Computer Science Standards for California Public Schools

Kindergarten through Grade Twelve

Adopted by the California State Board of Education
August 2018

Computer Science Standards for California Public Schools

Kindergarten through Grade Twelve

Adopted by the California State Board of Education
August 2018

Publishing Information

The California Department of Education (CDE), Instructional Quality Commission (IQC), and State Board of Education (SBE) commenced the process for developing new California computer science content standards in September 2016. Per California *Education Code* Section 60605.4, "on or before July 31, 2019, the Instructional Quality Commission shall consider developing and recommending to the SBE computer science content standards for kindergarten and grades 1 to 12, inclusive, pursuant to recommendations developed by a group of computer science experts." On September 6, 2018, the SBE adopted the *California Computer Science Standards for Public Schools, Kindergarten Through Grade Twelve*.

Portions of this work are based on the *K–12 Computer Science Framework*. Licensed under Creative Commons (CC-BY-NC-SA-4.0).

When the SBE adopted the *Computer Science Standards*, the members of the IQC were Dean Reese, Chair; Soomin Chao, Vice Chair; Jocelyn Broemmelsiek; Christine Chapman; Lizette Diaz; Shay Fairchild; Jose Flores; Jose Iniguez; Risha Krishna; Jose Lara; Yolanda Muñoz; Melanie Murphy-Corwin; Nicole Naditz; Alma-Delia Renteria; Julie Tonkovich; Jennifer Woo; and Sharon Quirk-Silva, Assemblywoman. The members of the SBE were Michael W. Kirst, President; Ilene W. Straus, Vice President; Sue Burr; Bruce Holaday; Feliza I. Ortiz-Licon; Patricia A. Rucker; Niki Sandoval; Ting L. Sun; Karen Valdes; Trish Williams; and Gema Q. Cardenas, Student Member.

This publication was edited by Alex Calinsky, CDE Press, working in cooperation with Renée Ousley-Swank, Curriculum Frameworks and Instructional Resources Division. The document was prepared for publication by the staff of CDE Press; Aristotle Ramirez created the cover and interior design. It was published by the Department of Education, 1430 N Street, Sacramento, CA 95814, and was distributed under the provisions of the Library Distribution Act and *Government Code* Section 11096.

© 2021 by the California Department of Education
All rights reserved

ISBN 978-0-8011-1809-8

Additional Publications and Educational Resources

For information about publications and educational resources available from the California Department of Education, please visit the CDE Press Educational Resources web page at https://cdep.klas.com/ or call the CDE Press sales office at 1-800-995-4099. Reproduction of this document for resale, in whole or in part, is not authorized.

Contents

A Message from the State Superintendent of Public Instruction and the State Board of Education **1**
Special Acknowledgements **3**
Vision **5**
Why Computer Science? **8**
Equity Issues **10**
Problem Solving and the Four Cs **12**
What is Computer Science? **15**
Computer Science Core Concepts 17
Computer Science Core Practices 20
California K–12 Computer Science Standards **30**
K–2 30
3–5 50
6–8 73
9–12 97
9–12 Specialty 127
References and Attributions **157**
California Computer Science Standards: Appendix **159**

> Computer science is no more about computers than astronomy is about telescopes.
>
> **—Anonymous, at times attributed to Edsger Dijkstra**

Page iv intentionally blank.

A Message from the State Superintendent of Public Instruction and the State Board of Education

The *Computer Science Standards for California Public Schools, Kindergarten Through Grade Twelve* is now available in its final format. The standards may be found on the CDE website at https://www.cde.ca.gov/be/st/ss/computerscicontentstds.asp. These standards, adopted by the State Board of Education in September 2018, set a groundbreaking vision for learning expectations that will help each student reach their creative potential in our digitally connected world. Computer science education not only enables students to understand how their digital world works, but it also encourages critical thinking and discussion around the broader ethical and social implications, including questions related to the growing capabilities of technology.

The California Computer Science Standards are based on the computer science core concepts and core practices, aligned to the K–12 Computer Science Framework. The standards were developed utilizing previous work by the Computer Science Teachers Association. The standards are designed to be accessible to each and every student in California.

Additionally, the standards place a strong emphasis on equity by providing educators with examples of ways they can broaden participation in computer science to include students who have not historically been provided the opportunity. While 60 percent of California's student population is Latinx or African American, only about 25 percent of students who take high school computer science courses are from these demographic groups. In the technology workforce, Latinx and African Americans represent about 15 percent of employees. The State Superintendent believes public education can play a critical role in dismantling systemic racism and historic barriers to opportunity in ways that help schools achieve educational equity.

Designed to help students move from being passive users of technology to becoming creators and innovators, the standards go beyond the goal of students learning to code and motivate students to communicate as scientists and find creative solutions to difficult problems. These standards provide our students with a deeper understanding of computer science that prepares them for careers and college and helps them succeed in a fiercely competitive global economy.

Computer science as a foundational discipline that should be accessible to all students is a relatively new concept in K–12 education. Computer science is the study of how technology and computing systems are created and their impact on society. The standards cover six core computer science concepts (such as algorithms and programming) and seven core practices (such as creating computational artifacts and recognizing computational problems). By contrast, learning to type, word processing, and computer repair are not within the scope of computer science as defined by the standards.

We encourage educators to utilize the computer science standards to develop curriculum, instruction, and assessment, which are the foundation of quality computer science learning in our schools, and we urge local educational agencies to ensure access to computer science learning for each student in California.

Special Acknowledgements

The State Board of Education extends its appreciation to the standards writing team: Katherine Goyette, Tulare County Office of Education; Aleata Hubbard, WestEd and Yvonne Kao, WestEd.

The State Board of Education extends its appreciation to the following members of the Computer Science Standards Advisory Committee contributed to the development of the standards document: Beth Simon Co-Chair; Bryan Twarek Co-Chair; Casey Agena; Jared Amalong; Stephen Callahan; Joseph Chipps; Christina Cowan-Hastle; Myra Deister; Nathan Drown; William Epps; Veronica Godinez; Ann Greyson; Richard Kick; Smita Kolhatkar; Steve Kong; Arthur Lopez; Elyse Sharp; Kevin Tambara; Gina Thackrey; and Andrew Williams.

Special appreciation is extended to Dr. Stephanie Gregson, Director, Curriculum Frameworks and Instructional Resources Division; Kristen Cruz-Allen, Administrator, Curriculum Frameworks and Instructional Resources Division; Cliff Rudnick, Administrator, Curriculum Frameworks and Instructional Resources Division; Paula Evans, Education Programs Consultant/Co-Lead for the development of the California Computer Science Standards, Professional Learning Support Division; and Renée Ousley-Swank Education Programs Consultant/Co-Lead for the development of the California Computer Science Standards, Curriculum Frameworks and Instructional Resources Division.

Special recognition is awarded to the following additional CDE staff who contributed to this document: Lisa Grant, Education Programs Consultant, Curriculum Frameworks and Instructional Resources Division; Erle Hall, Education Programs Consultant, Career Technical Education Leadership and Instructional Support Office; Letty Kraus, Education Programs Consultant, Curriculum Frameworks and Instructional Resources Division; Emily Oliva, Education Programs Consultant, Educator Excellence and Equity Division; and Tracie Yee, Associate Governmental Program Analyst, Curriculum Frameworks and Instructional Resources Division.

Page 4 intentionally blank.

Vision

The California Computer Science Standards (hereafter referred to as "the standards") are based on computer science core concepts and core practices, aligned to the K–12 Computer Science Framework. The standards were developed by educators (members of the State Board of Education-appointed Computer Science Standards Advisory Committee), utilizing work done by the Computer Science Teachers Association. The standards are designed to be accessible for each and every student in California. The standards inform teachers, curriculum developers, and educational leaders to ensure all students receive quality computer science instruction. Each standard includes a descriptive statement and examples for classroom application. Examples are not meant to be prescriptive or compulsory. Rather, they are designed as general suggestions.

Educators are encouraged to design computer science learning experiences, according to their local capacity and context, to meet the needs of their students. Computer science core concepts and practices in the standards are vertically aligned, coherent across grades, and designed in developmentally appropriate grade spans: K–2, 3–5, 6–8, and 9–12. The K–12 standards are referred to as "core." The ninth grade-to-twelfth grade span also includes an additional set of standards, referred to as "9–12 Specialty," which provides options for extending a pathway in computer science with content containing increased complexity and depth.

The 9–12 Specialty standards may be used to create electives that are outside an introductory course. As students progress through the standards from kindergarten to twelfth grade, they build conceptual knowledge through active engagement in creative problem-solving activities with an awareness of cultural and societal contexts.

Computers have been used in classrooms across the state for many years. However, students often take a passive role as mere users of these devices. The standards empower students to deepen their understanding of computer science as they explore core concepts, such as: the composition of computing systems (CS), the connective power of networks and information systems (NI), the informational potential of data and analysis processing (DA), the development of algorithms and programming (AP), and the impacts of computing on culture and society (IC). These core concepts provide foundational knowledge on key ideas, which build upon each other as students progress through grade spans. The computer science core concepts are covered in greater detail in the "What is Computer Science?" section.

The computer science core practices infuse computer science core concepts with purpose and relevance. The core practices focus on how students interact with computer science—as in, the ways they apply conceptual knowledge. These core practices enable

students to experience computer science as a creative process. Instead of merely using computing technology, students are actively creating and innovating, engaging with computer science as an artistic and collaborative endeavor. As students engage in the computer science core practices, they learn to persevere in solving authentic, community-based problems grounded in computer science core concepts. The computer science core practices, covered in greater detail in the "What is Computer Science?" section, include:

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

The standards integrate computer science practices with concept statements.

Practice	+ Concept	= Standard
Creating Computational Artifacts The process of developing computational artifacts embraces both creative expression and the exploration of ideas to create prototypes and solve computational problems. Students create artifacts that are personally relevant or beneficial to their community and beyond. Computational artifacts can be created by combining and modifying existing artifacts or by developing new artifacts.	**Computing Systems** Hardware and software determine a computing system's capability to store and process information. The design or selection of a computing system involves multiple considerations and potential tradeoffs, such as functionality, cost, size, speed, accessibility, and aesthetics.	**6-8.CS.2** Design a project that combines hardware and software components to collect and exchange data.

The standards contain significant themes, as referenced in the K–12 Computer Science Framework, include:

- **Equity.** Issues of equity, inclusion, and diversity are addressed in concepts and practices, the standards, and in examples of ways to broaden participation in computer science education listed in the appendix.
- **Powerful ideas.** The concepts and practices evoke authentic, powerful ideas that can be used to solve real-world problems and connect understanding across multiple disciplines (Papert 2000).
- **Computational thinking.** Computational thinking is the human ability to formulate problems so that their solutions can be represented as computational steps or algorithms to be executed by a computer. Computational thinking practices such as abstraction, modeling, and decomposition intersect with computer science concepts such as algorithms, automation, and data visualization.
- **Breadth of application.** Computer science is more than coding. It involves physical systems and networks; the collection, storage, and analysis of data; and the impact of computing on society. This broad view of computer science emphasizes the range of applications that computer science has in other fields.

As a field, computer science crosses multiple disciplines. To accurately reflect the field, the standards are interdisciplinary in nature to ensure that every student learns computer science core concepts in relevant contexts. The standards are consistent with State Board of Education-adopted curriculum standards in their emphasis on problem solving, communication, critical thinking, creativity, and collaboration. Many standards include interdisciplinary examples and the appendix provides additional cross-referenced alignment charts by grade level. The standards also complement the Career Technical Education (CTE) Standards for California Public Schools.

Why Computer Science?

Computer science is an essential component of a broad and comprehensive education, containing necessary foundational concepts and corresponding practices that generate opportunities for success in our increasingly competitive, globally connected economy. Computing systems are more than tools, as they have the power to facilitate personal and creative expression. As illustrated in the K–12 Computer Science Framework, computers are both the paint and paintbrush. Computer science education creates the artists.

Digital technologies are largely responsible for the global connectivity of the economy, and the impact of computer science on multiple areas of the human endeavor continues to increase rapidly. However, computer science education has not kept pace with this increased influence on society. The standards are an integral part of preparing students for their college and career. Student interest in computer science at the postsecondary level is increasing, and job opportunities in STEM fields are increasing. Since 2010, computer science ranks as one of the fastest growing undergraduate majors of all STEM fields (Fisher 2015), and Advanced Placement (AP) Computer Science A is the fastest growing AP course, despite being offered in only 5 percent of schools (Code.org 2015). The launch of AP Computer Science Principles was the largest course launch in history (College Board 2018). Despite the growth of AP Computer Science Principles, a mere 0.5 percent of high school students in California took the AP Computer Science A exam in 2016 (College Board 2016). Jobs that use computer science are some of the highest paying, highest growth (US Department of Labor, Bureau of Labor Statistics 2015), and most in-demand jobs that underpin the economy (The Conference Board 2016). The computer science field has a shortage of engineers and programmers and an increase in computer science education is vital to fill this need (DeNisco Rayome 2017).

While the aforementioned statistics highlight the growing need for computer science specialty courses, it is just as necessary that computer science be included as a core subject for every student in grade levels K–12. Computer science core concepts and practices can prepare all students for their college and career, even if they do not pursue a computer science degree or occupation. The computer science core practices help develop lifelong learners that persevere in the processes of creative problem solving in a way that promotes inclusion and celebrates diversity. Computer science core concepts provide foundational knowledge of computing principles, giving students opportunities to develop as computational, logical thinkers who carefully weigh the societal and cultural impacts of computing. The standards foster responsible citizenship, as they include themes of equity, engage students in practices that promote an inclusive computing

culture, and guide students toward responsible protection and use of information in networks and the internet. Students who study computer science become informed citizens who develop conceptual knowledge of how computing technology works and also contribute productively to society as a whole.

Computer science prepares students for the future and also fosters skills that support their education in the present, furthering their development and motivation as learners. The importance of computer science is recognized at a national level, with the Every Student Succeeds Act defining computer science as part of a "well-rounded education" (2015). Computer science education cultivates personal fulfillment by motivating students to become innovators. Students can build confidence in solving complex, open-ended problems by designing, creating, and developing computational artifacts. Computer science education is often implemented using a project-based approach, encouraging educators to actively engage students via solid pedagogical practices that empower learners to construct knowledge in a student-led environment.

The general public recognizes the need for computer science inclusion as a core subject for all students. Americans believe computer science is as important to learn as reading, writing, and math (Horizon Media 2015). While 90 percent of U.S. parents want their child to learn computer science, there is a disconnect from school administrators—only 7 percent of principals say there is a high demand for computer science among parents (Google and Gallup 2015, 12). It is vital that school administrators address this disconnect and align their vision to the communities they serve.

The standards are designed to increase access of computer science instruction for all students, as a core subject in addition to specialty courses. Computer science instruction empowers students, giving them confidence to use computers and computing tools to solve problems. As students learn computer science, they build an understanding of the importance of computing and computing tools. The standards prepare all students to enter their college and career as critical consumers and thoughtful computing technology creators and innovators.

Equity Issues

California schools house the largest, most diverse population of students in the United States (California Department of Education 2016). As such, it is imperative that all core subjects, including computer science, are not merely inclusionary, but that instruction uses practices that actively engage students and increase access for underserved populations. Equity in computer science education does not equate to preparing all students to major in computer science at the post-secondary level, or to pursue careers in software engineering or other areas of computing technologies. Rather, computer science education for all ensures that every student develops a foundation of conceptual knowledge and proficiency in computer science practices, which provides the skills to responsibly and productively participate in a world with broadly integrated digital technologies.

Equity is more than an availability of computer science classes—it requires leaders and educators to carefully consider the following: inclusive practices regarding how classes are taught, student recruitment and retention, instructional practices that guarantee universal access, and high expectations for all students. Computer science is not designed to be offered only to a select few, or as an elective for interested students. Equity in computer science calls for leaders and educators to guarantee computer science instruction for all students, as an essential core subject that is a necessary and valuable component of a comprehensive education.

Historically, computer science has been inaccessible to the majority of K–12 students. Approximately 65 percent of high schools in California offer no computing classes (Level Playing Field Institute 2016, 7). Computer science education rates at the K–8 level are even more dismal. While 59 percent of California's student population is Latinx or African American, these students comprise only 11 percent of students taking AP Computer Science A and 9 percent of the computing workforce nationwide (Level Playing Field Institute 2016, 8). While female students are 49 percent of the population, they comprised only 24 percent of AP Computer Science A test takers in California (Level Playing Field Institute 2016, 7). Students in small town or rural school districts face a digital divide despite their need for computer science education. While 86 percent of students in small town or rural school districts are somewhat or very likely to indicate they will have a job in the future that requires knowledge of computer science, and 92 percent express interest in learning computer science, principals from these schools are 7 percent less likely to indicate that computer science is a priority, when compared to principals from suburban and large city school districts (Google and Gallup 2017).

The standards are designed for all students, including underserved populations: girls, low-income students, homeless students, rural students, African-American and Latinx students, students

who are English learners, students with disabilities, and foster youth. Students' access to and achievement in computer science must not be predictable on the basis of race, ethnicity, gender, socioeconomic status, language, religion, sexual orientation, cultural affiliation, or special needs. All students are to be given access to computer science instruction as a core subject. To guarantee equitable access to computer science education regardless of socioeconomic status, many computer science concepts and practices can be learned with or without a computer or other digital device. Guidance for ensuring universal access to the standards through flexible implementation options is available in the standards appendix.

Computer science instruction has the potential to promote and foster inclusion, diversity, and equity. At its best, computer science focuses on user needs and continually increases accessibility through iterative development processes. This inclusive, user-centric mindset within computer science led to innovations in computing technology, such as wearable hearing aid devices, real-time translation services, and accessibility options, within computing hardware and software for individuals with disabilities, among others. The standards encourage students to study computer science core concepts within a context of its potential impacts on both local and global communities. These core concepts are coupled with core computer science practices that expressly require students to foster an inclusive computing culture addressing diverse needs and unique perspectives. As such, the study of computer science is a key factor in developing student empathy and celebrating diversity.

Problem Solving and the Four Cs

Colleges and careers of the future will require students to problem solve and demonstrate the Four Cs: collaboration, critical thinking, creativity, and communication. These skills are echoed throughout the California Common Core State Standards for many subjects. The California computer science standards similarly emphasize these skills.

As a field, computer science itself incorporates problem solving, communication, critical thinking, creativity, and collaboration into its work. The following is a representation of the California computer science core practices and their alignment to equity, problem solving, and the Four Cs.

Core Practice 1: Equity
Fostering an Inclusive Computing Culture

Core Practice 2: Collaboration
Collaborating Around Computing

Core Practice 3: Problem Solving
Recognizing and Defining Computational Problems

Core Practice 4: Critical Thinking
Developing and Using Abstractions

Core Practice 5: Creativity
Creating Computational Artifacts

Core Practice 6: Creativity
Testing and Refining Computational Artifacts

Core Practice 7: Communication
Communicating About Computing

Collaboration in computer science fosters contributions and feedback from others, which may result in improved outcomes as opposed to working independently. Core Practice 2, *Collaborating Around Computing*, encourages students to work in collaborative teams. A particular emphasis is placed on effective collaboration techniques, including shared norms and expectations, as well as using technological tools to support collaborative work. In the computer science industry, development teams collaborate together, soliciting feedback and providing feedback to others, to meet common goals.

Problem solving is the foundation of computer science. By nature, computer science exists to solve problems for people and the world. Computer scientists create algorithms, which are step-by-step instructions for a program, to design potential solutions for end users. The study of computer science core concepts promotes empathy—it urges students to identify authentic problems and create solutions to increase accessibility or functionality based on individual users' needs. The standards emphasize problem solving as a necessary practice in the study of computer science. Core Practice 3, *Recognizing and Defining Computational Problems*, requires students to not only identify real-world, interdisciplinary problems that can be solved computationally, but also break down these problems and develop potential solutions for the betterment of society.

Critical thinking is a key component of computer science, as exhibited in Core Practice 4, *Developing and Using Abstractions*. As students engage in Core Practice 4, they build knowledge of computer science core concepts through the cognitive work of identifying patterns, creating generalizations, evaluating existing functionalities, applying learning to new designs, and managing complexity. These tasks require more than rote memorization. Rather, they call upon students to analyze, evaluate, and problem solve.

Creativity and innovation drive the computer science field. Computer scientists explore programs and identify problems they work to solve through the creation of new computational artifacts. As students study computer science, they develop new ideas, create prototypes based on their ideas, share their ideas and prototypes with others, test their prototypes, reflect on the experience, generate new ideas for altering their prototypes, test their prototypes again, and continue in this iterative creative process. Core Practice 5, *Creating Computational Artifacts*, emphasizes the need for students to iteratively create when engaging in computer science learning. Core Practice 6, *Testing and Refining Computational Artifacts*, requires students to focus on creation as a process of continual refinement based on user needs.

Communication is a necessary skill for computer scientists. Core Practice 7, *Communicating About Computing*, emphasizes clear communication with precise language, incorporating consideration for audience needs. Computer scientists communicate with clients, team members, and end users. The study of computer science teaches students to consider their audience when communicating. A computer scientist's various stakeholder groups may have differing vernaculars, requiring them to learn how to communicate in varying language registers. According to the California English Language Arts/English Language Development (ELA/ELD) Framework, "Register refers to the ways in which

grammatical and lexical resources are combined to meet the expectations of the context (i.e., the content area, topic, audience, and mode in which the message is conveyed)" (California Department of Education 2015, Figure 2.14). When developing a program for end users, communication within an app must be user-friendly, and developers must learn the most effective ways of creating an interactive experience in which users provide feedback to continually improve functionality and accessibility. As such, in studying computer science concepts, students learn the importance of listening and responding to user needs. The iterative process inherent in product development provides students with a real-world example of the need for formative feedback and a focus on continual improvement.

What is Computer Science?

Computer science involves much more than use of computing systems. To provide universal access to computer science instruction for all students, we must thoroughly define computer science. Computer science is "the study of computers and algorithmic processes, including their principles, their hardware and software designs, their applications, and their impact on society" (Tucker et al. 2006, 2). Computer science is the science of computing. The term 'computer science' is often misconstrued with other technological and digital terminology.

Computer science **is**:

- a theory and practice that allows you to program a computer to do what you want it to do;
- a tool that helps you tell a story or make something happen with technology;
- a discipline that emphasizes persistence in problem solving—a skill that is applicable across disciplines, driving job growth and innovation across all sectors of the workforce; and
- a skill that teaches students how to use computers to create, not just consume.

Computer science **is not**:

- learning how to type or use a mouse;
- learning to use word processing, spreadsheet, or presentation software;
- learning how to build or repair computers;
- playing video games; and
- learning Smarter Balanced skills.

Adapted from the CS First curriculum (Google n.d.).

Computer science is not limited to computer literacy, educational technology, digital citizenship, and information technology. It builds on these topics and goes further in complexity and depth in the following ways:

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

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

Computer literacy, educational technology, digital citizenship, and information technology focus on use. Computer science requires students to not merely use technology as passive consumers. Computer science also calls for students to understand why and how computing technologies work, and then build upon that conceptual knowledge by creating computational artifacts.

The standards include five core concept areas, coupled with seven core practices that demonstrate ways in which students actively engage in computer science learning experiences that build conceptual knowledge.

