer technologies work and how to create those technologies.

Defining Computational Thinking

Integrated throughout the *2017 CSTA K–12 Computer Science Standards* is the concept of computational thinking. Computational thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent (Cuny, Snyder & Wing, 2010). It is an approach to solving problems in a way that can be implemented with a computer. It involves the use of concepts, such as abstraction, recursion, and iteration, to process and analyze data, and to create real and virtual artifacts (Computer Science Teachers Association & Association for Computing Machinery).

Computational thinking practices such as abstraction, modeling, and decomposition connect with computer science concepts such as algorithms, automation, and data visualization. Beginning with the elementary school grades and continuing through grade 12, students should develop a foundation of computer science knowledge and learn new approaches to problem solving that captures the power of computational thinking to become both users and creators of computing technology.

Equity

Equity is a fundamental component in the development of the *2017 CSTA K–12 Computer Science Standards*. The intent of equity is to ensure that all students have the basic knowledge that will allow them to productively participate in the world and make well informed decisions about their lives. Classrooms often include students of different races, genders, socioeconomic statuses, English learners, students with disabilities, and students with differing ways of learning. Regardless of these differences, all students have the right to high quality computer science education.

Equity is not limited to whether classes are available, but includes how classes are taught, how students are recruited for classes or activities, and how the classroom culture supports diverse learners and promotes continued studies in computer science. The result of equity is achieving the ability to meet the needs of diverse learners and having them feel capable of learning. It ensures that all students have the basic knowledge that will allow them to compete in a diverse world.

Computer Science Practices

The *2017 CSTA K–12 Computer Science Standards* incorporate seven practices. By Grade 12, it is expected that every computationally literate student will engage with these practice behaviors as they learn the standards and develop computational artifacts. The interrelated practices are listed in the chart below in an order that simulates the developmental process taken to produce computational artifacts.

Identifier	**Practice**
P1	Fostering an Inclusive Computing Culture
P1.1	Include the unique perspectives of others and reflect on one's own perspectives when designing and developing computational products
P1.2	Address the needs of diverse end users during the design process to produce artifacts with broad accessibility and usability
P1.3	Employ self- and peer-advocacy to address bias in interactions, product design, and development methods
P2	Collaborating Around Computing
P2.1	Cultivate working relationships with individuals possessing diverse perspectives, skills, and personalities
P2.2	Create team norms, expectations, and equitable workloads to increase efficiency and effectiveness
P2.3	Solicit and incorporate feedback from, and provide constructive feedback to, team members and other stakeholders

P2.4	Evaluate and select technological tools that can be used to collaborate on a project
P3	Recognizing and Defining Computational Problems
P3.1	Identify complex, interdisciplinary, real-world problems that can be solved computationally
P3.2	Decompose complex real-world problems into manageable subproblems that could integrate existing solutions or procedures
P3.3	Evaluate whether it is appropriate and feasible to solve a problem computationally
P4	Developing and Using Abstractions
P4.1	Extract common features from a set of interrelated processes or complex phenomena
P4.2	Evaluate existing technological functionalities and incorporate them into new designs
P4.3	Create modules and develop points of interaction that can apply to multiple situations and reduce complexity
P4.4	Model phenomena and processes and simulate systems to understand and evaluate potential outcomes
P5	Creating Computational Artifacts
P5.1	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
P5.2	Create a computational artifact for practical intent, personal expression, or to address a societal issue
P5.3	Modify an existing artifact to improve or customize it
P6	Testing and Refining Computational Artifacts
P6.1	Systematically test computational artifacts by considering all scenarios and using test cases
P6.2	Identify and fix errors using a systematic process
P6.3	Evaluate and refine a computational artifact multiple times to enhance its performance, reliability, usability, and accessibility
P7	Communicating About Computing
P7.1	Select, organize, and interpret large data sets from multiple sources to support a claim
P7.2	Describe, justify, and document computational processes and solutions using appropriate terminology consistent with the intended audience and purpose

P7.3	Articulate ideas responsibly by observing intellectual property rights and giving appropriate attribution

Organization of the Standards

Grade bands

The *2017 CSTA K–12 Computer Science Standards* are organized into grade bands with the goal being that students will have met the expectations by the end of grade 2 (Level 1A, ages 5–7), the end of grade 5 (Level 1B, ages 8–11), the end of grade 8 (Level 2, ages 11–14) and the end of grade 10 (Level 3A, ages 14–16). Furthermore, for students who wish to study computer science in high school beyond the level required for all students, Level 3B is provided.