The computer science **core concepts** include:

- Computing Systems (CS)
- Networks and the Internet (NI)
- Data and Analysis (DA)
- Algorithms and Programming (AP)
- Impacts of Computing (IC)

The computer science **core practices** include:

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

The five core computer science concepts and seven practices do not constitute a scope and sequence of appropriate pacing for a course, nor are they designed to be introduced to students as independent courses. Guidance regarding specific interdisciplinary connections are provided in the appendix. The examples that accompany the standards are not meant to prescribe requirements as to implementation and assessment but to illustrate potential models for standards implementation. These examples and interdisciplinary connections are designed to provide substantive guidance while also allowing for flexibility and innovation across local education agencies. The computer science core concepts and core practices are coherent across grades K–12. The standards are vertically aligned as a core subject area, according to the following grade spans: K–2, 3–5, 6–8, and 9–12. A more detailed look at the computer science core concepts and core practices follows.

Computer Science Core Concepts

The core concepts in the standards represent key content areas. The core concepts describe the content knowledge that students should understand regarding computer science. Each core concept contains subconcepts, which increase in complexity according to grade span, as seen in the standards. Core concept definitions and subconcept descriptions are taken from the K–12 Computer Science Framework.

Core Concept—Computing Systems (CS)

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

Subconcept: 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 the interaction between humans and devices influences design decisions.

Subconcept: 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.

Subconcept: 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.

Core Concept—Networks and the Internet (NI)

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

Subconcept: 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.

Subconcept: 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.

Core Concept—Data and Analysis (DA)

Subconcept: Storage

Core functions of computers are storing, representing, and retrieving data. In early grades, students learn how data is stored on computers. As they progress, students learn how to evaluate different storage methods, including the tradeoffs associated with those methods.

Subconcept: Collection, Visualization, and Transformation

Data is collected with both computational and noncomputational tools and processes. In early grades, students learn how data about themselves and their world is collected and used. As they progress, students learn the effects of collecting data with computational and automated tools. Data is transformed throughout the process of collection, digital representation, and analysis. In early grades, students learn how transformations can be used to simplify data. As they progress, students learn about more complex operations to discover patterns and trends and communicate them to others.

Subconcept: 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 the data collected from users or simulations. In early grades, students learn about the use of data to make simple predictions. As they progress, students learn how models and simulations can be used to examine theories and understand systems and how predictions and inferences are affected by more complex and larger data sets.

Core Concept—Algorithms and Programming (AP)

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

Subconcept: Algorithms

Algorithms are designed to be carried out by both humans and computers. In early grades, students learn about age-appropriate algorithms from the real world. As they progress, students learn about the development, combination, and decomposition of algorithms, as well as the evaluation of competing algorithms.

Subconcept: Variables

Computer programs store and manipulate data using variables. In early grades, students learn that different types of data, such as words, numbers, or pictures, can be used in different ways. As they progress, students learn about variables and ways to organize large collections of data into data structures of increasing complexity.

Subconcept: Control

Control structures specify the order in which instructions are executed within an algorithm or program. In early grades, students learn about sequential execution and simple control structures. As they progress, students expand their understanding to combinations of structures that support complex execution.

Subconcept: Modularity

Modularity involves breaking down tasks into simpler tasks and combining simple tasks to create something more complex. In early grades, students learn that algorithms and programs can be designed by breaking tasks into smaller parts and recombining existing solutions. As they progress, students learn about recognizing patterns to make use of general, reusable solutions for commonly occurring scenarios and clearly describing tasks in ways that are widely usable.

Subconcept: Program Development

Programs are developed through a design process that is often repeated until the programmer is satisfied with the solution. In early grades, students learn how and why people develop programs. As they progress, students learn about the tradeoffs in program design associated with complex decisions involving user constraints, efficiency, ethics, and testing.

Core Concept—Impacts of Computing (IC)

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

Subconcept: Culture

Computing influences culture—including belief systems, language, relationships, technology, and institutions—and culture shapes how people engage with and access computing. In early grades, students learn how computing can be helpful and harmful. As they progress, students learn about tradeoffs associated with computing and potential future impacts of computing on global societies.

Subconcept: Social Interactions

Computing can support new ways of connecting people, communicating information, and expressing ideas. In early grades, students learn that computing can connect people and support interpersonal communication. As they progress, students learn how the social nature of computing affects institutions and careers in various sectors.

Subconcept: Safety, Law, and Ethics

Legal and ethical considerations of using computing devices influence behaviors that can affect the safety and security of individuals. In early grades, students learn the fundamentals of digital citizenship and appropriate use of digital media. As they progress, students learn about the legal and ethical issues that shape computing practices.

Computer Science Core Practices

The computer science core practices in the standards represent how students do computer science while building conceptual knowledge of the key content areas. According to the K–12 Computer Science Framework, the seven core practices of computer science describe the behaviors and ways of thinking that computationally literate students use to fully engage in today's data-rich and interconnected world. Each core practice contains practice statements. Core practice definitions and practice statements are taken from the K–12 Computer Science Framework. Practice statements describe what students should be able to do by the end of grade level twelve.

CORE PRACTICE 1
Fostering an Inclusive Computing Culture

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

By the end of grade twelve, students should be able to:

1. Include the unique perspectives of others and reflect on one's own perspectives when designing and developing computational products.
    - At all grade levels, students should recognize that the choices people make when they create artifacts are based on personal interests, experiences, and needs. Young learners should begin to differentiate their technology preferences from the technology preferences of others. Initially, students should be presented with perspectives from people with different backgrounds, ability levels, and points of view. As students progress, they should independently seek diverse perspectives throughout the design process for the purpose of improving their computational artifacts. Students who are well-versed in fostering an inclusive computing culture should be able to differentiate backgrounds and skill sets and know when to call upon others, such as to seek out knowledge about potential end users or intentionally seek input from people with diverse backgrounds.
2. Address the needs of diverse end users during the design process to produce artifacts with broad accessibility and usability.
    - At any level, students should recognize that users of technology have different needs and preferences and that not everyone chooses to use, or is able to use, the same technology products. For example, young learners, with

teacher guidance, might compare a touchpad and a mouse to examine differences in usability. As students progress, they should consider the preferences of people who might use their products. Students should be able to evaluate the accessibility of a product to a broad group of end users, such as people with various disabilities. For example, they may notice that allowing an end user to change font sizes and colors will make an interface usable for people with low vision. At the higher grades, students should become aware of professionally accepted accessibility standards and should be able to evaluate computational artifacts for accessibility. Students should also begin to identify potential bias during the design process to maximize accessibility in product design. For example, they can test an app and recommend to its designers that it respond to verbal commands to accommodate users who are blind or have physical disabilities.

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

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

CORE PRACTICE 2
Collaborating Around Computing

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

By the end of grade twelve, students should be able to:

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

    - At any grade level, students should work collaboratively with others. Early on, they should learn strategies for working with team members who possess varying individual strengths. For example, with teacher support, students should begin to give each team member opportunities to contribute and to work with each other as co-learners. Eventually, students should become more sophisticated at applying strategies for mutual encouragement and support. They should express their ideas with logical reasoning and find ways to reconcile differences cooperatively. For example, when they disagree, they should ask others to explain their reasoning and work together to make respectful, mutual decisions. As they progress, students should

use methods for giving all group members a chance to participate. Older students should strive to improve team efficiency and effectiveness by regularly evaluating group dynamics. They should use multiple strategies to make team dynamics more productive. For example, they can ask for the opinions of quieter team members, minimize interruptions by more talkative members, and give individuals credit for their ideas and their work.

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

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

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

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

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

    - At any level, students should be able to use tools and methods for collaboration on a project. For example, in the early grades, students could collaboratively brainstorm by writing on a whiteboard. As students progress, they should use technological collaboration tools to manage teamwork, such as knowledge-sharing

tools and online project spaces. They should also begin to make decisions about which tools would be best to use and when to use them. Eventually, students should use different collaborative tools and methods to solicit input from not only team members and classmates but also others, such as participants in online forums or local communities.

CORE PRACTICE 3
Recognizing and Defining Computational Problems

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

By the end of grade twelve, students should be able to:

1. Identify complex, interdisciplinary, real-world problems that can be solved computationally.
    - At any level, students should be able to identify problems that have been solved computationally. For example, young students can discuss a technology that has changed the world, such as email or mobile phones. As they progress, they should ask clarifying questions to understand whether a problem or part of a problem can be solved using a computational approach. For example, before attempting to write an algorithm to sort a large list of names, students may ask questions about how the names are entered and what type of sorting is desired. Older students should identify more complex problems that involve multiple criteria and constraints. Eventually, students should be able to identify real-world problems that span multiple disciplines, such as increasing bike safety with new helmet technology, and can be solved computationally.
2. Decompose complex real-world problems into manageable subproblems that could integrate existing solutions or procedures.
    - At any grade level, students should be able to break problems down into their component parts. In the early grade levels, students should focus on breaking down simple problems. For example, in a visual programming environment, students could break down (or decompose) the steps needed to draw a shape. As students progress, they should decompose larger problems into manageable smaller problems. For example, young students may think of an animation as multiple scenes and thus create each scene independently. Students can also break down a program into subgoals: getting input from the user, processing the data, and displaying the result to the user. Eventually, as students encounter complex real-world problems that span multiple disciplines or social systems, they should decompose complex problems into manageable subproblems that could potentially be solved with programs or procedures that already exist. For example, students could create an app to solve a

community problem that connects to an online database through an application programming interface (API).

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

CORE PRACTICE 4
Developing and Using Abstractions

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

By the end of grade twelve, students should be able to:

1. Extract common features from a set of interrelated processes or complex phenomena.
    - Students at all grade levels should be able to recognize patterns. Young learners should be able to identify and describe repeated sequences in data or code through analogy to visual patterns or physical sequences of objects. As they progress, students should identify patterns as opportunities for abstraction, such as recognizing repeated patterns of code that could be more efficiently implemented as a loop. Eventually, students should extract common features from more complex phenomena or processes. For example, students should be able to identify common features in multiple segments of code and substitute a single segment that uses variables to account for the differences. In a procedure, the variables would take the form of parameters. When working with data, students should be able to identify important aspects and find patterns in related data sets such as crop output, fertilization methods, and climate conditions.
2. Evaluate existing technological functionalities and incorporate them into new designs.
    - At all levels, students should be able to use well-defined abstractions that hide complexity. Just as a car hides operating details, such as the mechanics of the engine, a computer program's "move" command relies on hidden details that cause an object to change location on the

screen. As they progress, students should incorporate predefined functions into their designs, understanding that they do not need to know the underlying implementation details of the abstractions that they use. Eventually, students should understand the advantages of, and be comfortable using, existing functionalities (abstractions) including technological resources created by other people, such as libraries and application programming interfaces (APIs). Students should be able to evaluate existing abstractions to determine which should be incorporated into designs and how they should be incorporated. For example, students could build powerful apps by incorporating existing services, such as online databases that return geolocation coordinates of street names or food nutrition information.

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

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

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

    - Students at all grade levels should be able to represent patterns, processes, or phenomena. With guidance, young students can draw pictures to describe a simple pattern, such as sunrise and sunset, or show the stages in a process, such as brushing your teeth. They can also create an animation to model a phenomenon, such as evaporation. As they progress, students should understand that computers can model real-world phenomena, and they should use existing computer simulations to learn about real-world systems. For example, they may use a preprogrammed model to explore how parameters affect a system, such as how rapidly a disease spreads. Older students should model phenomena as systems, with rules governing the interactions within the system. Students should analyze and evaluate these models against real-

world observations. For example, students might create a simple producer–consumer ecosystem model using a programming tool. Eventually, they could progress to creating more complex and realistic interactions between species, such as predation, competition, or symbiosis, and evaluate the model based on data gathered from nature.

CORE PRACTICE 5
Creating Computational Artifacts

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

By the end of grade twelve, students should be able to:

1. Plan the development of a computational artifact using an iterative process that includes reflection on and modification of the plan, taking into account key features, time and resource constraints, and user expectations.
    - At any grade level, students should participate in project planning and the creation of brainstorming documents. The youngest students may do so with the help of teachers. With scaffolding, students should gain greater independence and sophistication in the planning, design, and evaluation of artifacts. As learning progresses, students should systematically plan the development of a program or artifact and intentionally apply computational techniques, such as decomposition and abstraction, along with knowledge about existing approaches to artifact design. Students should be capable of reflecting on and, if necessary, modifying the plan to accommodate end goals.
2. Create a computational artifact for practical intent, personal expression, or to address a societal issue.
    - Students at all grade levels should develop artifacts in response to a task or a computational problem. At the earliest grade levels, students should be able to choose from a set of given commands to create simple animated stories or solve pre-existing problems. Younger students should focus on artifacts of personal importance. As they progress, student expressions should become more complex and of increasingly broader significance. Eventually, students should engage in independent, systematic use of design processes to create artifacts that solve problems with social significance by seeking input from broad audiences.
3. Modify an existing artifact to improve or customize it.
    - At all grade levels, students should be able to examine existing artifacts to understand what they do. As they progress, students should attempt to use existing solutions to accomplish a desired goal. For example,

students could attach a programmable light sensor to a physical artifact they have created to make it respond to light. Later on, they should modify or remix parts of existing programs to develop something new or to add more advanced features and complexity. For example, students could modify prewritten code from a single-player game to create a two-player game with slightly different rules.

CORE PRACTICE 6
Testing and Refining Computational Artifacts

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

By the end of grade twelve, students should be able to:

1. Systematically test computational artifacts by considering all scenarios and using test cases.
    - At any grade level, students should be able to compare results to intended outcomes. Young students should verify whether given criteria and constraints have been met. As students progress, they should test computational artifacts by considering potential errors, such as what will happen if a user enters invalid input. Eventually, testing should become a deliberate process that is more iterative, systematic, and proactive. Older students should be able to anticipate errors and use that knowledge to drive development. For example, students can test their program with inputs associated with all potential scenarios.
2. Identify and fix errors using a systematic process.
    - At any grade level, students should be able to identify and fix errors in programs (debugging) and use strategies to solve problems with computing systems (troubleshooting). Young students could use trial and error to fix simple errors. For example, a student may try reordering the sequence of commands in a program. In a hardware context, students could try to fix a device by resetting it or checking whether it is connected to a network. As students progress, they should become more adept at debugging programs and begin to consider logic errors: cases in which a program works, but not as desired. In this way, students will examine and correct their own thinking. For example, they might step through their program, line by line, to identify a loop that does not terminate as expected. Eventually, older students should progress to using more complex strategies for identifying and fixing errors, such as printing the value of a counter variable while a loop is running to determine how many times the loop runs.
3. Evaluate and refine a computational artifact multiple times to enhance its performance, reliability, usability, and accessibility.

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

CORE PRACTICE 7
Communicating About Computing

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

By the end of grade twelve, students should be able to:

1. Select, organize, and interpret large data sets from multiple sources to support a claim.
   - At any grade level, students should be able to refer to data when communicating an idea. Early on, students should, with guidance, present basic data through the use of visual representations, such as storyboards, flowcharts, and graphs. As students progress, they should work with larger data sets and organize the data in those larger sets to make interpreting and communicating it to others easier, such as through the creation of basic data representations. Eventually, students should be able to select relevant data from large or complex data sets in support of a claim or to communicate the information in a more sophisticated manner.
2. Describe, justify, and document computational processes and solutions using appropriate terminology consistent with the intended audience and purpose.
   - At any grade level, students should be able to talk about choices they make while designing a computational artifact. Early on, they should use language that articulates what they are doing and identifies devices and concepts they are using with correct terminology (e.g., program, input, and debug). Younger students should identify the goals and expected outcomes of their solutions. Along the way, students should provide documentation for end users that explains their artifacts and how they function, and they should both give and receive feedback. For example, students could provide a project overview and ask for input from users. As students progress, they should incorporate clear

comments in their product and document their process using text, graphics, presentations, and demonstrations.

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

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

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.

K–2

K-2.CS.1

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
K–2	K-2.CS.1	Select and operate computing devices that perform a variety of tasks accurately and quickly based on user needs and preferences.	Computing Systems	Devices	Inclusion	1.1

Descriptive Statement

People use computing devices to perform a variety of tasks accurately and quickly. Computing devices interpret and follow the given instructions literally. Students select and operate an appropriate computing device and corresponding program or app for a given task.

For example, students could use computing devices to describe what plants and animals (including humans) need to survive. In this case, students could choose to use a keyboard to type explanatory sentences onto graphics. They could also choose to use a touchscreen device with a stylus to annotate an image for a slideshow, or choose to use a camera enabled device to make a video. Student choices may reflect their own needs or the needs of others (CA NGSS: K-LS1-1; 2-LS4-1).

Alternatively, students could choose to use a computing device with audio recording capabilities to recount stories or poems. Students could clarify thoughts, ideas, or feelings via their preference of either using a device with digital drawing tools, or by creating paper and pencil drawing based on their needs and preferences (CA CCSS for ELA/Literacy SL.K.5, SL.1.5, SL.2.5).

K-2.CS.2

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
K–2	K-2.CS.2	Explain the functions of common hardware and software components of computing systems.	Computing Systems	Hardware & Software	Communicating	7.2

Descriptive Statement

A computing system is composed of hardware and software. Hardware includes the physical components of a computer system. Software provides instructions for the system. These instructions are represented in a form that a computer can understand and are designed for specific purposes. Students identify and describe the function of hardware, such as desktop computers, laptop computers, tablet devices, monitors, keyboards, mice, trackpads, microphones, and printers. Students also identify and describe common software applications such as web browsers, games, and word processors.

For example, students could create drawings of a computing system and label its major components with appropriate terminology. Students could then explain the function of each component (VAPA Visual Arts 2 5.0) (CA CCSS for ELA/Literacy SL.K.5, SL.K.6, SL.1.5, SL.1.6, SL.2.5, SL.2.6).

Alternatively, students could each be assigned a component of a computing system and arrange their bodies to represent the system. Students could then describe how their assigned component functions within the system (P.E.K.1, 1.1).

K-2.CS.3

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
K–2	K-2.CS.3	Describe basic hardware and software problems using accurate terminology.	Computing Systems	Troubleshooting	Testing, Communicating	6.2, 7.2

Descriptive Statement

Problems with computing systems have different causes. Accurate description of the problem aids users in finding solutions. Students 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.). Students at this level do not need to understand the causes of hardware and software problems.

For example, students could sort hardware and software terms on a word wall, and refer to the word wall when describing problems using "I see..." statements (e.g., "I see the pointer on the screen is missing", "I see that the computer will not turn on"). (CA CCSS for ELA/Literacy L.K.5.A, L.1.5.A, SL K.5, SL1.5, SL 2.5) (Visual Arts Kinder 5.2)

Alternatively, students could use appropriate terminology during collaborative conversations as they learn to debug, troubleshoot, collaborate, and think critically with technology. (CA CCSS for ELA/Literacy SL.K.1, SL.1.1, SL.2.1)

K-2.NI.4

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
K–2	K-2.NI.4	Model and describe how people connect to other people, places, information and ideas through a network.	Networks & the Internet	Network Communication & Organization	Abstraction	4.4

Descriptive Statement

Information is passed between multiple points (nodes) on a network. The internet is a network that enables people to connect with other people worldwide through many different points of connection. Students model ways that people communicate, find information, or acquire ideas through a network. Students use a network, such as the internet, to access information from multiple locations or devices.

For example, students could utilize a cloud-based platform to access shared documents or note-taking applications for group research projects, and then create a model (e.g., flowchart) to illustrate how this network aids collaboration (CA CCSS for ELA/Literacy W.K.7, W.1.7, W.2.7).

Alternatively, students could design devices that use light or sound to aid communication across distances (e.g., light source to send signals, paper cup and string "telephones," and a pattern of drum beats) and then describe how networks build connections (CA NGSS: 1-PS4-4).

K-2.NI.5

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
K–2	K-2.NI.5	Explain why people use passwords.	Networks & the Internet	Cybersecurity	Communicating	7.2

Descriptive Statement

Passwords protect information from unwanted use by others. When creating passwords, people often use patterns of familiar numbers and text to more easily remember their passwords. However, this may make the passwords weaker. Knowledge about the importance of passwords is an essential first step in learning about cybersecurity. Students explain that strong passwords are needed to protect devices and information from unwanted use.

For example, students could play a game of guessing a three-character code. In one version of the game, the characters are only numbers. In the second version, characters are numbers or letters. Students describe why it would take longer to guess the correct code in the second case.

Alternatively, students could engage in a collaborative discussion regarding passwords and their importance. Students may follow-up the discussion by exploring strong password components (combination of letters, numbers, and characters), creating their own passwords, and writing opinion pieces indicating reasons their passwords are strong (CA CCSS for ELA/Literacy SL.K.1, SL.1.1, SL 2.1, W.1.1, W.2.1).

K-2.NI.6

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
K–2	K-2.NI.6	Create patterns to communicate a message.	Networks & the Internet	Cybersecurity	Abstraction	4.4

Descriptive Statement

Connecting devices to a network or the internet provides great benefit, but care must be taken to protect devices and information from unauthorized access. Messages can be protected by using secret languages or codes. Patterns help to ensure that the intended recipient can decode the message. Students create a pattern that can be decoded and translated into a message.

For example, students could use a table to associate each text character with a number. Then, they could select a combination of text characters and use mathematical functions (e.g., simple arithmetic operations) to transform the numbers associated with the characters into a secret message. Using inverse functions, a peer could translate the secret message back into its original form (CA CCSS for Mathematics 2.OA.A.1, 2.OA.B.2).

Alternatively, students could use icons or invented symbols to represent patterns of beat, rhythm, or pitch to decode a musical phrase (VAPA Music K.1.1, 1.1.1, 2.1.1, 2.2.2).

K-2.DA.7

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
K–2	K-2.DA.7	Store, copy, search, retrieve, modify, and delete information using a computing device, and define the information stored as data.	Data & Analysis	Storage	Abstraction	4.2

Descriptive Statement

Information from the real world can be stored and processed by a computing device. When stored on a computing device, it is referred to as data. Data can include images, text documents, audio files, and video files. Students store, copy, search, retrieve, modify, and delete information using a computing device and define the information stored as data.

For example, students could produce a story using a computing device, storing it locally or remotely (e.g., in the cloud). They could then make a copy of the story for peer revision and editing. When the final copy of the story is complete, students delete any unnecessary files. They search for and retrieve data from a local or remote source, depending on where it was stored (CA CCSS for ELA/Literacy W.K.6, W.K.5, W1.6, W.1.5, W.2.6, W.2.5).

Alternatively, students could record their voices singing an age-appropriate song. They could store the data on a computing device, search for peers' audio files, retrieve their own files, and delete unnecessary takes (VAPA Music K.2.2, 1.2.2, 2.2.2).

K-2.DA.8

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
K–2	K-2.DA.8	Collect and present data in various visual formats.	Data & Analysis	Collection Visualization & Transformation	Communicating, Abstraction	7.1, 4.4

Descriptive Statement

Data can be collected and presented in various visual formats.

For example, students could measure temperature changes throughout a day. They could then discuss ways to display the data visually. Students could extend the activity by writing different narratives based on collected data, such as a story that begins in the morning when temperatures are low and one that begins in the afternoon when the sun is high and temperatures are higher (CA CCSS for ELA/Literacy RL.K.9, RL.1.9, RL.2.9, W.K.3, W.1.3, W.2.3).

Alternatively, students collect peers' favorite flavor of ice cream and brainstorm differing ways to display the data. In groups, students can choose to display and present the data in a format of their choice (CA CCSS for Mathematics K.MD.3, 1.MD.4, 2.MD.10).

K-2.DA.9

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
K–2	K-2.DA.9	Identify and describe patterns in data visualizations, such as charts or graphs, to make predictions.	Data & Analysis	Inference & Models	Abstraction	4.1

Descriptive Statement

Data can be used to make inferences or predictions about the world.

For example, students could record the number of each color of candy in a small packet. Then, they compare their individual data with classmates. Students could use the collected data to predict how many of each colored candy will be in a full size bag of like candy (CA CCSS for Mathematics K.MD.3, 1.MD.4, 2.MD.10).

Alternatively, students could sort and classify objects according to their properties and note observations. Students could then create a graph or chart of their observations and look for connections/relationships (e.g., items that are hard are usually also smooth, or items that are fluffy are usually also light in weight.) Students then look at pictures of additional objects and make predictions regarding the properties of the objects pictured (CA NGSS: 2-PS1-1, 2-PS1-2).

K-2.AP.10

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
K–2	K-2.AP.10	Model daily processes by creating and following algorithms to complete tasks.	Algorithms & Programming	Algorithms	Computational Problems, Abstraction	4.4, 3.2

Descriptive Statement

Algorithms are sequences of instructions that describe how to complete a specific task. Students create algorithms that reflect simple life tasks inside and outside of the classroom.

For example, students could create algorithms to represent daily routines for getting ready for school, transitioning through center rotations, eating lunch, and putting away art materials. Students could then write a narrative sequence of events (CA CCSS for ELA/Literacy W.K.3, W.1.3, W.2.3).

Alternatively, students could create a game or a dance with a specific set of movements to reach an intentional goal or objective (P.E K.2, 1.2, 2.2).

Additionally, students could create a map of their neighborhood and give step-by-step directions of how they get to school (HSS.K.4, 1.2, 2.2).

K-2.AP.11

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
K–2	K-2.AP.11	Model the way programs store data.	Algorithms & Programming	Variables	Abstraction	4.4

Descriptive Statement

Information in the real world can be represented in computer programs. Students model the digital storage of data by transforming real-world information into symbolic representations that include text, numbers, and images.

For example, after identifying symbols on a map and explaining what they represent in the real world, students could create their own symbols and corresponding legend to represent items on a map of their classroom (HSS.K.4.3, 1.2.3, 2.2.2).

Alternatively, students could invent symbols to represent beat and/or pitch. Students could then modify symbols within the notation and explain how the musical phrase changes (VAPA Music K.1.1, 1.1.1, 2.1.1, 2.2.2).

K-2.AP.12

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
K–2	K-2.AP.12	Create programs with sequences of commands and simple loops, to express ideas or address a problem.	Algorithms & Programming	Control, Modularity	Creating	5.2

Descriptive Statement

People create programs by composing sequences of commands that specify the precise order in which instructions should be executed. Loops enable programs to repeat a sequence of commands multiple times.

For example, students could follow simple movements in response to oral instructions. Students could then create a simple sequence of movement commands in response to a given problem (e.g., In how many ways can you travel from point A to point B?) and represent it as a computer program, using loops to repeat commands (VAPA Dance K.1.4, 1.2.3, 1.2.5, 1.2.8, 2.2.1, 2.2.2, 2.2.3).

Alternatively, on a mat with many different CVC words, students could program robots to move to words with a similar vowel sound. Students could look for multiple ways to solve the problem and simplify their solution by incorporating loops (CA CCSS for ELA/Literacy RF.K.2.D, RF.1.2.C).

K-2.AP.13

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
K–2	K-2.AP.13	Decompose the steps needed to solve a problem into a sequence of instructions.	Algorithms & Programming	Modularity	Computational Problems	3.2

Descriptive Statement

Decomposition is the act of breaking down tasks into simpler tasks.

For example, 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. In a visual programming environment, students could break down the steps needed to draw a shape (CA CCSS for Mathematics K.G.5, 1.G.1, 2.G.1).

Alternatively, students could decompose the planning of a birthday party into tasks such as: (1) Decide when and where it should be, (2) List friends and family to invite, (3) Send the invitations, (4) Bake a cake, (5) Decorate, etc.

K-2.AP.14

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
K–2	K-2.AP.14	Develop plans that describe a program's sequence of events, goals, and expected outcomes.	Algorithms & Programming	Program Development	Creating, Communicating	5.1, 7.2

Descriptive Statement

Creating a plan for what a program will do clarifies the steps that will be needed to create the program and can be used to check if a program runs as expected. Students create a planning document to illustrate their program's sequence of events, goals, and expected outcomes of what their program will do. Planning documents could include a story map, a storyboard, or a sequential graphic organizer, to illustrate their program's sequence of events, goals, and expected outcomes of what their program will do. Students at this level may complete the planning process with help from the teacher.

For example, students could create a storyboard or timeline that represents a family's history, leading to their current location of residence. Students could then create a plan for a program that animates the story of family locations (HSS 2.1.1) (CA CCSS for ELA/Literacy W.K.3, W.1.3, W.2.3).

K-2.AP.15

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
K–2	K-2.AP.15	Give attribution when using the ideas and creations of others while developing programs.	Algorithms & Programming	Program Development	Communicating	7.3

Descriptive Statement

Computing makes it easy to reuse and remix others' creations, and this comes with a level of responsibility. Students 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 if sharing work on a class blog or website. Proper attribution at this stage does not require formal citation, such as in a bibliography or works cited document.

For example, when creating an animation of the sun, moon, and stars using a blocks-based language, students could draw their own sun and use an image of the moon and stars from a website or a teammate. When students present the model to the class, they can orally give credit to the website or peer for the contributions (CA CCSS for ELA/Literacy SL.K.5, SL.1.5, SL.2.5) (NGSS.1-ESS1-1) (CA Model School Library Standards 2.3.b, 2.4.2.a).

K-2.AP.16

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
K–2	K-2.AP.16	Debug errors in an algorithm or program that includes sequences and simple loops.	Algorithms & Programming	Program Development	Testing	6.2

Descriptive Statement

Algorithms or programs may not always work correctly. Students 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.

For example, when given images placed in a random order, students could give step-by-step commands to direct a robot, or a student playing a robot, to navigate to the images in the correct sequence. Examples of images include storyboard cards from a familiar story (CA CCSS for ELA/Literacy RL.K.2, RL.1.2, RL.2.2) and locations of the sun at different times of the day (CA NGSS: 1-ESS1-1).

Alternatively, students could "program" the teacher or another classmate by giving precise instructions to make a peanut butter and jelly sandwich or navigate around the classroom. When the teacher or classmate doesn't respond as intended, students correct their commands. Additionally, students could receive a partially completed soundboard program that has a variety of animals programmed to play a corresponding sound when the user touches them. Students correct any sounds that don't match the animal (e.g., if the cat moos, students change the moo sound to meow).

K-2.AP.17

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
K–2	K-2.AP.17	Describe the steps taken and choices made during the iterative process of program development.	Algorithms & Programming	Program Development	Communicating	7.2

Descriptive Statement

Program developers make choices and iterate to continually refine their product. At this stage, students explain or write about the goals and expected outcomes of the programs they create and the choices that they made when creating programs. Students could use coding journals, discussions with a teacher, class presentations, or blogs.

For example, students could use a combination of images, verbal reflections, a physical model, and/or written text to show the step-by-step process taken to develop a program such as cutting and pasting coding commands into a journal, using manipulatives that represent different commands and control structures, and taking screenshots of code and adding to a digital journal. This iterative process could be documented via a speech, journal, one on one conference with teacher or peer, small group conference, or blog (CA CCSS for ELA/Literacy SL.K.5, SL.1.5, SL.2.5) (CA NGSS: K-2-ETS1.2).

K-2.IC.18

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
K–2	K-2.IC.18	Compare how people lived and worked before and after the adoption of new computing technologies.	Impacts of Computing	Culture	Computational Problems	3.1

Descriptive Statement

Computing technologies have changed the way people live and work. Students describe the positive and negative impacts of these changes.

For example, as a class, students could create a timeline that includes advancements in computing technologies. Each student could then choose an advancement from the timeline and make a graphic organizer noting how people's lives were different before and after its introduction into society. Student responses could include: 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 (HSS.K.6.3).

Alternatively, students could retell or dramatize stories, myths, and fairy tales from two distinct time periods before and after a particular computing technology had been introduced. For example, the setting of one story could take place before smartphones had been invented, while a second setting could take place with smartphones in use by characters in the story. Students could note the positive and negative aspects of smartphones on the daily lives of the characters in the story (VAPA Theatre Arts K.3.1, K.3.2, 1.2.2, 2.3.2) (CA CCSS for ELA/Literacy RL.K.2, RL.K.9, RL.1., RL.1.9, RL.2.2, RL.2.9).

K-2.IC.19

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
K–2	K-2.IC.19	Work respectfully and responsibly with others when communicating electronically.	Impacts of Computing	Social Interactions	Collaborating	2.1

Descriptive Statement

Electronic communication facilitates positive interactions, such as sharing ideas with many people, but the public and anonymous nature of electronic communication also allows intimidating and inappropriate behavior in the form of cyberbullying. Responsible electronic communication includes limiting access to personably identifiable information. Students learn and use appropriate behavior when communicating electronically (often called "netiquette").

For example, students could share their work on a classroom blog or in other collaborative spaces online, taking care to avoid sharing information that is inappropriate or that could personally identify themselves to others (CA CCSS for ELA/Literacy W.K.6, W.1.6, W.21.6).

Alternatively, students could provide feedback to others on their work in a kind and respectful manner. They could learn how written words can be easily misinterpreted and may seem negative when the intention may be to express confusion, give ideas, or prompt further discussion. They could also learn to identify harmful behavior on collaborative spaces and intervening to find the proper authority to help (CA CCSS for ELA/Literacy W.K.5, W.1.5, W.2.5) (HSS 1.1.2).

K-2.IC.20

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
K–2	K-2.IC.20	Describe approaches and rationales for keeping login information private, and for logging off of devices appropriately.	Impacts of Computing	Safety Law & Ethics	Computational Problems	3.1

Descriptive Statement

People use computing technology in ways that can help or hurt themselves and/or others. Harmful behaviors, such as sharing passwords or other private information and leaving public devices logged in should be recognized and avoided. Students keep login information private, log off of devices appropriately, and discuss the importance of these practices.

For example, while learning about individual responsibility and citizenship, students could create a "privacy folder" to store login information, and keep this folder in a secure location that is not easily seen and accessed by classmates. Students could discuss the relative benefits and impacts of choosing to store passwords in a folder online versus on paper. They could also describe how using the same login and password across many systems and apps could lead to significant security issues and requires even more vigilance in maintaining security (HSS K.1).

Alternatively, students can write an informational piece regarding the importance of keeping login information private and logging off of public devices (CA CCSS for ELA/Literacy W.K.2, W.1.2, W.2.2).

3–5

3-5.CS.1

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
3–5	3-5.CS.1	Describe how computing devices connect to other components to form a system.	Computing Systems	Devices	Communicating	7.2

Descriptive Statement

Computing devices often depend on other devices or components. Students describe physical and wireless connections to other components, including both input devices (e.g., keyboards, sensors, remote controls, microphones) and output devices (e.g., 3D printers, monitors, speakers).

For example, students could describe the relationship among the heart, lungs, muscles, blood, and oxygen during physical activity and then compare this to how a mouse, keyboard, printer, and desktop computer connect and interact to allow for input, processing, and output (P.E.3.4.7).

Alternatively, when describing how light reflected from objects enters the eye and is then transferred to the brain to construct a visual image, students could compare this to a computing system that uses programming to construct a visual image when data is transferred and constructed/reconstructed through a keyboard, camera, or other components (CA NGSS: 4-PS4-2).

3-5.CS.2

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
3–5	3-5.CS.2	Demonstrate how computer hardware and software work together as a system to accomplish tasks.	Computing Systems	Hardware & Software	Abstraction	4.4

Descriptive Statement

Hardware and software are both needed to accomplish tasks with a computing device. Students create a model to illustrate ways in which hardware and software work as a system. 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. At this level, a model should only include the basic elements of a computer system, such as input, output, processor, sensors, and storage.

For example, students could create a diagram or flow chart to indicate how a keyboard, desktop computer, monitor, and word processing software interact with each other. The keyboard (hardware) detects a key press, which the operating system and word processing application (software) displays as a new character that has been inserted into the document and is visible through the monitor (hardware). Students could also create a model by acting out the interactions of these different hardware and software components.

Alternatively, when describing that animals and people receive different types of information through their senses, process the information in their brain, and respond to the information in different ways, students could compare this to the interaction of how the information traveling through a computer from mouse to processor are similar to signals sent through the nervous system telling our brain about the world around us to prompt responses (CA NGSS: 4-LS1-2).

3-5.CS.3

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
3–5	3-5.CS.3	Determine potential solutions to solve simple hardware and software problems using common troubleshooting strategies.	Computing Systems	Troubleshooting	Testing	6.2

Descriptive Statement

Although computing systems vary, common troubleshooting strategies can be used across many different systems. Students use troubleshooting strategies to identify problems that could include a device not responding, lacking power, lacking a network connection, an app crashing, not playing sounds, or password entry not working. Students use and develop various solutions to address these problems. Solutions may include rebooting the device, checking for power, checking network availability, opening and closing an app, making sure speakers are turned on or headphones are plugged in, and making sure that the caps lock key is not on.

For example, students could prepare for and participate in a collaborative discussion in which they identify and list computing system problems and then describe common successful fixes (CA CCSS for ELA/Literacy SL.3.1, SL.4.1, SL.5.1).

Alternatively, students could write informative/explanatory texts, create a poster, or use another medium of communication to examine common troubleshooting strategies and convey these ideas and information clearly (CA CCSS for ELA/Literacy W.3.2, W.4.2, W.5.2).

3-5.NI.4

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
3–5	3-5.NI.4	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.	Networks & the Internet	Network Communication & Organization	Abstraction	4.4

Descriptive Statement

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 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 in which they physically act this out.

For example, students could design a structure using building blocks or other materials with the intention of re-engineering it in another location, just as early Americans did after the intercontinental railroad was constructed in the 1850s (HSS.4.4.1, 4.4.2). Students could deconstruct the designed structure, place materials into specific containers (or plastic bags/brown paper bags/etc.), and develop instructions on how to recreate the structure once each container arrives at its intended destination (CA NGSS: 3-5-ETS1).

For example, students could cut up a map of the United States by state lines. Students could then place the states in envelopes and transmit the "packets" through a physical network, represented by multiple students spreading out in arms reach of at least two others. At the destination, the student who receives the packets reassembles the individual states back into a map of the United States (HSS 5.9).

Alternatively, students could perform a similar activity with a diatonic scale, cutting the scale into individual notes. Each note, in order, should be placed into a numbered envelope based on its location on the scale. These envelopes can be transmitted across the network of students and reassembled at the destination (VAPA Music 4.1.2).

3-5.NI.5

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
3–5	3-5.NI.5	Describe physical and digital security measures for protecting personal information.	Networks & the Internet	Cybersecurity	Computational Problems	3.1

Descriptive Statement

Personal information can be protected physically and digitally. Cybersecurity is the protection from unauthorized use of electronic data, or the measures taken to achieve this. Students identify what personal information is and the reasons for protecting it. Students describe physical and digital approaches for protecting personal information such as using strong passwords and biometric scanners.

For example, students could engage in a collaborative discussion orally or in writing regarding 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 keep operating systems updated and use anti-virus software to protect data and systems. Students could also discuss physical measures that can be used to protect data including biometric scanners, locked doors, and physical backups (CA CCSS for ELA/Literacy SL.3.1, SL.4.1, SL.5.1).

3-5.NI.6

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
3–5	3-5.NI.6	Create patterns to protect information from unauthorized access.	Networks & the Internet	Cybersecurity	Abstraction	4.4

Descriptive Statement

Encryption is the process of converting information or data into a code, especially to prevent unauthorized access. At this level, students use patterns as a code for encryption, to protect information. Patterns should be decodable to the party for whom the message is intended, but difficult or impossible for those with unauthorized access.

For example, students could create encrypted messages via flashing a flashlight in Morse code. Other students could decode this established language even if it wasn't meant for them. To model the idea of protecting data, students should create their own variations on or changes to Morse code. This ensures that when a member of that group flashes a message only other members of their group can decode it, even if other students in the room can see it (CA NGSS: 4-PS4-3).

Alternatively, students could engage in a CS Unplugged activity that models public key encryption: One student puts a paper containing a written secret in a box, locks it with a padlock, and hands the box to a second student. Student 2 puts on a second padlock and hands it back. Student 1 removes her lock and hands the box to student 2 again. Student 2 removes his lock, opens the box, and has access to the secret that student 1 sent him. Because the box always contained at least one lock while in transit, an outside party never had the opportunity to see the message and it is protected.

3-5.DA.7

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
3–5	3-5.DA.7	Explain that the amount of space required to store data differs based on the type of data and/or level of detail.	Data & Analysis	Storage	Abstraction	4.2

Descriptive Statement

All saved data requires space to store it, whether locally or not (e.g., on the cloud). Music, images, video, and text require different amounts of storage. Video will often require more storage and different format than music or images alone because video combines both. The level of detail represented by that data also affects storage requirements. For instance, two pictures of the same object can require different amounts of storage based upon their resolution, and a high-resolution photo could require more storage than a low-resolution video. Students select appropriate storage for their data.

For example, students could create an image using a standard drawing app. They could save the image in different formats (e.g., .png, .jpg, .pdf) and compare file sizes. They should also notice that different file sizes can result in differences in quality or resolution (e.g., some pictures could be more pixelated while some could be sharper).

Alternatively, in an unplugged activity, students could represent images by coloring in squares within a large grid. They could model how a larger grid requires more storage but also represents a clearer image (i.e., higher resolution).

3-5.DA.8

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
3–5	3-5.DA.8	Organize and present collected data visually to highlight relationships and support a claim.	Data & Analysis	Collection Visualization & Transformation	Communicating	7.1

Descriptive Statement

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, students could create and administer electronic surveys to their classmates. Possible topics could include favorite books, family heritage, and after school activities. Students could then create digital displays of the data they have collected such as column histogram charts showing the percent of respondents in each grade who selected a particular favorite book. Finally, students could make quantitative statements supported by the data such as which books are more appealing to specific ages of students. As an extension, students could write an opinion piece stating a claim and supporting it with evidence from the data they collected (CA CCSS for Mathematics 3.MD.3, 4.MD.4, 5.MD.2) (CA CCSS for ELA/Literacy W.3.1, W.4.1, W.5.1).

Alternatively, students could represent data in tables and graphical displays to describe weather experienced in the last several years. They could select the type of graphical display based on the specific data represented (e.g., daily high/low temperatures on a scatter plot, average temperatures for a month across years in a column chart). Students could then make a claim about expected weather in future months based on the data (CA NGSS: 3-ESS2-1).

3-5.DA.9

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
3–5	3-5.DA.9	Use data to highlight and/or propose relationships, predict outcomes, or communicate ideas.	Data & Analysis	Inference & Models	Communicating	7.1

Descriptive Statement

The accuracy of data analysis is related to how the data is represented. Inferences or predictions based on data are less likely to be accurate if the data is insufficient, incomplete, or inaccurate or if the data is incorrect in some way. Additionally, people select aspects and subsets of data to be transformed, organized, and categorized. Students should be able to refer to data when communicating an idea, in order to highlight and/or propose relationships, predict outcomes, highlight different views and/or communicate insights and ideas.

For example, students can be provided a scenario in which they are city managers who have a specific amount of funds to improve a city in California. Students can collect data of a city concerning land use, vegetation, wildlife, climate, population density, services and transportation (HSS.4.1.5) to determine and present what area needs to be focused on to improve a problem. Students can compare their data and planned use of funds with peers, clearly communicating or predict outcomes based on data collected (CA CCCS for ELA/Literacy SL.3.1, SL.4.1, SL.5.1).

Alternatively, students could record the temperature at noon each day 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, the data would be incomplete and 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 use data provided by a governmental weather agency (CA NGSS: 3-ESS2-1).

3-5.AP.10

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
3–5	3-5.AP.10	Compare and refine multiple algorithms for the same task and determine which is the most appropriate.	Algorithms & Programming	Algorithms	Testing, Computational Problems	6.3, 3.3

Descriptive Statement

Different algorithms can achieve the same result, though sometimes one algorithm might be more appropriate for a specific solution. Students examine different ways to solve the same task and decide which would be the better solution for the specific scenario.

For example, students could use a map and create multiple algorithms to model the early land and sea routes to and from European settlements in California. They could then compare and refine their algorithms to reflect faster travel times, shorter distances, or avoid specific characteristics, such as mountains, deserts, ocean currents, and wind patterns (HSS.4.2.2).

Alternatively, students could identify multiple algorithms for decomposing a fraction into a sum of fractions with the same denominator and record each decomposition with an equation (e.g., 2 1/8 = 1 + 1 + 1/8 = 8/8 + 8/8 + 1/8). Students could then select the most efficient algorithm (e.g., fewest number of steps) (CA CCSS for Mathematics 4.NF.3b).

Additionally, students could compare algorithms that describe how to get ready for school and modify them for supporting different goals including having time to care for a pet, being able to talk with a friend before classes start, or taking a longer route to school to accompany a younger sibling to their school first. Students could then write an opinion piece, justifying with reasons their selected algorithm is most appropriate (CA CCSS for ELA/Literacy W.3.1, W.4.1, W.5.1).

3-5.AP.11

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
3–5	3-5.AP.11	Create programs that use variables to store and modify data.	Algorithms & Programming	Variables	Creating	5.2

Descriptive Statement

Variables are used to store and modify data. Students use variables in programs they create. At this level, students may need guidance in identifying when to create variables (i.e., performing the abstraction).

For example, students could create a game to represent predators and prey in an ecosystem. They could declare a "score" variable, assign it to 0 at the start of the game, and add 1 (increment) the score each time the predator captures its prey. They could also declare a second "numberOfLives" variable, assign it to 3 at the start of the game, and subtract 1 (decrement) each time a prey is captured. They could program the game to end when "numberOfLives" equals 0 (CA NGSS: 5-LS2-1) (CA CCSS for Mathematics 5.OA.3).

Alternatively, when students create programs to draw regular polygons, they could use variables to store the line size, line color, and/or side length. Students can extend learning by creatively combining a variety of polygons to create digital artwork, comparing and contrasting this to another work of art made by the use of different art tools and media, such as watercolor or tempera paints (CA CCSS for Mathematics 3.G.1) (VAPA Visual Arts 3.1.4).

3-5.AP.12

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
3–5	3-5.AP.12	Create programs that include events, loops, and conditionals.	Algorithms & Programming	Control	Creating	5.2

Descriptive Statement

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. Conditionals allow for the execution of a portion of code in a program when a certain condition is true. Loops allow for the repetition of a sequence of code multiple times.

For example, students could program an interactive map of the United States of America. They could use events to initiate a question when the user clicks on a state and conditionals to check whether the user input is correct. They could use loops to repeat the question until the user answers correctly or to control the length of a "congratulations" scenario that plays after a correct answer (HSS.5.9).