Strands

The *2017 CSTA K–12 Computer Science Standards* are also organized into strands called Concepts and Subconcepts. There are five Concepts: Algorithms & Programming, Computing Systems, Data & Analysis, Impacts of Computer, and Networks & the Internet, which are further broken down into sixteen Subconcepts. The chart below provides a brief overview of each sub concept for further clarification. In addition, there are five cross-cutting topics that are interwoven within each core concept throughout the standards, but do not have stand-alone descriptions, including Abstraction, System Relationships, Human- Computer Interaction, User Inspired Software Design, Privacy and Security, and Communication and Coordination. The vertically aligned standards are intended to reflect a comprehensive instructional program and document a progression of expected achievement in each of the strands. This organization of standards also reflects the gradual progression in the development of skills.

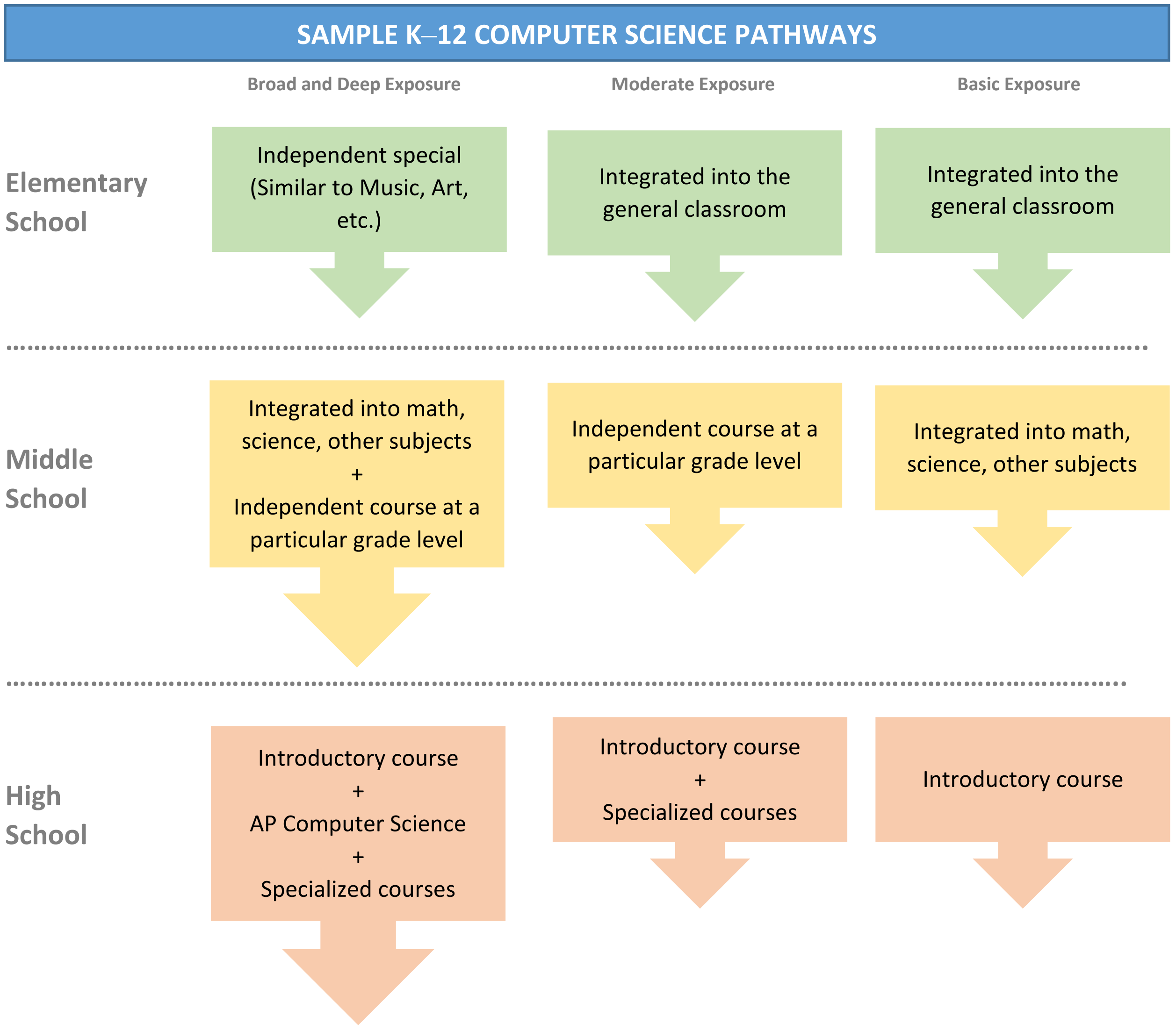
Concept	Sub concept	Overview
Algorithms and Programming	Algorithms	People evaluate and select algorithms based on performance, reusability, and ease of implementation. Knowledge of common algorithms improves how people develop software, secure data, and store information.
	Control	Programmers consider tradeoffs related to implementation, readability, and program performance when selecting and combining control structures.
	Modularity	Complex programs are designed as systems of interacting modules, each with a specific role, coordinating for a common overall purpose. These modules can be procedures within a program; combinations of data and procedures or independent, but interrelated, programs. Modules allow for better management of complex tasks.
	Program Development	Diverse teams can develop programs with broad impact through careful review and by drawing on the strengths of members in different roles. Design decisions often involve tradeoffs. The development of complex programs is aided by resources such as libraries and tools to edit and manage parts of the program. Systematic analysis is critical for identifying the effects of lingering bugs.