Alternatively, students could write a math fluency game that asks products of two one-digit numbers and then uses a conditional to check whether or not the answer that was entered is correct. They could use a loop to repeatedly ask another question. They could use events to allow the user to click on a green button to play again or a red button to end the game (CA CCSS for Mathematics 3.OA.7).

Additionally, students could create a program as a role-playing game based on a literary work. Loops could be used to animate a character's movement. When reaching a decision point in the story, an event could initiate the user to type a response. A conditional could change the setting or have the story play out differently based on the user input (CA CCSS for ELA/Literacy RL.5.3).

3-5.AP.13

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
3–5	3-5.AP.13	Decompose problems into smaller, manageable tasks which may themselves be decomposed.	Algorithms & Programming	Modularity	Computational Problems	3.2

Descriptive Statement

Decomposition is the act of breaking down tasks into simpler tasks. This manages complexity in the problem solving and program development process.

For example, students could create an animation to represent a story they have written. Students write a story and then break it down into different scenes. For each scene, they would select a background, place characters, and program actions in that scene (CA CCSS for ELA/Literacy W.3.3, W.4.3, W.5.3).

Alternatively, students could create a program to allow classmates to present data collected in an experiment. For example, if students collected rain gauge data once per week for 3 months, students could break down the program tasks: (1) ask the user to input 12 weeks' worth of data, (2) process the data (e.g., add the first four entries to calculate the rain amount for month 1, convert to metric system measurements), and (3) direct the creation or resizing of objects (e.g., one rectangular chart bar for each month) to represent the total number of rainfall for that month (CA NGSS: 3-ETS-1-2) (CA CCSS for Mathematics 3.MD.2).

3-5.AP.14

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
3–5	3-5.AP.14	Create programs by incorporating smaller portions of existing programs, to develop something new or add more advanced features.	Algorithms & Programming	Modularity, Program Development	Abstraction, Creating	4.2, 5.3

Descriptive Statement

Programs can be broken down into smaller parts, which can be incorporated into new or existing programs. Students incorporate predefined functions into their original designs. At this level, students do not need to understand all of the underlying implementation details of the abstractions that they use.

For example, students could use code from a ping pong animation to make a ball bounce in a new basketball game. They could also incorporate code from a single-player basketball game to create a two-player game with slightly different rules.

Alternatively, students could remix an animated story and add their own conclusion and/or additional dialogue (CA CCSS for ELA/Literacy W.3.3.B, W.3.3.D, W.4.3.B, W.4.3.E, W.5.3.B, W.5.3.E).

Additionally, when creating a game that occurs on the moon or planets, students could incorporate and modify code that simulates gravity on Earth. They could modify the strength of the gravitational force based on the mass of the planet or moon (CA NGSS: 5-PS2-1).

3-5.AP.15

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
3–5	3-5.AP.15	Use an iterative process to plan and develop a program by considering the perspectives and preferences of others.	Algorithms & Programming	Program Development	Inclusion, Creating	1.1, 5.1

Descriptive Statement

Planning is an important part of the iterative process of program development. Students gain a basic understanding of the importance and process of planning before beginning to write code for a program. They plan the development of a program by outlining key features, time and resource constraints, and user expectations. Students should document the plan as, for example, a storyboard, flowchart, pseudocode, or story map.

For example, students could collaborate with a partner to plan and develop a program that graphs a function. They could iteratively modify the program based on feedback from diverse users, such as students who are color blind and may have trouble differentiating lines on a graph based on the color (CA CCSS for Mathematics 5.G.1, 5.G.2).

Alternatively, students could plan as a team to develop a program to display experimental data. They could implement the program in stages, generating basic displays first and then soliciting feedback from others on how easy it is to interpret (e.g., are labels clear and readable?, are lines thick enough?, are titles understandable?). Students could iteratively improve their display to make it more readable and to better support the communication of the finding of the experiment (NGSS.3-5-ETS1-1, 3-5-ETS1-2, 3-5-ETS1-3).

3-5.AP.16

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
3–5	3-5.AP.16	Observe intellectual property rights and give appropriate attribution when creating, remixing, or combining programs.	Algorithms & Programming	Program Development	Creating, Communicating	5.2, 7.3

Descriptive Statement

Intellectual property rights can vary by country, but copyright laws give the creator of a work a set of rights and prevents others from copying the work and using it in ways that they may not like. Students consider common licenses that place limitations or restrictions on the use of others' work, such as images and music downloaded from the internet. When incorporating the work of others, students attribute the work. At this level, students could give attribution by including credits or links directly in their programs, code comments, or separate project pages.

For example, when making a program to model the life cycle of a butterfly, students could modify and reuse an existing program that describes the life cycle of a frog. Based on their research, students could identify and use Creative Commons-licensed or public domain images and sounds of caterpillars and butterflies. Students give attribution by properly citing the source of the original piece as necessary (CA NGSS: 3-LS-1-1) (CA CCSS for ELA/Literacy W.3.8, W.4.8, W.5.8).

Alternatively, when creating a program explaining the structure of the United States government, students find Creative Commons-licensed or public domain images to represent the three branches of government and attribute ownership of the images appropriately. If students find and incorporate an audio file of a group playing part of the national anthem, they appropriately give attribution on the project page (HSS.3.4.4).

3-5.AP.17

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
3–5	3-5.AP.17	Test and debug a program or algorithm to ensure it accomplishes the intended task.	Algorithms & Programming	Program Development	Testing	6.2

Descriptive Statement

Programs do not always run properly. Students need to understand how to test and make necessary corrections to their programs to ensure they run properly. Students successfully identify and fix errors in (debug) their programs and programs created by others. Debugging strategies at this level may include testing to determine the first place the solution is in error and fixing accordingly, leaving "breadcrumbs" in a program, and soliciting assistance from peers and online resources.

For example, when students are developing a program to control the movement of a robot in a confined space, students test various inputs that control movement of the robot to make sure it behaves as intended (e.g., if an input would cause the robot to move past a wall of the confined space, it should not move at all) (CA NGSS: 3-5-ETS1-3).

Additionally, students could test and debug an algorithm by tracing the inputs and outputs on a whiteboard. When noticing "bugs" (errors), students could identify what was supposed to happen and step through the algorithm to locate and then correct the error.

3-5.AP.18

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
3–5	3-5.AP.18	Perform different roles when collaborating with peers during the design, implementation, and review stages of program development.	Algorithms & Programming	Program Development	Collaborating	2.2

Descriptive Statement

Collaborative computing is the process of creating computational artifacts by working in pairs or on teams. It involves asking for the contributions and feedback of others. Effective collaboration can often lead to better outcomes than working independently. With teacher guidance, students take turns in different roles during program development, such as driver, navigator, notetaker, facilitator, and debugger, as they design and implement their program.

For example, while taking on different roles during program development, students could create and maintain a journal about their experiences working collaboratively (CA CCSS for ELA/Literacy W.3.10, W.4.10, W.5.10) (CA NGSS: 3-5-ETS1-2).

3-5.AP.19

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
3–5	3-5.AP.19	Describe choices made during program development using code comments, presentations, and demonstrations.	Algorithms & Programming	Program Development	Communicating	7.2

Descriptive Statement

People communicate about their code to help others understand and use their programs. Explaining one's design choices gives others a better understanding of one's work. Students may explain their step-by-step process of creating a program in a presentation or demonstration of their personal code journals. They describe how comments within code organize thought and process during the development of the program.

For example, students could describe the decision to have the score in a game flash when it can be rounded to 100 by writing a comment in the code (CA CCSS for Mathematics 3.NBT.1).

Alternatively, students could present their overall program development experience and justify choices made by using storyboards, annotated images, videos, and/or journal entries (CA CCSS for ELA/Literacy SL.3.4, SL.4.4, SL.5.4, SL.3.5, SL.4.5, SL.5.5) (CA NGSS: 3-5-ETS1-1, 3.5-ETS1-2, 3.5-ETS1-3).

3-5.IC.20

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
3–5	3-5.IC.20	Discuss computing technologies that have changed the world, and express how those technologies influence, and are influenced by, cultural practices.	Impacts of Computing	Culture	Computational Problems	3.1

Descriptive Statement

New computing technologies are 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, discuss topics that relate to the history of computing technologies and changes in the world due to these technologies. Topics could be based on current news content, such as robotics, wireless internet, mobile computing devices, GPS systems, wearable computing, and how social media has influenced social and political changes.

For example, students could conduct research in computing technologies that impact daily life such as self-driving cars. They engage in a collaborative discussion describing impacts of these advancements (e.g., self-driving cars could reduce crashes and decrease traffic, but there is a cost barrier to purchasing them) (CA CCSS for ELA/Literacy W.3.7, W.4.7, W.5.7, SL.3.1, SL.4.1, SL.5.1).

Alternatively, students could discuss how technological advancements affected the entertainment industry and then compare and contrast the impacts on audiences. For instance, people with access to high-speed internet may be able to choose to utilize streaming media (which may cost less than traditional media options), but those in rural areas may not have the same access and be able to reap those benefits (VAPA Theatre Arts 4.3.2, 4.4.2).

3-5.IC.21

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
3–5	3-5.IC.21	Propose ways to improve the accessibility and usability of technology products for the diverse needs and wants of users.	Impacts of Computing	Culture	Inclusion	1.2

Descriptive Statement

The development and modification of computing technology is driven by people's needs and wants and can affect groups differently. Students anticipate the needs and wants of diverse end users and propose ways to improve access and usability of technology, with consideration of potential perspectives of users with different backgrounds, ability levels, points of view, and disabilities.

For example, students could research a wide variety of disabilities that would limit the use of traditional computational tools for the creation of multimedia artifacts, including digital images, songs, and videos. Students could then brainstorm and propose new software that would allow students that are limited by the disabilities to create similar artifacts in new ways (e.g., graphical display of music for the deaf, the sonification of images for visually impaired students, voice input for those that are unable to use traditional input like the mouse and the keyboard) (CA CCSS for ELA/Literacy W.3.7, W.4.7, W.5.7).

Alternatively, as they anticipate unique user needs, students may consider using both speech and text 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 (CA NGSS: 3-5-ETS1-1, 3-5-ETS1-2, 3-5-ETS1-3).

3-5.IC.22

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
3–5	3-5.IC.22	Seek and explain the impact of diverse perspectives for the purpose of improving computational artifacts.	Impacts of Computing	Social Interactions	Inclusion	1.1

Descriptive Statement

Computing technologies enable global collaboration and sharing of ideas. Students solicit feedback from a diverse group of users and creators and explain how this input improves their computational artifacts.

For example, students could seek feedback from classmates via user surveys, in order to create an idea and then make a claim as to how to improve the overall structure and function of their computational artifact. Using the feedback students could write an opinion piece supporting their claim (CA CCSS for ELA/Literacy W.3.1, W.4.1, W.5.1).

Alternatively, with guidance from their teacher, students could use video conferencing tools, shared documents, or other online collaborative spaces, such as blogs, wikis, forums, or website comments, to gather and synthesize feedback from individuals and groups about programming projects (CA CCSS for ELA/Literacy SL.3.1, SL.4.1, SL.5.1).

3-5.IC.23

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
3–5	3-5.IC.23	Describe reasons creators might limit the use of their work.	Impacts of Computing	Safety Law & Ethics	Communicating	7.3

Descriptive Statement

Ethical complications arise from the opportunities provided by computing. With the ease of sending and receiving copies of media on the internet, in formats such as video, photos, and music, students consider the opportunities for unauthorized use, such as online piracy and disregard of copyrights. The license of a downloaded image or audio file may restrict modification, require attribution, or prohibit use entirely.

For example, students could take part in a collaborative discussion regarding reasons why musicians who sell their songs in digital format choose to license their work so that they can earn money for their creative efforts. If others share the songs without paying for them, the musicians do not benefit financially and may struggle to produce music in the future (CA CCSS for ELA/Literacy SL.3.1, SL.4.1, SL.5.1).

Alternatively, students could review the rights and reproduction guidelines for digital artifacts on a publicly accessible media source. They could then state an opinion with reasons they believe these guidelines are in place (CA CCSS for ELA/Literacy W.3.1, W.4.1, W.5.1)

6–8

6-8.CS.1

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
6–8	6-8.CS.1	Design modifications to computing devices in order to improve the ways users interact with the devices.	Computing Systems	Devices	Inclusion, Computational Problems	1.2, 3.3

Descriptive Statement

Computing devices can extend the abilities of humans, but design considerations are critical to make these devices useful. Students suggest modifications to the design of computing devices and describe how these modifications would improve usability.

For example, students could create a design for the screen layout of a smartphone that is more usable by people with vision impairments or hand tremors. They might also design how to use the device as a scanner to convert text to speech.

Alternatively, students could design modifications for a student ID card reader to increase usability by planning for scanner height, need of scanner device to be connected physically to the computer, robustness of scanner housing, and choice of use of RFID or line of sight scanners (CA NGSS: MS-ETS1-1).

6-8.CS.2

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
6–8	6-8.CS.2	Design a project that combines hardware and software components to collect and exchange data.	Computing Systems	Hardware & Software	Creating	5.1

Descriptive Statement

Collecting and exchanging data involves input, output, storage, and processing. When possible, students select the components for their project designs by considering tradeoffs between factors such as functionality, cost, size, speed, accessibility, and aesthetics. Students do not need to implement their project design in order to meet this standard.

For example, students could design a mobile tour app that displays information relevant to specific locations when the device is nearby or when the user selects a virtual stop on the tour. They select appropriate components, such as GPS or cellular-based geolocation tools, textual input, and speech recognition, to use in their project design.

Alternatively, students could design a project that uses a sensor to collect the salinity, moisture, and temperature of soil. They may select a sensor that connects wirelessly through a Bluetooth connection because it supports greater mobility, or they could instead select a physical USB connection that does not require a separate power source (CA NGSS: MS-ETS1-1, MS-ETS1-2).

6-8.CS.3

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
6–8	6-8.CS.3	Systematically apply troubleshooting strategies to identify and resolve hardware and software problems in computing systems.	Computing Systems	Troubleshooting	Testing	6.2

Descriptive Statement

When problems occur within computing systems, it is important to take a structured, step-by-step approach to effectively solve the problem 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. 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.

For example, students could work through a checklist of solutions for connectivity problems in a lab of computers connected wirelessly or through physical cables. They could also search for technical information online and engage in technical reading to create troubleshooting documents that they then apply (CA CCSS for ELA/Literacy RST.6-8.10).

Alternatively, students could explore and utilize operating system tools to reset a computer's default language to English.

Additionally, students could swap out an externally-controlled sensor giving fluctuating readings with a new sensor to check whether there is a hardware problem.

6-8.NI.4

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
6–8	6-8.NI.4	Model the role of protocols in transmitting data across networks and the internet.	Networks & the Internet	Network Communication & Organization	Abstraction	4.4

Descriptive Statement

Protocols are rules that define how messages between computers are sent. They determine how quickly and securely information is transmitted across networks, as well as how to handle errors in transmission. Students model how data is sent using protocols to choose the fastest path and to deal with missing information. Knowledge of the details of how specific protocols work is not expected. The priority at this grade level is understanding the purpose of protocols and how they enable efficient and errorless communication.

For example, students could devise a plan for sending data representing a textual message and devise a plan for resending lost information.

Alternatively, students could devise a plan for sending data to represent a picture, and devise a plan for interpreting the image when pieces of the data are missing.

Additionally, students could model the speed of sending messages by Bluetooth, Wi-Fi, or cellular networks and describe ways errors in data transmission can be detected and dealt with.

6-8.NI.5

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
6–8	6-8.NI.5	Explain potential security threats and security measures to mitigate threats.	Networks & the Internet	Cybersecurity	Computational Problems	3.1, 3.3

Descriptive Statement

Cybersecurity is an important field of study and it is valuable for students to understand the need for protecting sensitive data. Students identify multiple methods for protecting data and articulate the value and appropriateness for each method. Students are not expected to implement or explain the implementation of such technologies.

For example, students could explain the importance of keeping passwords hidden, setting secure router administrator passwords, erasing a storage device before it is reused, and using firewalls to restrict access to private networks.

Alternatively, students could explain the importance of two-factor authentication and HTTPS connections to ensure secure data transmission.

6-8.NI.6

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
6–8	6-8.NI.6	Apply multiple methods of information protection to model the secure transmission of information.	Networks & the Internet	Cybersecurity	Abstraction	4.4

Descriptive Statement

Digital information is protected using a variety of cryptographic techniques. Cryptography is essential to many models of cybersecurity. At its core, cryptography has a mathematical foundation. Cryptographic encryption can be as simple as letter substitution or as complicated as modern methods used to secure networks and the internet. Students encode and decode messages using encryption methods, and explore different levels of complexity used to hide or secure information.

For example, students could identify methods of secret communication used during the Revolutionary War (e.g., ciphers, secret codes, invisible ink, hidden letters) and then secure their own methods such as substitution ciphers or steganography (i.e., hiding messages inside a picture or other data) to compose a message from either the Continental Army or British Army (HSS.8.1).

Alternatively, students could explore functions and inverse functions for encryption and decryption and consider functions that are complex enough to keep data secure from their peers (CA CCSS for Mathematics 8.F.1).

6-8.DA.7

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
6–8	6-8.DA.7	Represent data in multiple ways.	Data & Analysis	Storage	Abstraction	4.4

Descriptive Statement

Computers store data as sequences of 0s and 1s (bits). Software translates to and from this low-level representation to higher levels that are understandable by people. Furthermore, higher level data can be represented in multiple ways, such as the digital display of a color and its corresponding numeric RGB value, or a bar graph, a pie chart, and table representation of the same data in a spreadsheet.

For example, students could use a color picker to explore the correspondence between the digital display or name of a color (high-level representations) and its RGB value or hex code (low-level representation).

Alternatively, students could translate a word (high-level representation) into Morse code or its corresponding sequence of ASCII codes (low-level representation).

6-8.DA.8

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
6–8	6-8.DA.8	Collect data using computational tools and transform the data to make it more useful.	Data & Analysis	Collection Visualization & Transformation	Communicating	7.1

Descriptive Statement

Data collection has become easier and more ubiquitous. The cleaning of data is an important transformation for ensuring consistent format, reducing noise and errors (e.g., removing irrelevant responses in a survey), and/or making it easier for computers to process. Students build on their ability to organize and present data visually to support a claim, understanding when and how to transform data so information can be more easily extracted. Students also transform data to highlight or expose relationships.

For example, students could use computational tools to collect data from their peers regarding the percentage of time technology is used for school work and entertainment, and then create digital displays of their data and findings. Students could then transform the data to highlight relationships representing males and females as percentages of a whole instead of as individual counts (CA CCSS for Mathematics 6.SP.4, 7.SP.3, 8.SP.1, 8.SP.4).

Alternatively, students could collect data from online forms and surveys, from a sensor, or by scraping a web page, and then transform the data to expose relationships. They could highlight the distribution of data (e.g., words on a web page, readings from a sensor) by giving quantitative measures of center and variability (CA CCSS for Mathematics 6.SP.5.c, 7.SP.4).

6-8.DA.9

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
6–8	6-8.DA.9	Test and analyze the effects of changing variables while using computational models.	Data & Analysis	Inference & Models	Abstraction, Testing	4.4, 6.1

Descriptive Statement

Variables within a computational model may be changed, in order to alter a computer simulation or to more accurately represent how various data is related. Students interact with a given model, make changes to identified model variables, and observe and reflect upon the results.

For example, students could test a program that makes a robot move on a track by making changes to variables (e.g., height and angle of track, size and mass of the robot) and discussing how these changes affect how far the robot travels (CA NGSS: MS-PS2-2).

Alternatively, students could test a game simulation and change variables (e.g., skill of simulated players, nature of opening moves) and analyze how these changes affect who wins the game (CA NGSS: MS-ETS1-3).

Additionally, students could modify a model for predicting the likely color of the next pick from a bag of colored candy and analyze the effects of changing variables representing the common color ratios in a typical bag of candy (CA CCSS for Mathematics 7.SP.7, 8.SP.4).

6-8.AP.10

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
6–8	6-8.AP.10	Use flowcharts and/or pseudocode to design and illustrate algorithms that solve complex problems.	Algorithms & Programming	Algorithms	Abstraction	4.4, 4.1

Descriptive Statement

Complex problems are problems that would be difficult for students to solve without breaking them down into multiple steps. Flowcharts and pseudocode are used to design and illustrate the breakdown of steps in an algorithm. Students design and illustrate algorithms using pseudocode and/or flowcharts that organize and sequence the breakdown of steps for solving complex problems.

For example, students might use a flowchart to illustrate an algorithm that produces a recommendation for purchasing sneakers based on inputs such as size, colors, brand, comfort, and cost.

Alternatively, students could write pseudocode to express an algorithm for suggesting their outfit for the day, based on inputs such as the weather, color preferences, and day of the week.

6-8.AP.11

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
6–8	6-8.AP.11	Create clearly named variables that store data, and perform operations on their contents.	Algorithms & Programming	Variables	Creating	5.1, 5.2

Descriptive Statement

A variable is a container for data, and the name used for accessing the variable is called the identifier. Students declare, initialize, and update variables for storing different types of program data (e.g., text, integers) using names and naming conventions (e.g. camel case) that clearly convey the purpose of the variable, facilitate debugging, and improve readability.

For example, students could program a quiz game with a score variable (e.g. quizScore) that is initially set to zero and increases by increments of one each time the user answers a quiz question correctly and decreases by increments of one each time a user answers a quiz question incorrectly, resulting in a score that is either a positive or negative integer (CA CCSS for Mathematics 6.NS.5).

Alternatively, students could write a program that prompts the user for their name, stores the user's response in a variable (e.g. userName), and uses this variable to greet the user by name.

6-8.AP.12

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
6–8	6-8.AP.12	Design and iteratively develop programs that combine control structures and use compound conditions.	Algorithms & Programming	Control	Creating	5.1, 5.2

Descriptive Statement

Control structures can be combined in many ways. Nested loops are loops placed within loops, and nested conditionals allow the result of one conditional to lead to another. Compound conditions combine two or more conditions in a logical relationship (e.g., using AND, OR, and NOT). Students appropriately use control structures to perform repetitive and selection tasks.

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 (CA CCSS for ELA/Literacy W.6.3, W.7.3, W.8.3).

Alternatively, students could use compound conditionals when writing a program to test whether two points lie along the line defined by a particular linear function (CA CCSS for Mathematics 8.EE.7).

Additionally, students could use nested loops to program a character to do the “chicken dance” by opening and closing the beak, flapping the wings, shaking the hips, and clapping four times each; this dance “chorus” is then repeated several times in its entirety.

6-8.AP.13

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
6–8	6-8.AP.13	Decompose problems and subproblems into parts to facilitate the design, implementation, and review of programs.	Algorithms & Programming	Modularity	Computational Problems	3.2

Descriptive Statement

Decomposition facilitates program development by allowing students to focus on one piece at a time (e.g., getting input from the user, processing the data, and displaying the result to the user). Decomposition also enables different students to work on different parts at the same time. Students break down (decompose) problems into subproblems, which can be further broken down to smaller parts.

Students could create an arcade game, with a title screen, a game screen, and a win/lose screen with an option to play the game again. To do this, students need to identify subproblems that accompany each screen (e.g., selecting an avatar goes in the title screen, events for controlling character action and scoring goes in the game screen, and displaying final and high score and asking whether to play again goes in the win/lose screen).