Algorithms and Programming	Variables	Data structures are used to manage program complexity. Programmers choose data structures based on functionality, storage, and performance tradeoffs.
Computing Systems	Devices	Many everyday objects contain computational components that sense and act on the world. In early grades, students learn features and applications of common computing devices. As they progress, students learn about connected systems and how interaction between humans and devices influences design decisions.
	Hardware and 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.
Data and Analysis	Collection, Visualization, and Transformation	Data are collected with both computational and non-computational tools and processes. In early grades, students learn how data about themselves and their world is collected and used. As they progress, students learn the effects of collecting data with computational and automated tools.
	Inference and Models	Data science is one example where computer science serves many fields. Computer science and science use data to make inferences, theories, or predictions based upon 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	Data can be composed of multiple data elements that relate to one another. For example, population data may contain information about age, gender, and height. People make choices about how data elements are organized and where data are stored. These choices affect cost, speed, reliability, accessibility, privacy, and integrity.
Impacts of Computing	Culture	The design and use of computing technologies and artifacts can improve, worsen, or maintain inequitable access to information and opportunities.
	Safety, Law and Ethics	Laws govern many aspects of computing, such as privacy, data, property, information, and identity. These laws can have beneficial and harmful effects, such as expediting or delaying advancements in computing and protecting or infringing upon people's rights. International differences in laws and ethics have implications for computing.
	Social Interactions	Many aspects of society, especially careers, have been affected by the degree of communication afforded by computing. The increased connectivity between people in different cultures and in different career fields has changed the nature and content of many careers.

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

Implementation Models

In the following examples, a computer science experience can range from a few hours a week to a semester- or year-long course. Computer science may be integrated into current curriculum or offered as an independent course based on student and district readiness.


SAMPLE K–12 COMPUTER SCIENCE PATHWAYS
Broad and Deep Exposure
Moderate Exposure
Basic Exposure
Elementary School
Independent special (Similar to Music, Art, etc.)
Integrated into the general classroom
Integrated into the general classroom
Middle School
Integrated into math, science, other subjects + Independent course at a particular grade level
Independent course at a particular grade level
Integrated into math, science, other subjects
High School
Introductory course + AP Computer Science + Specialized courses
Introductory course + Specialized courses
Introductory course

Elementary and Middle School

Computer science at the K–2, 3–5, and 6–8 grade bands can be embedded within the curriculum and/or offered as a stand-alone course, depending on the school's program. This flexible implementation allows schools the choice to determine their own timeline on how they will ensure that all students will have the opportunity to learn computer science. All certified staff members and subject areas are encouraged to integrate computer science instruction into their classrooms.

Below are various suggestions for implementation:

- integrate computer science into a particular subject area (i.e., math, science, technology) on a weekly or biweekly basis within elementary classrooms;
- plan districtwide and schoolwide participation in the annual "Hour of Code" for all grade levels;
- offer computer science in a particular grade level and then expand the program to additional grade levels in subsequent years;
- provide small group instruction;
- provide a weekly "specials"/ "unified arts" course designed to specifically teach the computer science standards;
- integrate computer science instruction into existing Library/Media time; and
- incorporate computer science in a similar fashion as Maker Spaces, Genius Hour etc.