Alternatively, students could decompose the problem of calculating and displaying class grades. Subproblems might include: accept input for students grades on various assignments, check for invalid grade entries, calculate per assignment averages, calculate per student averages, and display histograms of student scores for each assignment (CA CCSS for Mathematics 6.RP.3c, 6.SP.4, 6.SP.5).

6-8.AP.14

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
6–8	6-8.AP.14	Create procedures with parameters to organize code and make it easier to reuse.	Algorithms & Programming	Modularity	Abstraction	4.1, 4.3

Descriptive Statement

Procedures support modularity in developing programs. Parameters can provide greater flexibility, reusability, and efficient use of resources. Students create procedures and/or functions that are used multiple times within a program to repeat groups of instructions. They generalize the procedures and/or functions by defining parameters that generate different outputs for a wide range of inputs.

For example, students could create a procedure to draw a circle which involves many instructions, but all of them can be invoked with one instruction, such as "drawCircle." By adding a radius parameter, students can easily draw circles of different sizes (CA CCSS for Mathematics 7.G.4).

Alternatively, calculating the area of a regular polygon requires multiple steps. Students could write a function that accepts the number and length of the sides as parameters and then calculates the area of the polygon. This function can then be re-used inside any program to calculate the area of a regular polygon (CA CCSS for Mathematics 6.G.1).

6-8.AP.15

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
6–8	6-8.AP.15	Seek and incorporate feedback from team members and users to refine a solution that meets user needs.	Algorithms & Programming	Program Development	Collaborating, Inclusion	2.3, 1.1

Descriptive Statement

Development teams that employ user-centered design processes create solutions (e.g., programs and devices) that can have a large societal impact (e.g., an app that allows people with speech difficulties to allow a smartphone to clarify their speech). Students 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, or color contrast.

For example, if students are designing an app to teach their classmates about recycling, they could first interview or survey their classmates to learn what their classmates already know about recycling and why they do or do not recycle. After building a prototype of the app, the students could then test the app with a sample of their classmates to see if they learned anything from the app and if they had difficulty using the app (e.g., trouble reading or understanding text). After gathering interview data, students could refine the app to meet classmate needs (CA NGSS: MS-ETS1-4).

6-8.AP.16

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
6–8	6-8.AP.16	Incorporate existing code, media, and libraries into original programs, and give attribution.	Algorithms & Programming	Program Development	Abstraction, Creating, Communicating	4.2, 5.2, 7.3

Descriptive Statement

Building on the work of others enables students to produce more interesting and powerful creations. Students use portions of code, algorithms, digital media, and/or data created by others in their own programs and websites. They give attribution to the original creators to acknowledge their contributions.

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.

Alternatively, when creating a website to demonstrate their knowledge of historical figures from the Civil War, students may use a professionally-designed template and public domain images of historical figures (HSS.8.10.5).

Additionally, students could import libraries and connect to web application program interfaces (APIs) to make their own programming processes more efficient and reduce the number of bugs (e.g., to check whether the user input is a valid date, to input the current temperature from another city).

6-8.AP.17

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
6–8	6-8.AP.17	Systematically test and refine programs using a range of test cases.	Algorithms & Programming	Program Development	Testing	6.1

Descriptive Statement

Use cases and test cases are created to evaluate whether programs function as intended. At this level, students develop use cases and test cases with teacher guidance. Testing should become a deliberate process that is more iterative, systematic, and proactive than at lower levels.

For example, students 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).

Alternatively, in an interactive program, students could test that the character cannot move off of the screen in any direction, cannot move through walls, and can interact with other characters. They then adjust character behavior as needed.

6-8.AP.18

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
6–8	6-8.AP.18	Distribute tasks and maintain a project timeline when collaboratively developing computational artifacts.	Algorithms & Programming	Program Development	Collaborating, Creating	2.2, 5.1

Descriptive Statement

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 assume pre-defined roles within their teams and manage the project workflow using structured timelines. With teacher guidance, they begin to create collective goals, expectations, and equitable workloads.

For example, students could decompose 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.

Alternatively, students could work as a team to develop a storyboard for an animation representing a written narrative, and then program the scenes individually (CA CCSS for ELA/Literacy W.6.3, W.7.3, W.8.3).

6-8.AP.19

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
6–8	6-8.AP.19	Document programs in order to make them easier to use, read, test, and debug.	Algorithms & Programming	Program Development	Communicating	7.2

Descriptive Statement

Documentation allows creators, end users, and other developers to more easily use and understand a program. Students provide documentation for end users that explains their artifacts and how they function (e.g., project overview, user instructions). They also include comments within code to describe portions of their programs and make it easier for themselves and other developers to use, read, test, and debug.

For example, students could add comments to describe functionality of different segments of code (e.g., input scores between 0 and 100, check for invalid input, calculate and display the average of the scores). They could also communicate the process used by writing design documents, creating flowcharts, or making presentations (CA CCSS for ELA/Literacy SL.6.5, SL.7.5, SL.8.5).

6-8.IC.20

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
6–8	6-8.IC.20	Compare tradeoffs associated with computing technologies that affect people's everyday activities and career options.	Impacts of Computing	Culture	Communicating	7.2

Descriptive Statement

Advancements in computer technology are neither wholly positive nor negative. However, the ways that people use computing technologies have tradeoffs. Students consider current events related to broad ideas, including privacy, communication, and automation.

For example, students could compare and contrast the impacts of computing technologies with the impacts of other systems developed throughout history such as the Pony Express and US Postal System (HSS.7.8.4).

Alternatively, students could identify tradeoffs for both personal and professional uses of a variety of computing technologies. For instance, driverless cars can increase convenience and reduce accidents, but they may be susceptible to hacking. The emerging industry will reduce the number of taxi and shared-ride drivers, but may create more software engineering and cybersecurity jobs.

6-8.IC.21

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
6–8	6-8.IC.21	Discuss issues of bias and accessibility in the design of existing technologies.	Impacts of Computing	Culture	Inclusion	1.2

Descriptive Statement

Computing technologies should support users of many backgrounds and abilities. In order to maximize accessibility, these differences need to be addressed by examining diverse populations. With the teacher's guidance, students test and discuss the usability of various technology tools, such as apps, games, and devices.

For example, students could discuss the impacts of facial recognition software that works better for lighter skin tones and recognize that the software was likely developed with a homogeneous testing group. Students could then discuss how accessibility could be improved by sampling a more diverse population (CA CCSS for ELA/Literacy SL.6.1, SL.7.1, SL.8.1).

6-8.IC.22

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
6–8	6-8.IC.22	Collaborate with many contributors when creating a computational artifact.	Impacts of Computing	Social Interactions	Collaborating, Creating	2.4, 5.2

Descriptive Statement

Users have diverse sets of experiences, needs, and wants. These need to be understood and integrated into the design of computational artifacts. Students use applications that enable crowdsourcing to gather services, ideas, or content from a large group of people. At this level, crowdsourcing can be done at the local level (e.g., classroom, school, or neighborhood) and/or global level (e.g., age-appropriate online communities).

For example, a group of students could use electronic surveys to solicit input from their neighborhood regarding an important social or political issue. They could collaborate with a community artist to combine animations and create a digital community collage informing the public about various points of view regarding the topic (VAPA Visual Art 8.5.2, 8.5.4).

6-8.IC.23

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
6–8	6-8.IC.23	Compare tradeoffs associated with licenses for computational artifacts to balance the protection of the creators' rights and the ability for others to use and modify the artifacts.	Impacts of Computing	Safety Law & Ethics	Communicating	7.3

Descriptive Statement

Using and building on the works of others allows people to create meaningful works and fosters innovation. Copyright is an important law that helps protect the rights of creators so they receive credit and get paid for their work. Creative Commons is a kind of copyright that makes it easier for people to copy, share, and build on creative work, as long as they give credit for it. There are different kinds of Creative Commons licenses that allow people to do things such as change, remix, or make money from their work. As creators, students can pick and choose how they want their work to be used, and then create a Creative Commons license that they include in their work.

For example, students could create interactive animations to educate others on bullying or protecting the environment. They then select an appropriate license to reflect how they want their program to be used by others (e.g., allow others to use their work and alter it, as long as they do not make a profit from it). Students use established methods to both protect their artifacts and attribute use of protected artifacts.

6-8.IC.24

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
6–8	6-8.IC.24	Compare tradeoffs between allowing information to be public and keeping information private and secure.	Impacts of Computing	Safety Law & Ethics	Communicating	7.2

Descriptive Statement

While it is valuable to establish, maintain, and strengthen connections between people online, security attacks often start with intentionally or unintentionally providing personal information online. Students identify situations where the value of keeping information public outweighs privacy concerns, and vice versa. They also recognize practices such as phishing and social engineering and explain best practices to defend against them.

For example, students could discuss the benefits of artists and designers displaying their work online to reach a broader audience. Students could also compare the tradeoffs of making a shared file accessible to anyone versus restricting it to specific accounts (CA CCSS for ELA/Literacy SL.6.1, SL.7.1, SL.8.1).

Alternatively, students could discuss the benefits and dangers of the increased accessibility of information available on the internet, and then compare this to the advantages and disadvantages of the introduction of the printing press in society (HSS.7.8.4).

9–12

9-12.CS.1

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.CS.1	Describe ways in which abstractions hide the underlying implementation details of computing systems to simplify user experiences.	Computing Systems	Devices	Abstraction	4.1

Descriptive Statement

An abstraction is a representation of an idea or phenomenon that hides details irrelevant to the question at hand. Computing systems, both stand alone and embedded in products, are often integrated with other systems to simplify user experiences.

For example, students could identify geolocation hardware embedded in a smartphone and describe how this simplifies the users experience since the user does not have to enter her own location on the phone.

Alternatively, students might select an embedded device such as a car stereo, identify the types of data (e.g., radio station presets, volume level) and procedures (e.g., increase volume, store/recall saved station, mute) it includes, and explain how the implementation details are hidden from the user.

9-12.CS.2

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.CS.2	Compare levels of abstraction and interactions between application software, system software, and hardware.	Computing Systems	Hardware & Software	Abstraction	4.1

Descriptive Statement

At its most basic level, a computer is composed of physical hardware on which software runs. Multiple layers of software are built upon various layers of hardware. Layers manage interactions and complexity in the computing system. System software manages a computing device's resources so that software can interact with hardware. Application software communicates with the user and the system software to accomplish its purpose. Students compare and describe how application software, system software, and hardware interact.

For example, students could compare how various levels of hardware and software interact when a picture is to be taken on a smartphone. Systems software provides low-level commands to operate the camera hardware, but the application software interacts with system software at a higher level by requesting a common image file format (e.g., .png) that the system software provides.

9-12.CS.3

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.CS.3	Develop guidelines that convey systematic troubleshooting strategies that others can use to identify and fix errors.	Computing Systems	Troubleshooting	Testing	6.2

Descriptive Statement

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 and adapt solutions that have worked in the past.

For example, students could create a list of troubleshooting strategies to debug network connectivity problems such as checking hardware and software status and settings, rebooting devices, and checking security settings.

Alternatively, students could create troubleshooting guidelines for help desk employees based on commonly observed problems (e.g., problems connecting a new device to the computer, problems printing from a computer to a network printer).

9-12.NI.4

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.NI.4	Describe issues that impact network functionality.	Networks & the Internet	Network Communication & Organization	Abstraction	4.1

Descriptive Statement

Many different organizations, including educational, governmental, private businesses, and private households rely on networks to function adequately in order to engage in online commerce and activity. Quality of Service (QoS) refers to the capability of a network to provide better service to selected network traffic over various technologies from the perspective of the consumer. Students define and discuss performance measures that impact network functionality, such as latency, bandwidth, throughput, jitter, and error rate.

For example, students could use online network simulators to explore how performance measures impact network functionality and describe impacts when various changes in the network occur.

Alternatively, students could describe how pauses in television interviews conducted over satellite telephones are impacted by networking factors such as latency and jitter.

9-12.NI.5

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.NI.5	Describe the design characteristics of the internet.	Networks & the Internet	Network Communication & Organization	Communicating	7.2

Descriptive Statement

The internet connects devices and networks all over the world. Large-scale coordination occurs among many different machines across multiple paths every time a web page is opened or an image is viewed online. Through the domain name system (DNS), devices on the internet can look up Internet Protocol (IP) addresses, allowing end-to-end communication between devices. The design decisions that direct the coordination among systems composing the internet also allow for scalability and reliability. Students factor historical, cultural, and economic decisions in their explanations of the internet.

For example, students could explain how hierarchy in the DNS supports scalability and reliability.

Alternatively, students could describe how the redundancy of routing between two nodes on the internet increases reliability and scales as the internet grows.

9-12.NI.6

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.NI.6	Compare and contrast security measures to address various security threats.	Networks & the Internet	Cybersecurity	Communication	7.2

Descriptive Statement

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 compare and contrast different types of security measures based on factors such as efficiency, feasibility, ethical impacts, usability, and security. At this level, students are not expected to develop or implement the security measures that they discuss.

For example, students could review case studies or current events in which governments or organizations experienced data leaks or data loss as a result of these types of attacks. Students could provide an analysis of actual security measures taken comparing to other security measure which may have led to different outcomes.

Alternatively, 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.

9-12.NI.7

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.NI.7	Compare and contrast cryptographic techniques to model the secure transmission of information.	Networks & the Internet	Cybersecurity	Computational Problems, Abstraction	3.3, 4.4

Descriptive Statement

Cryptography is a technique for transforming information on a computer in such a way that it becomes unreadable by anyone except authorized parties. Cryptography is useful for supporting secure communication of data across networks. Examples of cryptographic methods include hashing, symmetric encryption/decryption (private key), and asymmetric encryption/decryption (public key/private key). Students use software to encode and decode messages using cryptographic methods. Students compare the costs and benefits of using various cryptographic methods. At this level, students are not expected to perform the mathematical calculations associated with encryption and decryption.

For example, students could compare and contrast multiple examples of symmetric cryptographic techniques.

Alternatively, students could compare and contrast symmetric and asymmetric cryptographic techniques in which they apply for a given scenario.

9-12.DA.8

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.DA.8	Translate between different representations of data abstractions of real-world phenomena, such as characters, numbers, and images.	Data & Analysis	Storage	Abstraction	4.1

Descriptive Statement

Computers represent complex real-world concepts such as characters, numbers, and images through various abstractions. Students translate between these different levels of data representations.

For example, students could convert an HTML (Hyper Text Markup Language) tag for red font into RGB (Red Green Blue), HEX (Hexadecimal Color Code), HSL (Hue Saturation Lightness), RGBA (Red Green Blue Alpha), or HSLA (Hue Saturation Lightness and Alpha) representations.

Alternatively, students could convert the standard representation of a character such as ! into ASCII or Unicode.

9-12.DA.9

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.DA.9	Describe tradeoffs associated with how data elements are organized and stored.	Data & Analysis	Storage	Computational Problems	3.3

Descriptive Statement

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 describe implications for a given data organization or storage choice in light of a specific problem.

For example, students might consider the cost, speed, reliability, accessibility, privacy, and integrity tradeoffs between storing photo data on a mobile device versus in the cloud.

Alternatively, students might compare the tradeoffs between file size and image quality of various image file formats and how choice of format may be influenced by the device on which it is to be accessed (e.g., smartphone, computer).

9-12.DA.10

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.DA.10	Create data visualizations to help others better understand real-world phenomena.	Data & Analysis	Collection Visualization & Transformation	Communicating	5.2

Descriptive Statement

People transform, generalize, simplify, and present large data sets in different ways to influence how other people interpret and understand the underlying information. Students select relevant data from large or complex data sets in support of a claim or to communicate the information in a more sophisticated manner. Students use software tools or programming to perform a range of mathematical operations to transform and analyze data and create powerful data visualizations (that reveal patterns in the data).

For example, students could create data visualizations to reveal patterns in voting data by state, gender, political affiliation, or socioeconomic status.

Alternatively, students could use U.S. government data on critically endangered animals to visualize population change over time.

9-12.DA.11

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.DA.11	Refine computational models to better represent the relationships among different elements of data collected from a phenomenon or process.	Data & Analysis	Inference & Models	Abstraction, Testing	4.4, 6.3

Descriptive Statement

Computational models are used to make predictions about processes or phenomena based on selected data and features. They allow people to investigate the relationships among different variables to understand a system. Predictions are tested to validate models. Students evaluate these models against real-world observations.

For example, students could use a population model that allows them to speculate about interactions among different species, evaluate the model based on data gathered from nature, and then refine the model to reflect more complex and realistic interactions.

9-12.AP.12

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.AP.12	Design algorithms to solve computational problems using a combination of original and existing algorithms.	Algorithms & Programming	Algorithms	Creating, Abstraction	5.1, 4.2

Descriptive Statement

Knowledge of common algorithms improves how people develop software, secure data, and store information. Some algorithms may be easier to implement in a particular programming language, work faster, require less memory to store data, and be applicable in a wider variety of situations than other algorithms. Algorithms used to search and sort data are common in a variety of software applications.

For example, students could design an algorithm to calculate and display various sports statistics and use common sorting or mathematical algorithms (e.g., average) in the design of the overall algorithm.

Alternatively, students could design an algorithm to implement a game and use existing randomization algorithms to place pieces randomly in starting positions or to control the “roll” of a dice or selection of a “card” from a deck.

9-12.AP.13

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.AP.13	Create more generalized computational solutions using collections instead of repeatedly using simple variables.	Algorithms & Programming	Variables	Abstraction	4.1

Descriptive Statement

Computers can automate repetitive tasks with algorithms that use collections to simplify and generalize computational problems. Students identify common features in multiple segments of code and substitute a single segment that uses collections (i.e., arrays, sets, lists) to account for the differences.

For example, students could take a program that inputs students' scores into multiple variables and modify it to read these scores into a single array of scores.

Alternatively, instead of writing one procedure to find averages of student scores and another to find averages of student absences, students could write a single general average procedure to support both tasks.

9-12.AP.14

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.AP.14	Justify the selection of specific control structures by identifying tradeoffs associated with implementation, readability, and performance.	Algorithms & Programming	Control	Creating	5.2

Descriptive Statement

The selection of control structures in a given programming language impacts readability and performance. Readability refers to how clear the program is to other programmers and can be improved through documentation. Control structures at this level may include, for example, conditional statements, loops, event handlers, and recursion. Students justify control structure selection and tradeoffs in the process of creating their own computational artifacts. The discussion of performance is limited to a theoretical understanding of execution time and storage requirements; a quantitative analysis is not expected.

For example, students could compare the readability and program performance of iterative and recursive implementations of procedures that calculate the Fibonacci sequence.

Alternatively, students could compare the readability and performance tradeoffs of multiple if statements versus a nested if statement.

9-12.AP.15

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.AP.15	Iteratively design and develop computational artifacts for practical intent, personal expression, or to address a societal issue by using events to initiate instructions.	Algorithms & Programming	Control	Creating	5.1, 5.2, 5.3

Descriptive Statement

In this context, relevant computational artifacts can 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.

For example, students might create a tool for drawing on a canvas by first implementing a button to set the color of the pen.

Alternatively, students might create a game where many events control instructions executed (e.g., when a score climbs above a threshold, a congratulatory sound is played; when a user clicks on an object, the object is loaded into a basket; when a user clicks on an arrow key, the player object is moved around the screen).

9-12.AP.16

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.AP.16	Decompose problems into smaller subproblems through systematic analysis, using constructs such as procedures, modules, and/or classes.	Algorithms & Programming	Control	Abstraction	3.2

Descriptive Statement

Decomposition enables solutions to complex problems to be designed and implemented as more manageable subproblems. Students decompose a given problem into subproblems that can be solved using existing functionalities, or new functionalities that they design and implement.

For example, students could design a program for supporting soccer coaches in analyzing their teams' statistics. They decompose the problem in terms of managing input, analysis, and output. They decompose the data organization by designing what data will be stored per player, per game, and per team. Team players may be stored as a collection. Data per team player may include: number of shots, misses, saves, assists, penalty kicks, blocks, and corner kicks. Students design methods for supporting various statistical analyses and display options. Students design output formats for individual players or coaches.

9-12.AP.17

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.AP.17	Create computational artifacts using modular design.	Algorithms & Programming	Modularity	Abstraction, Creating	4.3, 5.2

Descriptive Statement

Computational artifacts are created by combining and modifying existing computational artifacts and/or by developing new artifacts. To reduce complexity, large programs can be designed as systems of interacting modules, each with a specific role, coordinating for a common overall purpose. Students should create computational artifacts with interacting procedures, modules, and/or libraries.

For example, students could incorporate a physics library into an animation of bouncing balls.

Alternatively, students could integrate open-source JavaScript libraries to expand the functionality of a web application.

Additionally, students could create their own game to teach Spanish vocabulary words using their own modular design (e.g., including methods to: control scoring, manage wordlists, manage access to different game levels, take input from the user, etc.).

9-12.AP.18

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.AP.18	Systematically design programs for broad audiences by incorporating feedback from users.	Algorithms & Programming	Program Development	Inclusion, Creating	1.1, 5.1

Descriptive Statement

Programmers use a systematic design and review process to meet the needs of a broad audience. The process includes planning to meet user needs, developing software for broad audiences, testing users from a cross-section of the audience, and refining designs based on feedback.

For example, students could create a user satisfaction survey and brainstorm distribution methods to collect feedback about a mobile application. After collecting feedback from a diverse audience, students could incorporate feedback into their product design.

Alternatively, while developing an e-textiles project with human touch sensors, students could collect data from peers and identify design changes needed to improve usability by users of different needs.

9-12.AP.19

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.AP.19	Explain the limitations of licenses that restrict use of computational artifacts when using resources such as libraries.	Algorithms & Programming	Program Development	Communicating	7.3

Descriptive Statement

Software licenses include copyright, freeware, and open-source licensing schemes. Licenses are used to protect the intellectual property of the author while also defining accessibility of the code. Students consider licensing implications for their own work, especially when incorporating libraries and other resources.

For example, students might consider two software libraries that address a similar need, justifying their choice of one over the other. The choice could be based upon least restrictive licensing or further protections for their own intellectual property.

9-12.AP.20

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.AP.20	Iteratively evaluate and refine a computational artifact to enhance its performance, reliability, usability, and accessibility.	Algorithms & Programming	Program Development	Testing	6.3

Descriptive Statement

Evaluation and refinement of computational artifacts involves measuring, testing, debugging, and responding to the changing needs and expectations of users. Aspects that can be evaluated include correctness, performance, reliability, usability, and accessibility.

For example, after witnessing common errors with user input in a computational artifact, students could refine the artifact to validate user input and provide an error message if invalid data is provided.

Alternatively, students could observe a robot in a variety of lighting conditions to determine whether the code controlling a light sensor should be modified to make it less sensitive.

Additionally, students could also incorporate feedback from a variety of end users to help guide the size and placement of menus and buttons in a user interface.

9-12.AP.21

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.AP.21	Design and develop computational artifacts working in team roles using collaborative tools.	Algorithms & Programming	Program Development	Collaborating	2.4

Descriptive Statement

Collaborative tools can be as complex as a source code version control system or as simple as a collaborative word processor. Team roles in pair programming are driver and navigator but students can take on more specialized roles in larger teams. Teachers or students should choose resources that aid collaborative program development as programs grow more complex.

For example, students might work as a team to develop a mobile application that addresses a problem relevant to the school or community, using appropriate tools to support actions such as: establish and manage the project timeline; design, share, and revise graphical user interface elements; implement program components, track planned, in-progress, and completed components, and design and implement user testing.

9-12.AP.22

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.AP.22	Document decisions made during the design process using text, graphics, presentations, and/or demonstrations in the development of complex programs.	Algorithms & Programming	Program Development	Communicating	7.2

Descriptive Statement

Complex programs are often iteratively designed as systems of interacting modules, each with a specific role, coordinating for a common overall purpose. Comments are included in code both to document the purpose of modules as well as the implementation details within a module. Together these support documentation of the design process. Students use resources such as libraries and tools to edit and manage parts of the program and corresponding documentation.

For example, during development of a computational artifact students could comment their code (with date, modification, and rationale), sketch a flowchart to summarize control flow in a code journal, and share ideas and updates on a white board. Students may document their logic by explaining the development process and presenting to the class. The presentation could include photos of their white board, a video or screencast explaining the development process, or recorded audio description.

9-12.IC.23

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.IC.23	Evaluate the ways computing impacts personal, ethical, social, economic, and cultural practices.	Impacts of Computing	Culture	Computational Problems, Inclusion	3.1, 1.2

Descriptive Statement

Computing may improve, harm, or maintain practices. An understanding of how equity deficits, such as minimal exposure to computing, access to education, and training opportunities, are related to larger, systemic problems in society enables students to create more meaningful artifacts. Students illustrate the positive, negative, and/or neutral impacts of computing.

For example, students could evaluate the accessibility of a product for a broad group of end users, such as people who lack access to broadband or who have various disabilities. Students could identify potential bias during the design process and evaluate approaches to maximize accessibility in product design.

Alternatively, students could evaluate the impact of social media on cultural, economic, and social practices around the world.

9-12.IC.24

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.IC.24	Identify impacts of bias and equity deficit on design and implementation of computational artifacts and apply appropriate processes for evaluating issues of bias.	Impacts of Computing	Culture	Inclusion	1.2

Descriptive Statement

Biases could include incorrect assumptions developers have made about their users, including minimal exposure to computing, access to education, and training opportunities. Students identify and use strategies to test and refine computational artifacts with the goal of reducing bias and equity deficits and increasing universal access.

For example, students could use a spreadsheet to chart various forms of equity deficits, and identify solutions in existing software. Students could use and refine the spreadsheet solutions to create a strategy for methodically testing software specifically for bias and equity.

9-12.IC.25

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.IC.25	Demonstrate ways a given algorithm applies to problems across disciplines.	Impacts of Computing	Culture	Computational Problems	3.1

Descriptive Statement

Students identify how a given algorithm can be applied to real-world problems in different disciplines.

For example, students could demonstrate how a randomization algorithm can be used to select participants for a clinical medical trial or to select a flash card to display on a vocabulary quiz.

Alternatively, students could demonstrate how searching and sorting algorithms are needed to organize records in manufacturing settings, or to support doctors queries of patient records, or to help governments manage support services they provide to their citizens.

9-12.IC.26

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.IC.26	Study, discuss, and think critically about the potential impacts and implications of emerging technologies on larger social, economic, and political structures, with evidence from credible sources.	Impacts of Computing	Culture	Communicating	7.2

Descriptive Statement

For example, after studying the rise of artificial intelligence, students create a cause and effect chart to represent positive and negative impacts of this technology on society.

9-12.IC.27

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.IC.27	Use collaboration tools and methods to increase connectivity with people of different cultures and careers.	Impacts of Computing	Social Interactions	Collaborating	2.4

Descriptive Statement

Increased digital connectivity and communication between people across a variety of cultures and in differing professions has changed the collaborative nature of personal and professional interaction. Students identify, explain, and use appropriate collaborative tools.

For example, students could compare ways that various technological collaboration tools could help a team become more cohesive and then choose one of these tools to manage their teamwork.

Alternatively, students could use different collaborative tools and methods to solicit input from not only team members and classmates but also others, such as participants in online forums or local communities.

9-12.IC.28

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.IC.28	Explain the beneficial and harmful effects that intellectual property laws can have on innovation.	Impacts of Computing	Safety Law & Ethics	Communicating	7.3

Descriptive Statement

Laws and ethics govern aspects of computing such as privacy, data, property, information, and identity. Students explain the beneficial and harmful effects of intellectual property laws as they relate to potential innovations and governance.

For example, students could explain how patents protect inventions but may limit innovation.

Alternatively, students could explain how intellectual property laws requiring that artists be paid for use of their media might limit the choice of songs developers can use in their computational artifacts.

9-12.IC.29

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.IC.29	Explain the privacy concerns related to the collection and generation of data through automated processes.	Impacts of Computing	Safety Law & Ethics	Communicating	7.2

Descriptive Statement

Data can be collected and aggregated across millions of people, even when they are not actively engaging with or physically near the data collection devices. Students recognize automated and non-evident collection of information and the privacy concerns they raise for individuals.

For example, students could explain the impact on an individual when a social media site's security settings allows for mining of account information even when the user is not online.

Alternatively, students could discuss the impact on individuals of using surveillance video in a store to track customers.

Additionally, students could discuss how road traffic can be monitored to change signals in real time to improve road efficiency without drivers being aware and discuss policies for retaining data that identifies drivers' cars and their behaviors.

9-12.IC.30

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12	9-12.IC.30	Evaluate the social and economic implications of privacy in the context of safety, law, or ethics.	Impacts of Computing	Safety Law & Ethics	Communicating	7.2

Descriptive Statement

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 make and justify claims about potential and/or actual privacy implications of policies, laws, or ethics and consider the associated tradeoffs, focusing on society and the economy.

For example, students could explore the case of companies tracking online shopping behaviors in order to decide which products to target to consumers. Students could evaluate the ethical and legal dilemmas of collecting such data without consumer knowledge in order to profit companies.

Alternatively, students could evaluate the implications of net neutrality laws on society's access to information and on the impacts to businesses of varying sizes.

9–12 Specialty

9-12S.CS.1

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.CS.1	Illustrate ways computing systems implement logic through hardware components.	Computing Systems	Devices	Communicating, Abstractions	7.2, 4.4

Descriptive Statement

Computing systems use processors (e.g., a central processing unit or CPU) to execute program instructions. Processors are composed of components that implement the logical or computational operations required by the instructions. AND, OR, and NOT are examples of logic gates. Adders are examples of higher-leveled circuits built using low-level logic gates. Students illustrate how modern computing devices are made up of smaller and simpler components which implement the logic underlying the functionality of a computer processor. At this level, knowledge of how logic gates are constructed is not expected.

For example, students could construct truth tables, draw logic circuit diagrams, or use an online logic circuit simulator. Students could explore the interaction of the CPU, RAM, and I/O by labeling a diagram of the von Neumann architecture.

Alternatively, students could design higher-level circuits using low-level logic gates (e.g., adders).

9-12S.CS.2

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.CS.2	Categorize and describe the different functions of operating system software.	Computing Systems	Hardware & Software	Communicating	7.2

Descriptive Statement

Operating systems (OS) software is the code that manages the computer's basic functions. Students describe at a high level the different functions of different components of operating system software. Examples of functions could include memory management, data storage/retrieval, processes management, and access control.

For example, students could use monitoring tools including within an OS to inspect the services and functions running on a system and create an artifact to describe the activity that they observed (e.g., when a browser is running with many tabs open, memory usage is increased). They could also inspect and describe changes in the activity monitor that occur as different applications are executing (e.g., processor utilization increases when a new application is launched).

9-12S.NI.3

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.NI.3	Examine the scalability and reliability of networks, by describing the relationship between routers, switches, servers, topology, and addressing.	Networks & the Internet	Network Communication & Organization	Abstractions	4.4

Descriptive Statement

Choice of network topology is determined, in part, by how many devices can be supported and the character of communication needs between devices. Each device is assigned an address that uniquely identifies it on the network. Routers function by comparing addresses to determine how information on the network should reach its designation. Switches compare addresses to determine which computers will receive information. Students explore and explain how network performance degrades when various factors affect the network.

For example, students could use online network simulators to describe how network performance changes when the number of devices increases.

Alternatively, students could visualize and describe changes to the distribution of network traffic when a router on the network fails.

9-12S.NI.4

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.NI.4	Explain how the characteristics of the internet influence the systems developed on it.	Networks & the Internet	Network Communication & Organization	Communicating	7.2

Descriptive Statement

The design of the internet includes hierarchy and redundancy to help it scale reliably. An end-to-end architecture means that key functions are placed at endpoints in the network (i.e., an internet user's computer and the server hosting a website) rather than in the middle of the network. Open standards for transmitting information across the internet help fuel its growth. This design philosophy impacts systems and technologies that integrate with the internet. Students explain how internet-based systems depend on these characteristics.

For example, students could explain how having common, standard protocols enable products and services from different developers to communicate.

Alternatively, students could describe how the end-to-end architecture and redundancy in routing enables internet users to access information and services even if part of the network is down; the information can still be routed from one end to another through a different path.

9-12S.NI.5

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.NI.5	Develop solutions to security threats.	Networks & the Internet	Cybersecurity	Creating	5.3

Descriptive Statement

Designing and implementing cybersecurity measures requires knowledge of software, hardware, and human components and understanding tradeoffs. Students design solutions to security threats and compare tradeoffs of easier access and use against the costs of losing information and disrupting services.

For example, students could refine a technology that allows users to use blank or weak passwords.

Alternatively, students could implement a firewall or proxy protection between an organization's private local area network (LAN) and the public internet.

Additionally, students could find and close exploitable threats on an infected computer in order to protect information.

9-12S.NI.6

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.NI.6	Analyze cryptographic techniques to model the secure transmission of information.	Networks & the Internet	Cybersecurity	Computational Problems, Abstractions	3.3, 4.2

Descriptive Statement

Cryptography is essential to many models of cybersecurity. Open standards help to ensure cryptographic security. Certificate Authorities (CAs) issue digital certificates that validate the ownership of encrypted keys used in secured communications across the internet. Students encode and decode messages using encryption and decryption methods, and they should understand the different levels of complexity to hide or secure information.

For example, students could analyze the relative designs of private key vs. public key encryption techniques and apply the best choice for a particular scenario.

Alternatively, students could analyze the design of the Diffie-Helman algorithm to RSA (Rivest–Shamir–Adleman) and apply the best choice for a particular scenario. They could provide a cost-benefit analysis of runtime and ease of cracking for various encryption techniques which are commonly used to secure transmission of data over the internet.

9-12S.DA.7

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.DA.7	Select and use data collection tools and techniques to generate data sets.	Data & Analysis	Collection Visualization & Transformation	Communicating	7.1

Descriptive Statement

Data collection and organization is essential for obtaining new information insights and revealing new knowledge in our modern world. As computers are able to process larger sets of data, gathering data in an efficient and reliable matter remains important. The choice of data collection tools and quality of the data collected influences how new information, insights, and knowledge will support claims and be communicated. Students devise a reliable method to gather information, use software to extract digital data from data sets, and clean and organize the data in ways that support summaries of information obtained from the data. At this level, students may, but are not required to, create their own data collection tools.

For example, students could create a computational artifact that records information from a sonic distance sensor to monitor the motion of a prototype vehicle.

Alternatively, students could develop a reliable and practical way to automatically digitally record the number of animals entering a portion of a field to graze.

Additionally, students could also find a web site containing data (e.g., race results for a major marathon), scrape the data from the web site using data collection tools, and format the data so it can be analyzed.

9-12S.DA.8

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.DA.8	Use data analysis tools and techniques to identify patterns in data representing complex systems.	Data & Analysis	Collection Visualization & Transformation	Communicating, Abstraction	7.1, 4.1

Descriptive Statement

Data analysis tools can be useful for identifying patterns in large amounts of data in many different fields. Computers can help with the processing of extremely large sets of data making very complex systems manageable. Students use computational tools to analyze, summarize, and visualize a large set of data.

For example, students could analyze a data set containing marathon times and determine how age, gender, weather, and course features correlate with running times.

Alternatively, students could analyze a data set of social media interactions to identify the most influential users and visualize the intersections between different social groups.

9-12S.DA.9

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.DA.9	Evaluate the ability of models and simulations to test and support the refinement of hypotheses.	Data & Analysis	Inference & Models	Abstraction	4.4

Descriptive Statement

A model could be implemented as a diagram or a program that represents key properties of a physical or other system. A simulation is based on a model, and enables observation of the system as key properties change. Students explore, explain, and evaluate existing models and simulations, in order to support the refinement of hypotheses about how the systems work. At this level, the ability to accurately and completely model and simulate complex systems is not expected.

For example, a computer model of ants following a path created by other ants who found food explains the trail-like travel patterns of the insect. Students could evaluate if the output of the model fits well with their hypothesis that ants navigate the world through the use of pheromones. They could explain how the computer model supports this hypothesis and how it might leave out certain aspects of ant behavior and whether these are important to understanding ant travel behavior.

Alternatively, students could hypothesize how different ground characteristics (e.g., soil type, thickness of sediment above bedrock) relate to the severity of shaking at the surface during an earthquake. They could add or modify input about ground characteristics into an earthquake simulator, observe the changed simulation output, and then evaluate their hypotheses.

9-12S.AP.10

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.AP.10	Describe how artificial intelligence drives many software and physical systems.	Algorithms & Programming	Algorithms	Communicating, Computational Problems	7.2, 3.1

Descriptive Statement

Artificial intelligence is a sub-discipline of computer science that enables computers to solve problems previously handled by biological systems. There are many applications of artificial intelligence, including computer vision and speech recognition. Students research and explain how artificial intelligence has been employed in a given system. Students are not expected to implement an artificially intelligent system in order to meet this standard.

For example, students could observe an artificially intelligent system and notice where its behavior is not human-like, such as when a character in a videogame makes a mistake that a human is unlikely to make, or when a computer easily beats even the best human players at a given game.

Alternatively, students could interact with a search engine asking various questions, and after reading articles on the topic, they could explain how the computer is able to respond to queries.

9-12S.AP.11

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.AP.11	Implement an algorithm that uses artificial intelligence to overcome a simple challenge.	Algorithms & Programming	Algorithms	Creating, Computational Problems	5.3, 3.1

Descriptive Statement

Artificial intelligence algorithms allow a computer to perceive and move in the world, use knowledge, and engage in problem solving. Students create a computational artifact that is able to carry out a simple task commonly performed by living organisms. Students do not need to realistically simulate human behavior or solve a complex problem in order to meet this standard.

For example, students could implement an algorithm for playing tic-tac-toe that would select an appropriate location for the next move.

Alternatively, students could implement an algorithm that allows a solar-powered robot to move to a sunny location when its batteries are low.

9-12S.AP.12

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.AP.12	Implement searching and sorting algorithms to solve computational problems.	Algorithms & Programming	Algorithms	Abstraction, Creating	4.2, 5.2

Descriptive Statement

One of the core uses of computers is to store, organize, and retrieve information when working with large amounts of data. Students create computational artifacts that use searching and/or sorting algorithms to retrieve, organize, or store information. Students do not need to select their algorithm based on efficiency.

For example, students could write a script to sequence their classmates in order from youngest to oldest.

Alternatively, students could write a program to find certain words within a text and report their location.

9-12S.AP.13

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.AP.13	Evaluate algorithms in terms of their efficiency.	Algorithms & Programming	Algorithms	Abstraction	3.3

Descriptive Statement

Algorithms that perform the same task can be implemented in different ways, which take different amounts of time to run on a given input set. Algorithms are commonly evaluated using asymptotic analysis (i.e., "Big O") which involves exploration of behavior when the input set grows very large. Students classify algorithms by the most common time classes (e.g., log n, linear, n log n, and quadratic or higher).

For example, students could read a given algorithm, identify the control constructs, and in conjunction with input size, identify the efficiency class of the algorithm.

9-12S.AP.14

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.AP.14	Compare and contrast fundamental data structures and their uses.	Algorithms & Programming	Variables	Abstraction	4.2

Descriptive Statement

Data structures are designed to provide different ways of storing and manipulating data sets to optimize various aspects of storage or runtime performance. Choice of data structures is made based on expected data characteristics and expected program functions. Students = compare and contrast how basic functions (e.g.., insertion, deletion, and modification) would differ for common data structures including lists, arrays, stacks, and queues.

For example, students could draw a diagram of how different data structures change when items are added, deleted, or modified. They could explain tradeoffs in storage and efficiency issues.

Alternatively, when presented with a description of a program and the functions it would be most likely to be running, students could list pros and cons for a specific data structure use in that scenario.

9-12S.AP.15

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.AP.15	Demonstrate the flow of execution of a recursive algorithm.	Algorithms & Programming	Control	Computational Problems, Communicating	3.2, 7.2

Descriptive Statement

Recursion is a powerful problem-solving approach where the problem solution is built on solutions of smaller instances of the same problem. A base case, which returns a result without referencing itself, must be defined, otherwise infinite recursion will occur. Students represent a sequence of calls to a recursive algorithm and show how the process resolves to a solution.

For example, students could draw a diagram to illustrate flow of execution by keeping track of parameter and returned values for each recursive call.

Alternatively, students could create a video showing the passing of arguments as the recursive algorithm runs.

9-12S.AP.16

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.AP.16	Analyze a large-scale computational problem and identify generalizable patterns or problem components that can be applied to a solution.	Algorithms & Programming	Modularity	Computational Problems, Abstraction	3.2, 4.2

Descriptive Statement

As students encounter complex, real-world problems that span multiple disciplines or social systems, they need to be able to decompose problems and apply already developed code as part of their solutions. Students decompose complex problems into manageable subproblems that could potentially be solved with programs or procedures that can be reused or already exist.

For example, in analyzing an internet radio app, students could identify that users need to create an account and enter a password. They could identify a common application programming interface (API) for checking and displaying password strength. Additionally, students could recognize that the songs would need to be sorted by the time last played in order to display the most recently played songs and identify a common API for sorting dates from most to least recent.

Alternatively, in analyzing the problem of tracking medical treatment in a hospital, students could recognize that patient records need to be stored in a database and identify a database solution to support quick access and modification of patient records. Additionally, they could recognize that records in the database need to be stored securely and could identify an encryption API to support the desired level of privacy.

9-12S.AP.17

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.AP.17	Construct solutions to problems using student-created components, such as procedures, modules, and/or objects.	Algorithms & Programming	Modularity	Abstraction, Creating	4.3, 5.2

Descriptive Statement

Programmers often address complex tasks through design and decomposition using procedures and/or modules. In object-oriented programming languages, classes can support this decomposition. Students create a computational artifact that solves a problem through use of procedures, modules, and/or objects. This problem should be of sufficient complexity to benefit from decomposition and/or use of objects.

For example, students could write a flashcard program in which each card is able to show both the question and answer and record user history.

Alternatively, students could create a simulation of an ecosystem in which sprites carry out behaviors, such as consuming resources.

9-12S.AP.18

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.AP.18	Demonstrate code reuse by creating programming solutions using libraries and APIs.	Algorithms & Programming	Modularity	Abstractions, Creating, Troubleshooting	4.2, 5.3, 6.2

Descriptive Statement

Code reuse is critical both for managing complexity in modern programs, but also in increasing programming efficiency and reliability by having programmers reuse code that has been highly vetted and tested. Software libraries allow developers to integrate common and often complex functionality without having to reimplement that functionality from scratch. Students identify, evaluate, and select appropriate application programming interfaces (APIs) from software libraries to use with a given language and operating system. They appropriately use resources such as technical documentation, online forums, and developer communities to learn about libraries and troubleshoot problems with APIs that they have chosen.

For example, students could import charting and graphing modules to display data sets, adopt an online service that provides cloud storage and retrieval for a database used in a multiplayer game, or import location services into an app that identifies points of interest on a map. Libraries of APIs can be student-created or publicly available (e.g., common graphics libraries or map/navigation APIs).

9-12S.AP.19

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.AP.19	Plan and develop programs for broad audiences using a specific software life cycle process.	Algorithms & Programming	Program Development	Collaborating, Creating	2.2, 2.3, 5.2

Descriptive Statement

Software development processes are used to help manage the design, development, and product/project management of a software solution. Various types of processes have been developed over time to meet changing needs in the software landscape. The systems development life cycle (SDLC), also referred to as the application development life cycle, is a term used in systems engineering, information systems, and software engineering to describe a process for planning, creating, testing, and deploying an information system. Other examples of common processes could include agile, spiral, or waterfall. Students develop a program following a specific software life cycle process, with proper scaffolding from the teacher.

For example, students could work in teams on a common project using the agile development process, which is based on breaking product development work into small increments.

Alternatively, students could be guided in implementing sprints to focus work on daily standup meetings or scrums to support efficient communication.

9-12S.AP.20

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.AP.20	Develop programs for multiple computing platforms.	Algorithms & Programming	Program Development	Creating	5.2

Descriptive Statement

Humans use computers in various forms in their lives and work. Depending on the situation, software solutions are more appropriate or valuable when available on different computational platforms or devices. Students develop programs for more than one computing platform (e.g. desktop, web, or mobile).

For example, students could develop a mobile app for a location-aware software product and a different program that is installed on a computer.

Alternatively, students could create a browser-based product and make it accessible across multiple platforms or computers (e.g., email).

9-12S.AP.21

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.AP.21	Identify and fix security issues that might compromise computer programs.	Algorithms & Programming	Program Development	Troubleshooting	6.2

Descriptive Statement

Some common forms of security issues arise from specific programming languages, platforms, or program implementation choices. Students read a given a piece of code that contains a common security vulnerability, explain the code's intended function or purpose, provide and explain examples of how a specific input could exploit that vulnerability (e.g., the program accessing data or performing in unintended ways), and implement a change in the code to mitigate this vulnerability.

For example, students could review code that takes a date as input, recognize that the code doesn't check for appropriate last days of the month, and modify the code to do that.

Alternatively, students could review code that supports entry of patient data (e.g., height and weight) and doesn't prompt users to double check unreasonable values (e.g., height at 6 feet and weight at 20 pounds).

9-12S.AP.22

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.AP.22	Develop and use a series of test cases to verify that a program performs according to its design specifications.	Algorithms & Programming	Program Development	Testing	6.1

Descriptive Statement

Testing software is a critically important process. The ability of students to identify a set of important test cases communicates their understanding of the design specifications and potential issues due to implementation choices. Students select and apply their own test cases to cover both general behavior and the edge cases which show behavior at boundary conditions.

For example, for a program that is supposed to accept test scores in the range of [0,100], students could develop appropriate tests (e.g., a negative value, 0, 100, and a value above 100).

Alternatively, students developing an app to allow users to create and store calendar appointments could develop and use a series of test cases for various scenarios including checking for correct dates, flagging for user confirmation when a calendar event is very long, checking for correct email address format for invitees, and checking for appropriate screen display as users go through the process of adding, editing, and deleting events.

9-12S.AP.23

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.AP.23	Modify an existing program to add additional functionality and discuss intended and unintended implications.	Algorithms & Programming	Program Development	Creating, Abstraction	5.3, 4.2

Descriptive Statement

Modularity and code reuse is key in modern software. However, when code is modified, the programmer should consider relevant situations in which this code might be used in other places. Students create and document modifications to existing programs that enhance functionality, and then identify, document, and correct unintended consequences.

For example, students could take an existing a procedure that calculates the average of a set of numbers and returns an integer (which lacks precision) and modify it to return a floating-point number instead. The student would explain how the change might impact multiple scenarios.