High School

Implementation at the high school level is best achieved through course offerings specific to computer science. All high schools should offer at least one rigorous computer science course. Ideally high schools develop computer science pathways for students to explore based on need and interest.

Curriculum and Instruction Resources

Implementing the computer science standards will require many curricula and instructional decisions.

- Schools may decide to teach a specific curriculum or combine multiple resources to deliver computer science instruction.
- Computer science instruction can be implemented with limited access to technology. Students are encouraged to work together and can share devices.
- Computer science can be individualized and collaborative.
- "Unplugged" lessons are available and districts are encouraged to use a combination of "plugged" for an authentic computer science experience.
- 20 hours of Computer Science instruction per year will provide a rigorous and well-developed experience for students at the elementary and middle school levels.
- One credit or its equivalent in computer science at the high school level will best prepare students to be college and career ready.

In an effort to assist districts in making curriculum decisions related to the implementation of computer science a sampling of resources is provided. This is not an all-inclusive list and the resources are not endorsed by CSDE. However, the intent is to provide information so that districts may begin researching options to support computer science implementation in their schools.

Elementary School

Organization	**Curriculum**
Apple	The lessons in the Get Started with Code Teacher Guides, which are part of the Everyone Can Code Curriculum, are designed to help you bring coding into the early primary classroom.
Bee-Bot	Bee-Bot Lessons contains 100 detailed lesson plans, with accompanying images, for using Bee-Bot to teach across the curriculum. Problem-Solving with Bee-Bot provides 150 sequential student challenges that use Bee-Bot to develop problem-solving, critical-thinking, and decision-making skills.
codeSpark Academy	Ignite interest in computer science and turn programming into play.
Code Studio(Code.org)	Computer Science Fundamentals is comprised of 6 courses of about 15 lessons that may be implemented as one unit or over the course of a semester.
Code Monkey	The Code Monkey game is accompanied by a curriculum guide which includes 35 detailed lesson plans with both online and offline activities.
Computer Science for All in SF	Creative Computing Curriculum for K – 2 and 3 – 5 introduces computer science as a creative, collaborative, and engaging discipline across 15 – 20 lessons at each grade level.
Kodable	Courses for every grade K – 5 enabling students to learn foundational skills in computer science preparing them for the next step in their learning.

Organization	Curriculum
Project Lead The Way	PLTW Launch modules engage students and build knowledge and skills in the area of computer science.
ScratchEd	Activities are designed to support familiarity and increasing fluency with computational creativity and computational thinking using Scratch. Units can be used as a semester-long computing course or as part of other curriculum areas.
Tynker	Seven coding courses designed for students K – 5.

Middle School

Organization	Curriculum
Apple	The lessons in the Learn to Code Teacher Guides, which are part of the Everyone Can Code Curriculum, are designed to help students learn fundamental coding concepts.
Bootstrap	Teach algebra through video-game programming, with a module to go alongside or inside a math class.
CodeHS	CodeHS helps schools and districts build a comprehensive Middle School computer science program starting with introductory level block-based programming courses. There are courses available for all grades 6-8.
Code.org	Computer Science Discoveries is an introductory computer science course recommended for grades 6 - 10 that empowers students to create authentic artifacts and engage with computer science as a medium for creativity, communication, problem solving, and fun.

Organization	Curriculum
Code Monkey	The Code Monkey game is accompanied by a curriculum guide which includes 35 detailed lesson plans with both online and offline activities.
Codesters	Range of courses where students use Python to build projects through structured lessons, then modify their code to create custom projects.
Computer Science for All in SF	MyCS intended for grade 6 and App Inventor for grade 7 highlight the personal relevance of computer science to middle school students and attempt to present computer science as a fun, creative, and collaborative discipline.
Edhesive	Explorations in Coding course is specifically designed for middle school classrooms, this blended online course covers foundational concepts and skills of computer science.
Globaloria	Standalone and core subject integration pathways.
GUTS	In partnership with Code.org, Middle School CS in Science includes four modules each consisting of five or six lessons.
Project Lead The Way	PLTW Gateway units include units that engage students in computer science through robotics, hardware and software development, and mobile app development.
Pythonroom	Pythonroom's curriculum is designed to teach students problem-solving and algorithmic thinking while introducing computer science to young students.
ScratchEd	Activities are designed to support familiarity and increasing fluency with computational creativity and computational thinking using Scratch. Units

Organization	Curriculum
ScratchEd	can be used as a semester-long computing course or as part of other curriculum areas.
Tynker	Seven coding courses designed for students 6 - 8.
UC Davis C-STEM	Multiple academic year-long courses on computing in math, programming, robotics, and film production.