9-12S.AP.24

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.AP.24	Evaluate key qualities of a program through a process such as a code review.	Algorithms & Programming	Program Development	Testing	6.3

Descriptive Statement

Code reviews are a common software industry practice and valuable for developing technical communication skills. Key qualities of code include correctness, usability, readability, efficiency, and scalability. Students walk through code they created and explain how it works. Additionally, they follow along when someone else is explaining their code and ask appropriate questions.

For example, students could present their code to a group or visually inspect code in pairs.

Alternatively, in response to another student's presentation, students could provide feedback including comments on correctness of the code, comments on how code interacts with code that calls it, and design and documentation features.

9-12S.AP.25

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.AP.25	Use version control systems, integrated development environments (IDEs), and collaborative tools and practices (e.g., code documentation) while developing software within a group.	Algorithms & Programming	Program Development	Collaborating, Creating	2.4, 5.2

Descriptive Statement

Software development is a process that benefits from the use of tools that manage complexity, iterative development, and collaboration. Large or complex software projects often require contributions from multiple developers. Version control systems and other collaborative tools and practices help coordinate the process and products contributed by individuals on a development team. An integrated development environment (IDE) is a program within which a developer implements, compiles or interprets, tests, debugs, and deploys a software project. Students use common software development and documentation support tools in the context of a group software development project. At this level, facility with the full functionality available in the collaborative tools is not expected.

For example, students could use common version control systems to modify and improve code or revert to a previous code version.

Alternatively, students could use appropriate IDEs to support more efficient code design and development.

Additionally, students could use various collaboration, communication, and code documentation tools designed to support groups engaging in complex and interrelated work.

9-12S.AP.26

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.AP.26	Compare multiple programming languages, and discuss how their features make them suitable for solving different types of problems.	Algorithms & Programming	Program Development	Communicating	7.2

Descriptive Statement

Particular problems may be more effectively solved using some programming languages than other programming languages. Students provide a rationale for why a specific programming language is better suited for a solving a particular class of problem.

For example, students could explain how a language with a large library base can make developing a web application easier.

Alternatively, students could explain how languages that support particular programming paradigms (e.g., object-oriented or functional) can make implementation more aligned with design choices.

Additionally, students could discuss how languages that implement garbage collection are good for simplicity of memory management, but may result in poor performance characteristics.

9-12S.IC.27

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.IC.27	Evaluate computational artifacts with regard to improving their beneficial effects and reducing harmful effects on society.	Impacts of Computing	Culture	Testing, Inclusion	6.1, 1.2

Descriptive Statement

People design computational artifacts to help make the lives of humans better. Students evaluate an artifact and comment on aspects of it which positively or negatively impact users and give ideas for reducing the possible negative impacts.

For example, students could discuss how algorithms that screen job candidates' résumés can cut costs for companies (a beneficial effect) but introduce or amplify bias in the hiring process (a harmful effect).

Alternatively, students could discuss how turn-by-turn navigation tools can help drivers avoid traffic and find alternate routes (a beneficial effect), but sometimes channel large amounts of traffic down small neighborhood streets (a harmful effect).

Additionally, students could discuss how social media algorithms can help direct users' attention to interesting content (a beneficial effect), while simultaneously limiting users' exposure to information that contradicts pre-existing beliefs (a harmful effect).

9-12S.IC.28

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.IC.28	Evaluate how computational innovations that have revolutionized aspects of our culture might evolve.	Impacts of Computing	Culture	Communicating	7.2

Descriptive Statement

It is important to be able to evaluate current technologies and innovations and their potential for future impact on society. Students describe how a given computational innovation might change in the future and impacts these evolutions could have on society, economy, or culture.

For example, students could consider ways in which computers may support education (or healthcare) in the future, or how developments in virtual reality might impact arts and entertainment.

Alternatively, students could consider how autonomous vehicles will affect individuals' car ownership and car use habits as well as industries that employ human drivers (e.g., trucking, taxi service).

9-12S.IC.29

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.IC.29	Evaluate the impact of equity, access, and influence on the distribution of computing resources in a global society.	Impacts of Computing	Culture	Inclusion	1.2

Descriptive Statement

Computers, computation, and technology can help improve the lives of humans and support positive developments in society, economy, and/or culture. However, access to such resources is not the same for everyone in the world. Students define and evaluate ways in which different technologies, applications, or computational tools might benefit all people in society or might only benefit those with the greatest access or resources.

For example, students could describe ways in which groups of people benefit, do not benefit, or could benefit better by access to high-speed internet connectivity.

Alternatively, students could describe educational impacts of children not having access to a computer in their home.

9-12S.IC.30

Grade	Standard Identifier	Standard	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Subpractice(s)
9–12 Specialty	9-12S.IC.30	Debate laws and regulations that impact the development and use of software.	Impacts of Computing	Safety Law & Ethics	Communicating	7.2

Descriptive Statement

Laws and regulations influence what software gets developed and how society benefits or does not.

For example, students could debate the pros and cons of changes to regulations around net neutrality: Many believe that mandating that internet service providers (ISPs) maintain net neutrality facilitates competition between internet-based content providers and supports consumer choice, but others believe such regulations represent government overreach.

Alternatively, students could debate the impacts of different copyright rules in various countries and impacts on economy, society, and culture: Long-lasting copyrights in the United States enable creators to profit from their works but also prevent works from entering the public domain where they can be freely used and adapted to create new works.

References and Attributions

California Department of Education. 2015. *English Language Arts/English Language Development Framework for California Public Schools, Kindergarten Through Grade Twelve*. https://www.cde.ca.gov/ci/rl/cf/elaeldfrmwrksbeadopted.asp.

California Department of Education. 2016. *History–Social Science Framework for California Public Schools, Kindergarten Through Grade Twelve*. https://www.cde.ca.gov/ci/hs/cf/documents/hssframeworkwhole.pdf.

Code.org. 2015, July 2. *Computer Science Is the Fastest Growing AP Course of the 2010s* [Blog post]. https://blog.code.org/post/123032125688/apcs-2015.

College Board. 2016. National and state summary reports: California. https://research.collegeboard.org/programs/ap/data/archived/ap-2015.

College Board. 2018. Largest Course Launch in AP History. https://advancesinap.collegeboard.org/stem/ap-computer-science-principles.

The Conference Board. 2016. National Demand Rate and OES Employment Data by Occupation [Data file]. https://www.conference-board.org/.

DeNisco Rayome, A. 2017, June 16. CIO Jury: 83% of CIOs struggle to find tech talent. *TechRepublic*. https://www.techrepublic.com/article/cio-jury-83-of-cios-struggle-to-find-tech-talent/.

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

Fisher, A. 2015. The Fastest-growing STEM Major in the U.S. *Fortune*. https://fortune.com/2015/02/10/college-major-statistics-fastest-growing/.

Google. n.d. CS First [curriculum]. https://csfirst.withgoogle.com/en/home.

Google and Gallup. 2015. Searching for Computer Science: Access and Barriers in U.S. K-12 education. https://services.google.com/fh/files/misc/searching-for-computer-science_report.pdf.

Google and Gallup. 2017. Computer Science Learning: Closing the Gap: Rural and Small Town School Districts. https://services.google.com/fh/files/misc/computer-science-learning-closing-the-gap-rural-small-town-brief.pdf.

Horizon Media. 2015, October 5. Horizon Media Study Reveals Americans Prioritize STEM Subjects Over the Arts; Science is "Cool," Coding Is New Literacy. *PR Newswire*. https://www.prnewswire.com/news-releases/horizon-media-study-reveals-americans-prioritize-stem-subjects-over-the-arts-science-is-cool-coding-is-new-literacy-300154137.html.

K–12 Computer Science Framework. 2016. https://k12cs.org/wp-content/uploads/2016/09/K%E2%80%9312-Computer-Science-Framework.pdf.

Level Playing Field Institute. 2015. Path Not Found: Disparities in Access to CS Courses in California High Schools. https://www.kaporcenter.org/wp-content/uploads/2017/05/lpfi_path_not_found_report.pdf.

Papert, S. 2000. "What's the Big Idea? Toward a Pedagogy of Idea Power." *IBM Systems Journal*, 39 (3/4), 720–729.

Tucker, A., D. McCowan, F. Deek, C. Stephenson, J. Jones, and A. Verno. 2006. *A Model Curriculum for K–12 Computer Science: Final Report of the ACM K–12 Task Force Curriculum Committee.* 2nd ed. New York, NY: Association for Computing Machinery.

US Department of Labor, Bureau of Labor Statistics. 2015. *Employment Projections*. https://www.bls.gov/emp/tables.htm.

US Equal Employment Opportunity Commission. 2016. *Diversity in High Tech*. ~~https://www.eeoc.gov/special-report/diversity-high-tech~~. [Link no longer available]

California Computer Science Standards: Appendix

Contents

I. Guide for Leadership 160

Strategies to Support Computer Science Standards Implementation 160

Supporting Computer Science through Professional Learning 161

II. Guide for Flexible Implementation 163

Flexible Implementation Models for Computer Science Standards 163

Integrating Computer Science 163

III. Guide for Instructional Practices Alignment 167

Instructional Practices Alignment Considerations 167

Assessment 168

Universal Access 168

IV. Interdisciplinary Connections 173

Practices 173

Interdisciplinary Connections to Standards by Grade Band 178

Grade levels K–2 178

Grade levels 3–5 190

Grade levels 6–8 194

Grade levels 9–12 198

V. Career Technical Education (CTE) Connections 207

CTE Standards for Career Ready Practice 207

Information and Communication Technologies Sector 210

Other Sectors 220

VI. Connections to Postsecondary Education 227

California State University/University of California Freshman Minimum Admission Requirements 227

Advanced Placement (AP) 228

International Baccalaureate 241

VII. Glossary 243

VIII. References 260

Guide for Leadership

Strategies to Support Computer Science Standards Implementation

Educational leaders foster systemic change by clearly communicating a strong, compelling vision regarding the necessity of computer science education and its importance for developing college and career readiness as well as lifelong learning. For additional guidance regarding the need for computer science implementation, refer to the "Why Computer Science?" section of the introduction to the standards.

Consistent messaging from leadership is vital to ensure coherence across an organization. This messaging is most effective when communicated to multiple stakeholders including administrators, teachers, paraprofessionals, parents, community members, and students. Shared vision and common language builds coherence, inspires stakeholders and promotes sustainable change.

While communication of a compelling vision is vital, it must be accompanied by systems of support to build capacity of administrators, educators, community members, and students. The vision should be reflected in school board policies and resolutions, and in Local Control and Accountability Plans (LCAP) within the action and services portion of the annual update. Local education agencies are encouraged to consider adopting board resolutions supporting computer science, such as creating a computer science advisory committee. Educational leaders must also consider a sustained investment in resources to maintain vision implementation over time. Local education agencies are urged to consider their parcel tax and/or bonds to be submitted during elections and add computer science as an addendum to their district technology plan and/or district strategic plan.

The following strategies may be used to support standards implementation:

- **Communication**
  - Define a system-wide vision written in a language accessible to teachers, curriculum leaders, community members, school board members, students, parents, and families
  - Communicate the vision to stakeholders including multimedia methods (website, infographics, posters, etc.)
  - Conduct community meetings and parent nights to build awareness for computer science education and provide resources
  - Include student voices in messaging to stakeholders and encourage students to support the vision
  - Survey educators and other stakeholders to assess needs for professional learning and additional support
- **Building Capacity**
  - Ensure stakeholders can define the vision and understand their role in its implementation
  - Provide professional learning opportunities for teachers and paraeducators in alignment to vision
  - Provide professional learning for administrators and encourage them to learn alongside teachers

- Facilitate student leadership opportunities to promote students teaching students, cross-age tutoring, and mentorships
- Personalize job-embedded professional learning to maximize effectiveness

- **Sustainability**
  - Align computer science work to the vision, using common language to foster coherence across the system
  - Encourage educators to engage in peer observations and collaborative learning communities to share best practices
  - Foster collaboration across departments and grade levels to promote interdisciplinary connections and greater coherence within the system
  - Build partnerships between students and industry professionals and/or organizations
  - Reflect upon professional learning and implementation to guide continual improvement

Supporting Computer Science through Professional Learning

Professional learning for educators and administrators builds collective efficacy and directly impacts student achievement. Educators who will be teaching computer science may or may not have a computer science background or certification. Therefore, care must be made to customize the professional learning experience by differentiating activities and instruction based on computer science experience or certification.

Professional learning should lower educator anxiety regarding content knowledge by focusing on growth mindsets and building a safe culture of risk-taking. Professional learning experiences must always align to the context of educators' content area, instructional goals, and courses taught. The choice of programming languages or tools should follow, not dictate, the learning goals that align to classroom instruction. Ideal professional learning includes time for educators to experience learning in a pedagogically sound, learner-centered environment in which they have time to explore new concepts, followed by time to collaborate with colleagues in planning ways to apply these concepts in their own classroom. Peer observations, follow-up coaching, and time for reflection further cements the learning, builds capacity, and supports sustainable change.

Due to issues of equity within computer science education, professional learning in computer science should address instructional practices that promote inclusivity and provide universal access to all students, particularly those from underserved populations. Professional learning in computer science should also address ways to increase access depending on device availability. The standards were developed to be learned within a variety of environments with differing levels of device availability. In schools that do not offer a computing device for each student, strategies may include students working collaboratively on shared computing devices, rotating students through a computing experience via a weekly schedule within their classroom, or scheduling multiple classrooms across a campus to share a room that contains multiple computing devices.

Policies to Promote Computer Science

Vision and systems of support must be accompanied by updates to policy in order to achieve sustainable change. Recommended policies to promote and support the standards include:

- **Define 'computer science' and formally adopt the standards as local school board policy.** Computer science instruction should be guided by standards and align to an accurate definition of computer science.
- **Allocate funding for relevant, rigorous professional learning and course development support.** Local education agencies and schools need to dedicate funding for building capacity of educators to teach computer science, including development or purchase of course materials and technical infrastructure.
- **Maintain awareness of, and support, certification pathways.** In addition to traditional certification pathways, incentives and expedited, alternative pathways may be created to address short- and long-term need for computer science teachers.
- **Develop partnerships with higher education organizations.** Local education agencies and schools can benefit from creating direct pathways for preservice teachers to enter service in high need areas.
- **Create dedicated computer science positions at local education agencies.** Leadership positions devoted to computer science education promote sustainable change and build capacity.
- **Require all high schools to offer computer science.** While computer science is to be available for all grades K–12, ensuring all high schools offer at least one discrete computer science course will assist in creating momentum to foster systemic change.
- **Allow computer science to count as a graduation requirement.** Computer science courses that count toward a graduation requirement rather than as an elective result in increased participation by students.
- **Allow computer science to count as an admissions requirement for institutes of higher education.** Computer science courses that meet an admission requirement for an institute of high education will result in increased participation by students.

Guide for Flexible Implementation

The overarching goal of the standards is guidance that fosters computer science instruction for all students. Implementation is flexible and should be based on needs of local capacity and context.

Flexible Implementation Models for Computer Science Standards

Opportunity for all *could* include, but is not limited to, one or more of the following options. Beginning at the top row, the examples move from basic exposure to broad and deep exposure at the bottom row.

Elementary Level	Middle School Level	High School Level
Integrated into the general education classroom	Integrated into math, science, and/or other subjects	Integrated into math, science, and/or other subjects
Integrated into an existing special classroom (e.g., media arts, computer lab, makerspace)	Integrated into an existing special classroom (e.g., media arts, computer lab, makerspace)	Introductory and/or independent course(s)
Independent special class (push-in or pull-out similar to models sometimes used for music, arts, etc.)	Introductory and/or independent course(s)	A menu of course options available for all students, including advanced courses (e.g., honors, AP, IB)
Integrated for all with additional independent enrichment course via extended hours options	Integrated for all with additional independent enrichment course elective options	Specialized courses (e.g., game design, cybersecurity, networking, robotics)

For a detailed description of building course pathways, view the source K–12 Computer Science Framework at https://k12cs.org/curriculum-assessment-pathways/.

Integrating Computer Science

Particularly at grade levels kindergarten through five, computer science education can be integrated into multiple subject classrooms to move toward an interdisciplinary approach. Computer science fits naturally into an interdisciplinary learning environment. Integration of computing experiences into multiple subject classrooms has been in existence for many years (Papert 1980). The standards contain interdisciplinary connection examples in grades K–8. Consider the following examples taken from the standards, each with multiple interdisciplinary connections.

- **K-2.AP.16** For example, when given images placed in a random order, students could give step-by-step commands to direct a robot, or a student playing a robot, to navigate to the images in the correct sequence. Examples of images include storyboard cards from a familiar story (CA CCSS for ELA/Literacy RL.K.2, RL.1.2, RL.2.2) and locations of the sun at different times of the day (CA NGSS: 1-ESS1-1).

- **3-5.DA.8** For example, students could create and administer electronic surveys to their classmates. Possible topics could include favorite books, family heritage, and after-school activities. Students could then create digital displays of the data they collected, such as column histogram charts showing the percent of respondents in each grade who selected a particular favorite book. Finally, students could make quantitative statements supported by the data, like which books are more appealing to specific ages of students. As an extension, students could write an opinion piece stating a claim and support it with evidence from the data they collected (CA CCSS for Mathematics 3.MD.3, 4.MD.4, 5.MD.2) (CA CCSS for ELA/Literacy W.3.1, W.4.1, W.5.1).

- **6-8.IC.24** For example, students could discuss the benefits and dangers of the increased accessibility of information available on the internet, and then compare this to the advantages and disadvantages of the introduction of the printing press in society (HSS.7.8.4).

Elementary and middle schools often have greater flexibility than high schools in their options for integrating computer science for all students. Options may include instructional units dedicated to computer science within technology, career exploration, or media arts classes, weekly computing classes offered as electives, or integration of computer science into other content areas. These courses could be taught by elementary education teachers with a multiple subject credential, content area teachers (mathematics, science, technology, music, art, or media arts), or dedicated computer science teachers. While the implementation models vary, local education agencies need to align computer science education with the standards as covered in the curriculum alignment considerations section of the appendix.

The interdisciplinary guide section of the appendix provides general education, special classroom, and independent course educators with guidance regarding interdisciplinary connections to encourage integration and/or cross-departmental collaboration. Examples of an integrated approach could include but are not limited to:

- Science teachers using computer science practices to analyze data in labs
- Science teachers having students program simulations
- Science and health teachers having students create and modify computational models that represent the transmission of infectious disease in order to predict changes in infected, susceptible, and recovered populations
- Math teachers creating computational artifacts with students to illustrate graphing
- Math teachers having students use computer algebra systems to explore problems that are realistic, yet

computationally intensive and impractical to attempt using traditional techniques

- Art teachers bringing algorithms into a discussion of filters for photographs
- Art teachers having students write and modify software to produce digital works of art that would be impractical to create with traditional techniques
- English teachers having students create algorithms to analyze text when comparing characteristics of writing samples from various authors
- Physical education teachers having students collect and analyze data related to physical activities using computational tools and techniques
- Social studies teachers having students produce digital map sequences, computationally controlled to determine animation speeds, to illustrate changes in populations and related social issues over time
- English language development and world languages teachers having students collect and analyze data on the global impact of biliteracy

Discrete Computer Science

Computer science can be taught as a discrete, independent course. For example, at the elementary level, the course can exist as a weekly push-in or pull-out program from a specialized computer science educator. In middle school, computer science can be taught as a semester-long or yearlong course for a particular grade level or for all levels. In high school, computer science can be taught as a standalone course—as an introductory course, an advanced placement course, and/or a specialized course.

The danger of this model is that standalone courses are often offered as electives, preventing access for those students who do not opt in to the courses. Even when using a standalone, independent model, educators are encouraged to seek out interdisciplinary connections and collaborate with colleagues in other departments to increase relevance for students. Computer science education can also be provided by enhancing pre-existing technology education credits and courses.

The core K–12 standards are designed for all students, even those in grade levels nine through twelve. While the specialty standards can aid the development of elective courses, local education agencies are encouraged to begin by offering at least one computer science course per high school. As computer science is implemented in grade levels kindergarten through eight, the need for high schools to increase their rigor and expand computer science offerings will grow. High schools may build their computer science offerings by including specialty courses that can provide college entrance credit, career and technical preparation, or advanced placement courses. While these courses may be more advanced or specialized than the standards call for, the K–12 progression provides a conceptual foundation for their increasingly complex content.

Local education agencies may consider whether to categorize computer science within an academic pathway, a Career Technical

Education (CTE) pathway, or both. It is appropriate to house computer science in both CTE and academic pathways. The early, foundational courses in a CTE program of study can be dual-coded as part of the CTE pathway as well as a math, science, or technology credit.

Guide for Instructional Practices Alignment

Instructional Practices Alignment Considerations

In accordance to the standards, computer science instruction and instructional practices should align to the five concepts and seven practices of computer science, grounded in solid pedagogical philosophies of student-led learning environments. The five concepts are:

- Computing Systems
- Networks and the Internet
- Data and Analysis
- Algorithms and Programming
- Impacts of Computing

And the seven practices include:

- Fostering an Inclusive Computing Culture
- Collaborating Around Computing
- Recognizing and Defining Computational Problems
- Developing and Using Abstractions
- Creating Computational Artifacts
- Testing and Refining Computational Artifacts
- Communicating About Computing

A comprehensive computer science education requires inclusion of each of the concepts and practices, as defined in the Introduction. Computer science is more than coding (programming). Leaders and educators must ensure that the computer science courses and corresponding instructional resources offered in their local education agencies reflect the breadth of computer science areas via the five concepts, and also provides the opportunity for students to actively engage with the content via the seven practices.

The concepts and practices are to be integrated into instruction simultaneously in order to provide relevant, meaningful learning experiences for students. Computer science instruction should allow students to construct content knowledge through active, exploratory activities that provide opportunities to solve problems, create, collaborate, and communicate. Educators are urged to lead with learning, never with tools (programming tools, equipment, or computing languages). The five concepts should drive instructional planning, accompanied by the seven practices as a means of engaging students with the content. Tools are to be chosen not as a focus, but as a means of supporting students in mastering the concepts while engaging in practices.

Instructional practices must also take into consideration the progressive grade-bands of the framework. The standards are designed to be developmentally appropriate, building in complexity per grade-band. Careful attention to these progressions will ensure a coherent computer science education for students.

Instructional practices should reflect countless socially relevant and culturally situated contexts. Students should be engaged in learning experiences that promote social connections across cultures, celebrate diversity, and allow opportunities to address relevant community issues.

Assessment

Assessment of computer science in classrooms should be used to drive instructional planning and measure performance toward mastery of the standards. Traditional assessments that seek one solution to a problem are not recommended. The following assessment models more accurately reflect the computer science field, motivate students in meaningful learning experiences, and more accurately assess students' depth of understanding.

Authentic projects allow educators to gauge not only students' content knowledge, but their ability to apply this knowledge to real-world contexts. Throughout a long-term project, educators may use predetermined timelines as check-ins to be used as formative assessment. The assessment data collected should be used to support students according to their specific learning needs. Rubrics are recommended to ensure that students and educators have clear expectations of the final goals of the project. This also promotes student metacognition, as learners may monitor their progress according to the rubric.

Portfolios developed over time provide students and educators the opportunity to validate the process of learning computer science, and to celebrate growth over time. As students create portfolios, they build confidence and begin to take ownership of their learning.