High School

Organization	Curriculum
Apple	The Intro to App Development with Swift and App Development with Swift curricula were designed to teach high school students with little or no programming experience how to be app developers, capable of bringing their own ideas to life.
Beauty and Joy of Computing	Introductory computer science curriculum intended for high school juniors and seniors that is aligned to the AP CS Principles course.
Bootstrap	Teach algebra through video-game programming, with a 20-hr module to go alongside or inside a math class.
CodeHS	Four-year high school computer science pathway. Intro CS JavaScript, Intro CS Python, AP CS Principles, AP CS in Java, Computing Ideas, Web Design and more.
Code.org	Two years of Computer Science courses for beginners. The first course, Computer Science Discoveries, is appropriate for grades 6-10 and the second,

Organization	Curriculum
Code.org	Computer Science Principles, can be implemented as an AP course or an introductory course.
Edhesive	Year-long AP Computer Science courses.
Exploring Computer Science	Year-long introductory high school course aimed at broadening participation in computer science.
Globaloria	Standalone and core subject integration pathways.
Mobile CSP	Mobile CSP is a College Board-endorsed AP Computer Science Principles curriculum.
Project Lead The Way	PLTW Computer Science engages students in true-to-life activities like creating an online art portal or developing problem-solving apps.
ScratchEd	Activities are designed to support familiarity and increasing fluency with computational creativity and computational thinking using Scratch. Units can be used as a semester-long computing course or as part of other curriculum areas.
TEALS	TEALS helps high schools build and grow sustainable computer science programs by pairing experienced and trained software engineer professionals with classroom teachers. TEALS has two standard high school course offerings and offers support for additional courses.
UC Davis C-STEM	Multiple academic year-long courses on computing in math, programming, and robotics.

Organization	Curriculum
UTeach CS Principles	A classroom-ready curriculum that is fully aligned with the College Board's AP Computer Science Principles framework and endorsed by the College Board.

Additional resources to support computer science curricula and instruction can be accessed on the Connecticut Computer Science Teachers' Association website.

References

Computer Science Teachers Association (2017). CSTA K–12 Computer Science Standards, Revised 2017. Retrieved from CS Teachers.

"K–12 Computer Science Framework." *k12cs.Org*, Computer Science Teachers Association, k12cs.org/.

Education, Virginia Department of. "Computer Science." *VDOE :: Computer Science Standards of Learning Resources*, Virginia Department of Education, Nov. 2017, www.doe.virginia.gov/testing/sol/standards_docs/computer-science/index.shtml.

"Anybody Can Learn." *Code.org*, Code.org, code.org/.

"CSTA." *CSTA*, csteachers.org/.

"Proposed Nevada K–12 Computer Science Standards." www.doe.nv.gov/uploadedFiles/nde.doe.nv.gov/content/Standards_Instructional_Support/Nevada_Academic_Standards/Comp_Tech_Standards/DRAFTNevadaK–12ComputerScienceStandards.pdf.

"3rd Party Educator Resources." *CSEd Week*, csedweek.org/educate/curriculum/3rd-party.

"ISTE - International Society for Technology in Education - Home." *ISTE - International Society for Technology in Education - Home*, www.iste.org

2016

ISTE STANDARDS FOR STUDENTS

The 2016 ISTE Standards for Students emphasize the skills and qualities we want for students, enabling them to engage and thrive in a connected, digital world. The standards are designed for use by educators across the curriculum, with every age student, with a goal of cultivating these skills throughout a student's academic career. Both students and teachers will be responsible for achieving foundational technology skills to fully apply the standards. The reward, however, will be educators who skillfully mentor and inspire students to amplify learning with technology and challenge them to be agents of their own learning.

1. Empowered Learner

Students leverage technology to take an active role in choosing, achieving and demonstrating competency in their learning goals, informed by the learning sciences. Students:

a. articulate and set personal learning goals, develop strategies leveraging technology to achieve them and reflect on the learning process itself to improve learning outcomes.
b. build networks and customize their learning environments in ways that support the learning process.
c. use technology to seek feedback that informs and improves their practice and to demonstrate their learning in a variety of ways.
d. understand the fundamental concepts of technology operations, demonstrate the ability to choose, use and troubleshoot current technologies and are able to transfer their knowledge to explore emerging technologies.

2. Digital Citizen

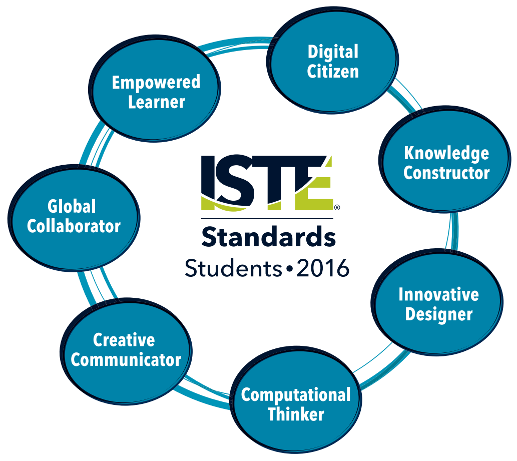
Students recognize the rights, responsibilities and opportunities of living, learning and working in an interconnected digital world, and they act and model in ways that are safe, legal and ethical. Students:

a. cultivate and manage their digital identity and reputation and are aware of the permanence of their actions in the digital world.
b. engage in positive, safe, legal and ethical behavior when using technology, including social interactions online or when using networked devices.
c. demonstrate an understanding of and respect for the rights and obligations of using and sharing intellectual property.
d. manage their personal data to maintain digital privacy and security and are aware of data-collection technology used to track their navigation online.

3. Knowledge Constructor

Students critically curate a variety of resources using digital tools to construct knowledge, produce creative artifacts and make meaningful learning experiences for themselves and others. Students:

a. plan and employ effective research strategies to locate information and other resources for their intellectual or creative pursuits.
b. evaluate the accuracy, perspective, credibility and relevance of information, media, data or other resources.
c. curate information from digital resources using a variety of tools and methods to create collections of artifacts that demonstrate meaningful connections or conclusions.
d. build knowledge by actively exploring real-world issues and problems, developing ideas and theories and pursuing answers and solutions.



4. Innovative Designer

Students use a variety of technologies within a design process to identify and solve problems by creating new, useful or imaginative solutions. Students:

a. know and use a deliberate design process for generating ideas, testing theories, creating innovative artifacts or solving authentic problems.
b. select and use digital tools to plan and manage a design process that considers design constraints and calculated risks.
c. develop, test and refine prototypes as part of a cyclical design process.
d. exhibit a tolerance for ambiguity, perseverance and the capacity to work with open-ended problems.

5. Computational Thinker

Students develop and employ strategies for understanding and solving problems in ways that leverage the power of technological methods to develop and test solutions. Students:

a. formulate problem definitions suited for technology-assisted methods such as data analysis, abstract models and algorithmic thinking in exploring and finding solutions.
b. collect data or identify relevant data sets, use digital tools to analyze them, and represent data in various ways to facilitate problem-solving and decision-making.
c. break problems into component parts, extract key information, and develop descriptive models to understand complex systems or facilitate problem-solving.
d. understand how automation works and use algorithmic thinking to develop a sequence of steps to create and test automated solutions.

6. Creative Communicator

Students communicate clearly and express themselves creatively for a variety of purposes using the platforms, tools, styles, formats and digital media appropriate to their goals. Students:

a. choose the appropriate platforms and tools for meeting the desired objectives of their creation or communication.
b. create original works or responsibly repurpose or remix digital resources into new creations.
c. communicate complex ideas clearly and effectively by creating or using a variety of digital objects such as visualizations, models or simulations.
d. publish or present content that customizes the message and medium for their intended audiences.

7. Global Collaborator

Students use digital tools to broaden their perspectives and enrich their learning by collaborating with others and working effectively in teams locally and globally. Students:

a. use digital tools to connect with learners from a variety of backgrounds and cultures, engaging with them in ways that broaden mutual understanding and learning.
b. use collaborative technologies to work with others, including peers, experts or community members, to examine issues and problems from multiple viewpoints.
c. contribute constructively to project teams, assuming various roles and responsibilities to work effectively toward a common goal.
d. explore local and global issues and use collaborative technologies to work with others to investigate solutions.