Educators may assess computer science concepts and skills via performance tasks. These assessments may evaluate multiple concepts and practices simultaneously. As performance tasks contain more than a single method and/or answer to a question, they promote student creativity, allow students to demonstrate problem solving skills, and provide educators with valuable insight into student understanding.

Assessment in computer science should contain breadth across concepts and integrate practices, according to the K–12 Computer Science Framework at https://k12cs.org/. For example, teachers can assess students' ability to analyze the advantages and disadvantages of different encryption algorithms. This evaluation addresses the idea of algorithmic performance (Algorithms and Programming) as well as cybersecurity (Networks and the Internet).

When programming is the focus, students should be assessed on not only their ability to write the program but also their ability to communicate the product's significance and development process (Communicating About Computing). This also includes the collaboration among members (Collaborating Around Computing). For instance, students can submit planning documents used to produce the program, do a presentation on the impact that their program will have on a target audience, and write a reflection on how the team worked to put the program together.

Universal Access

The standards are designed to be accessible to all, regardless of race, ethnicity, gender, socioeconomic status, language, religion, sexual orientation, cultural affiliation, or special needs. In addition to developing a schedule that ensures each and every student receives access to core concepts and practices, computer science learning experiences must be designed to be inclusive of all learners beginning with instructional planning.

The Universal Design for Learning (UDL) framework provides a proactive, research based framework to guide educators in planning instruction that meets the varying needs of a diverse group of students (visit the UDL website at https://medium.com/udl-center, and the Center for Applied Special Technology (CAST) website at https://www.cast.org).

UDL principles for instruction include: Principle I) Provide multiple means of engagement to tap individual learners' interests, challenge them appropriately, and motivate them to learn; Principle II) Provide multiple means of representation to give students various ways of acquiring, processing, and integrating information and knowledge; and Principle III) Provide multiple means of action and expression to provide students with options for navigating and demonstrating learning (CAST 2019).

The following tables, developed by the Creative Technology Research Lab at the University of Illinois, demonstrate examples of UDL integration in computer science instruction (Israel, Lash, and Jeong 2017).

Multiple Means of **Representation**	Multiple Means of **Action and Expression**	Multiple Means of **Engagement**
Provide options for perception: ▪ Model computing using physical representations and drawings ▪ Give access to modeled code while students work independently ▪ Provide access to video tutorials of computing tasks ▪ Select coding apps and websites that allow the students to adjust visual settings (such as font size and contrast) and that are compatible with screen readers	*Provide options for physical action:* ▪ Provide teacher's codes as templates ▪ Include unplugged activities that show physical relationship of abstract computing concepts ▪ Use assistive technology including larger and smaller mice, and touch-screen devices ▪ Select coding apps and websites that allow coding with keyboard shortcuts, in addition to dragging-and-dropping with a mouse	*Provide options for recruiting interest:* ▪ Give students choices like choosing a project, software, or topic ▪ Allow students to make projects relevant to culture and age, and addressing societal needs ▪ Minimize possible common "pitfalls" for both computing and content ▪ Allow for differences in the pacing and length of work sessions ▪ Provide options to increase or decrease sensory stimulation, like listening to music with headphones or noise-cancelling headphones)

Multiple Means of **Representation**	Multiple Means of **Action and Expression**	Multiple Means of **Engagement**
		▪ Allow for differences in pacing and the length of work sessions ▪ Allow students to address standards as part of larger projects ▪ Allow computer programming assignments that facilitate artistic expression
Provide options for perception: ▪ Model computing using physical representations and drawings ▪ Give access to modeled code while students work independently ▪ Provide access to video tutorials of computing tasks ▪ Select coding apps and websites that allow the students to adjust visual settings (such as font size and contrast) and that are compatible with screen readers	*Provide options for physical action:* ▪ Provide teacher's codes as templates ▪ Include unplugged activities that show physical relationship of abstract computing concepts ▪ Use assistive technology including larger and smaller mice, and touch-screen devices ▪ Select coding apps and websites that allow coding with keyboard shortcuts, in addition to dragging-and-dropping with a mouse	*Provide options for recruiting interest:* ▪ Give students choices like choosing a project, software, or topic ▪ Allow students to make projects relevant to culture and age, and addressing societal needs ▪ Minimize possible common "pitfalls" for both computing and content ▪ Allow for differences in the pacing and length of work sessions ▪ Provide options to increase or decrease sensory stimulation, like listening to music with headphones or noise-cancelling headphones) ▪ Allow for differences in pacing and the length of work sessions ▪ Allow students to address standards as part of larger projects

Multiple Means of **Representation**	Multiple Means of **Action and Expression**	Multiple Means of **Engagement**
		▪ Allow computer programming assignments that facilitate artistic expression
Provide options for language mathematical expressions, and symbols: ▪ Teach and review content specific vocabulary ▪ Teach and review computing vocabulary like code, animations, computing, and algorithms ▪ Post anchor charts and provide reference sheets with images of blocks or common syntax when using text	*Provide options for expression and communication:* ▪ Give options of unplugged activities and computing software and materials ▪ Give opportunities to practice computing skills and content through projects that build prior lessons ▪ Provide sentence starters or checklists for communicating to collaborate, give feedback, and explain work ▪ Create physical manipulatives of commands, blocks or lines of code ▪ Provide options that include starter code	*Provide options for sustaining effort and persistence:* ▪ Remind students of computing and content goals ▪ Provide support or extensions for students to keep engaged ▪ Teach and encourage peer collaboration by sharing products ▪ Utilize pair programming and group work with clearly defined roles ▪ Discuss the integral role of perseverance and problem solving in computer science, and recognize students for demonstrating perseverance and problem solving in the classroom
Provide options for comprehension: ▪ Activate background knowledge by making computing tasks interesting and culturally relevant ▪ State lesson content and computing goals ▪ Encourage students to ask questions as comprehension checkpoints	*Provide options for executive functions:* ▪ Guide students to set goals for long-term projects ▪ Record students' progress; have planned checkpoints during lessons for understanding and progress for computing skills and content	*Provide options for self-regulation:* ▪ Communicate clear expectations for computing tasks, collaboration, and seeking help ▪ Develop ways for students to self-assess and reflect on their own projects and those of others

Multiple Means of **Representation**	Multiple Means of **Action and Expression**	Multiple Means of **Engagement**
▪ Use relevant analogies and make cross-curricular connections explicit, like comparing iterative product development to the writing process ▪ Provide graphic organizers for students to "translate" programs into pseudocode	▪ Provide exemplars of completed products ▪ Embed prompts to stop and plan, test, or debug throughout a lesson or project ▪ Provide graphic organizers to facilitate planning, goal-setting, and debugging ▪ Provide explicit instruction on skills such as asking for help, providing feedback, and using problem-solving techniques ▪ Demonstrate debugging with think-alouds	▪ Use assessment rubrics that evaluate both content and process ▪ Break up coding activities with opportunities for reflection, such as turn-and-talks or written questions ▪ Acknowledge difficulty and frustration; model different strategies for dealing with frustration appropriately

Interdisciplinary Connections

Life is not divided into subject areas, and computer science is no different. As a field, computer science spans multiple disciplines.

The following resources provide guidance for interdisciplinary connections between the computer science standards and other curriculum standards adopted by the California State Board of Education. CTE connections are listed in section V of the appendix.

The interdisciplinary connections are meant to be general suggestions as to relationships between content areas and do not constitute guidance for synonymous instruction between disciplines.

Practices

Broad interdisciplinary connections from kindergarten to grade twelve, followed by specific cross-disciplinary references by grade band, are provided below.

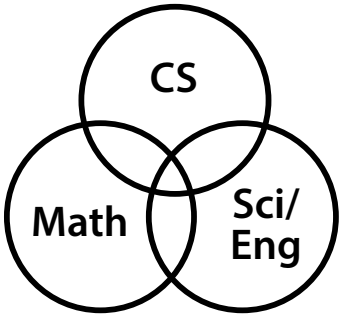
Computer Science Practice	Related Interdisciplinary Connections
1. Fostering an Inclusive Computing Culture	**Visual and Performing Arts** Historical and Cultural Context (Strand 3) **Next Generation Science Standards** Using Mathematics and Computational Thinking (Science and Engineering Practice 5)
2. Collaborating Around Computing	**English Language Arts** College and Career Readiness Anchor Standards for Speaking and Listening Comprehension and Collaboration (Standards 1, 2, 3) **English Language Development** Part I: Interacting in Meaningful Ways Communicative Mode: Collaborative (Standards 1–4) **Next Generation Science Standards** Using Mathematics and Computational Thinking (Science and Engineering Practice 5)

Computer Science Practice	Related Interdisciplinary Connections
3. Recognizing and Defining Computational Problems	**Next Generation Science Standards** Asking Questions (for science) and defining problems (for engineering) (Science and Engineering Practice 1) Using Mathematics and Computational Thinking (Science and Engineering Practice 5) **Mathematics** Make Sense of Problems and Persevere in Solving Them (Standards of Mathematical Practice 1) **Health Education Content Standards** Decision Making (Standard 5)
4. Developing and Using Abstractions	**Next Generation Science Standards** Developing and Using Models (Science and Engineering Practice 2) **Mathematics** Reason Abstractly and Quantitatively (Standards of Mathematical Practice 2) Model with Mathematics (Standards of Mathematical Practice 4) Look For and Make Use of Structure (Standards of Mathematical Practice 7) Look For and Express Regularity in Repeated Reasoning (Standards of Mathematical Practice 8)
5. Creating Computational Artifacts	**Visual and Performing Arts** Creative Expression (Strand 2)

Computer Science Practice	Related Interdisciplinary Connections
	English Language Arts College and Career Readiness Anchor Standards for Writing Text Types and Purposes (Standards 1, 2, 3) Production and Distribution of Writing (Standards 4, 5, 6) **Next Generation Science Standards** Constructing Explanations (for science) and Designing Solutions (for engineering) (Science and Engineering Practice 6)
6. Testing and Refining Computational Artifacts	**Next Generation Science Standards** Planning and Carrying Out Investigations (Science and Engineering Practice 3) Developing and Using Models (Science and Engineering Practice 2)
7. Communicating About Computing	**California History–Social Studies Framework** Argumentative Writing (Concept and Disciplinary Practice) **Model School Library Standards** Students Use Information (Standard 3) **Next Generation Science Standards** Analyzing and Interpreting Data (Science and Engineering Practice 4) Engaging in Argument from Evidence (Science and Engineering Practice 7) Obtaining, Evaluating, and Communicating Information

Computer Science Practice	Related Interdisciplinary Connections
	(Science and Engineering Practice 8) **Mathematics** Construct Viable Arguments and Critique the Reasoning of Others (Standards of Mathematical Practice 3) Attend to Precision (Standards of Mathematical Practice 6) **English Language Arts** College and Career Readiness Anchor Standards for Writing Production and Distribution of Writing (Standards 4, 5, 6) College and Career Readiness Anchor Standards for Speaking and Listening Comprehension and Collaboration (Standards 1, 2, 3) College and Career Readiness Anchor Standards for Speaking and Listening Presentation of Knowledge and Ideas (Standards 4, 5, 6) College and Career Readiness Anchor Standards for Language Vocabulary Acquisition and Use (Standards 4, 6) **English Language Development** Part I: Interacting in Meaningful Ways Communicative Mode: Productive (Standards 9-12) **Health Education Content Standards** Interpersonal Communication (Standard 4)

The Computer Science (CS) practices intersect with the practices described in the Next Generation Science Standards, which also include engineering, and the Common Core Standards for Mathematical Practice (see figure below).



CS and Math Intersection	CS and Science/Engineering Intersection	CS, Math, and Science/Engineering Intersection
Develop and use abstractions M2. Reason abstractly and quantitatively M7. Look for and make use of structure M8. Look for and express regularity in repeated reasoning CS4. Developing and Using Abstractions **Use tools when collaborating** M5. Use appropriate tools strategically CS2. Collaborating Around Computing **Communicate precisely** M6. Attend to precision CS7. Communicating About Computing	**Communicate with data** S4. Analyze and interpret data CS7. Communicating About Computing **Create artifacts** S3. Plan and carry out investigations S6. Construct explanations and design solutions CS4. Developing and Using Abstractions CS5. Creating Computational Artifacts CS6. Testing and Refining Computational Artifacts	**Model** S2. Develop and use models M4. Model with mathematics CS4. Developing and Using Abstractions CS6. Testing and Refining Computational Artifacts **Use computational thinking** S5. Use mathematics and computational thinking CS3. Recognizing and Defining Computational Problems CS4. Developing and Using Abstractions CS5. Creating Computational Artifacts **Define problems** S1. Ask questions and define problems M1. Make sense of problems and persevere in solving them CS3. Recognizing and Defining Computational Problems **Communicate rationale** S7. Engage in argument from evidence S8. Obtain, evaluate, and communicate information M3. Construct viable arguments and critique the reasoning of others CS7. Communicating About Computing

Adapted from K–12 Computer Science Framework 2016, p. 72.
https://k12cs.org/wp-content/uploads/2016/09/K%E2%80%9312-Computer-Science-Framework.pdf

Communicating about Computing is a key practice that is reinforced throughout the CS standards by asking students to describe, explain, and justify program design decisions, computing phenomena, and the impacts of computing on society. These CS standards support numerous ELA speaking, listening, and writing standards as well as ELD standards, including synthesizing information from multiple sources, writing discipline-specific arguments, developing facility with technical vocabulary, and understanding diverse perspectives.

Interdisciplinary Connections to Standards by Grade Band

Note: Cross-disciplinary references from the interdisciplinary examples listed in the standards document may or may not be included on this table, as the connections listed below are meant to be general connections rather than aligned to specific examples.

Grade levels K–2

Key: Asterisk symbol (*) = Not found.

Computer Science Standard	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Physical Education	Health Education
K-2.CS.1	W.K.6, W.1.6, W.2.6, SL.K.5, SL.1.5, SL.2.5	P.I.K.2, P.I.K.9, P.I.K.10, P.I.K.12, P.I.1.2, P.I.1.9, P.I.1.10, P.I.1.12, P.I.2.2, P.I.2.9, P.I.2.10, P.I.2.12	*	*	*	*	K.1.3.G, K.1.4.A, K.3.3, K.4.1, 1.1.3.A, 1.1.4, 1.3.3, 1.4.1, 2.1.3.B, 2.1.3.D, 2.1.4, 2.4.3.B	*	*

Computer Science Standard	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Physical Education	Health Education
K-2.CS.2	SL.K.4, SL.K.5, SL.K.6, SL.1.4, SL.1.5, SL.1.6, SL.2.4, SL.2.5, SL.2.6	P.I.K.1, P.I.K.3, P.I.K.9, P.I.K.11, P.I.K.12, P.II.K.1, P.II.K.2, P.II.K.3, P.II.K.4, P.II.K.5, P.I.1.3, P.I.1.9, P.I.1.11, P.I.1.12, P.II.1.1, P.II.1.2, P.II.1.3, P.II.1.4, P.II.1.5, P.II.1.6, P.II.1.7, P.I.2.1, P.I.2.3, P.I.2.4, P.I.2.9, P.I.2.11, P.I.2.12, P.II.2.1; P.II.2.2, P.II.2.3, P.II.2.4, P.II.2.5, P.II.2.6, P.II.2.7	*	*	*	K-2-ETS1-2	1.1.3.C, 2.3.3.B, 2.3.3.D	*	*

Computer Science Standard	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Physical Education	Health Education
K-2.CS.3	SL.K.4, SL.K.5, SL.K.6, SL.1.4, SL.1.5, SL.1.6, SL.2.4, SL.2.5, SL.2.6	P.I.K.1, P.I.K.3, P.I.K.9, P.I.K.11, P.I.K.12, P.II.K.1, P.II.K.2, P.II.K.3, P.II.K.4, P.II.K.5, P.I.1.3, P.I.1.9, P.I.1.11, P.I.1.12, P.II.1.1, P.II.1.2, P.II.1.3, P.II.1.4, P.II.1.5, P.II.1.6, P.II.1.7, P.I.2.1, P.I.2.3, P.I.2.4, P.I.2.9, P.I.2.11, P.I.2.12, P.II.2.1, P.II.2.2, P.II.2.3, P.II.2.4, P.II.2.5, P.II.2.6, P.II.2.7	*	*	*	*	*	*	*

Computer Science Standard	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Physical Education	Health Education
K-2.NI.4	SL.K.4, SL.K.5, SL.K.6, SL.1.4, SL.1.5, SL.1.6, SL.2.4, SL.2.5, SL.2.6	P.I.K.1, P.I.K.3, P.I.K.9, P.I.K.11, PI.K.12, P.II.K.1, P.II.K.2, P.II.K.3, P.II.K.4, P.II.K.5, P.I.1.3, P.I.1.9, P.I.1.11, P.I.1.12, P.II.1.1, P.II.1.2, P.II.1.3, P.II.1.4, P.II.1.5, P.II.1.6, P.II.1.7, P.I.2.1, P.I.2.3, P.I.2.4, P.I.2.9, P.I.2.11, P.I.2.12, P.II.2.1, P.II.2.2, P.II.2.3, P.II.2.4, P.II.2.5, P.II.2.6, P.II.2.7	*	*	*	1-PS-4-4, K-2-ETS1-2	1.3.1.B	*	*

Computer Science Standard	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Physical Education	Health Education
K-2.NI.5	SL.K.4, SL.K.5, SL.K.6, SL.1.4, SL.1.5, SL.1.6, SL.2.4, SL.2.5, SL.2.6	P.I.K.1, P.I.K.3, P.I.K.9, P.I.K.11, P.I.K.12, P.II.K.1, P.II.K.2, P.II.K.3, P.II.K.4, P.II.K.5, P.I.1.3, P.I.1.9, P.I.1.11, P.I.1.12, P.II.1.1, P.II.1.2, P.II.1.3, P.II.1.4, P.II.1.5, P.II.1.6, P.II.1.7, P.I.2.1, P.I.2.3, P.I.2.4, P.I.2.9, P.I.2.11, P.I.2.12, P.II.2.1, P.II.2.2, P.II.2.3, P.II.2.4, P.II.2.5, P.II.2.6, P.II.2.7	*	*	*	*	K.3.1.A, 2.3.1.E	*	*
K-2.NI.6	*	*	*	*	Music.K.1.1	1-PS-4-4	*	*	*

Computer Science Standard	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Physical Education	Health Education
K-2.DA.7	W.K.5, W.K.6, W.1.5, W.1.6, W.2.5, W.2.6	PI.K.2, PI.K.10, PI.K.12, PII.K.1, PII.K.2, PII.K.3, PII.K.4, PII.K.5, PII.K.6, PI.1.2, PI.1.10, PI.1.12, PII.1.1, PII.1.2, PII.1.3, PII.1.4, PII.1.5, PII.1.6, PII.1.7, PI.2.2, PI.2.4, PI.2.10, PI.2.12, PII.2.1, PII.2.3, PII.2.4, PII.2.5, PII.2.6, PII.2.7	*	*	*	*	*	*	*
K-2.DA.8	*	*	*	K.MD.3, 1.MD.4, 2.MD.4 2.MD.9 2.MD.10	*	K-2-ETS1-3, K-PS2-2, K-LS1-1, K-ESS2-1, 1-ESS1-1, 1-ESS1-2, 2-PS1-2	*	*	*

Computer Science Standard	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Physical Education	Health Education
K-2.DA.9	*	PI.K.9, PI.K.12b, PII.K.2, PII.K.5, PII.K.6, PI.1.9, PI.1.12b, PII.1.2, PII.1.5, PII.1.6, PI.2.9, PI.2.12b, PII.2.2, PII.2.5, PII.2.6	*	K.MD.3, K.G.4, 1.MD.4, 2.MD.10	*	2-PS1-1, 2-PS1-2, K-PS3-1, K-ESS2-1, K-ESS3-2, 1-ESS1-1, 1-ESS1-2	*	*	*
K-2.AP.10	W.K.3, W.1.3, W.2.3	PI.K.10, PII.K.1, PII.K.2, PII.K.6, PI.1.10, PII.1.1, PII.1.2, PII.1.6, PII.1.7, PI.2.10, PII.2.1, PII.2.2, II.2.6, PII.2.7	K.5	*	Dance 1.2.3, Dance 1.2.5, Dance 2.2.3	*	*	2.1.18, 2.1.19	*
K-2.AP.11	*	*	*	*	*	*	*	*	*

Computer Science Standard	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Physical Education	Health Education
K-2.AP.12	*	*	*	*	Dance 1.2.3, Dance 1.2.5, Dance 2.2.3	*	*	2.1.18, 2.1.19	*
K-2.AP.13	*	*	*	K.G.5, 1.G.2, 2.NBT.9	*	2-PS1-3	*	*	1.6.1.P
K-2.AP.14	W.K.3, W.1.3, W.2.3	PI.K.10, PII.K.1, PII.K.2, PII.K.6, PI.1.10, PII.1.1, PII.1.2, PII.1.6, PII.1.7, PI.2.10, PII.2.1, PII.2.2, PII.2.6, PII.2.7	*	*	*	*	*	*	K.7.2.N, K.6.1.M, 1.6.1.P, 2.7.2.N
K-2.AP.15	*	PI.K.9, PI.1.9, PI.2.9, PI.K.10, PI.1.10, PI.2.10	*	*	*	*	2.3.1.B	*	*
K-2.AP.16	*	*	*	*	*	*	*	*	*

Computer Science Standard	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Physical Education	Health Education
K-2.AP.17	SL.K.4, SL.K.5, SL.K.6, SL.1.4, SL.1.5, SL.1.6, SL.2.4, SL.2.5, SL.2.6	P.I.K.1, P.I.K.3, P.I.K.9, P.I.K.11, P.I.K.12, P.II.K.1, P.II.K.2, P.II.K.3, P.II.K.4, P.II.K.5, P.I.1.3, P.I.1.9, P.I.1.11, P.I.1.12, P.II.1.1, P.II.1.2, P.II.1.3, P.II.1.4, P.II.1.5, P.II.1.6, P.II.1.7, P.I.2.1, P.I.2.3, P.I.2.4, P.I.2.9, P.I.2.11, P.I.2.12, P.II.2.1, P.II.2.2, P.II.2.3, P.II.2.4, P.II.2.5, P.II.2.6, P.II.2.7	*	*	*	*	*	*	*

Computer Science Standard	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Physical Education	Health Education
K-2.IC.18	SL.K.4, SL.K.5, SL.K.6, SL.1.4, SL.1.5, SL.1.6, SL.2.4, SL.2.5, SL.2.6	P.I.K.1, P.I.K.3, P.I.K.9, P.I.K.11, PI.K.12, P.II.K.1, P.II.K.2, P.II.K.3, P.II.K.4, P.II.K.5, P.I.1.3, P.I.1.9, P.I.1.11, P.I.1.12, P.II.1.1, P.II.1.2, P.II.1.3, P.II.1.4, P.II.1.5, P.II.1.6, P.II.1.7, P.I.2.1, P.I.2.3, P.I.2.4, P.I.2.9, P.I.2.11, P.I.2.12, P.II.2.1, P.II.2.2, P.II.2.3, P.II.2.4, P.II.2.5, P.II.2.6, P.II.2.7	K.6.3, 1.4, 2.1, 2.4	*	*	*	*	*	*

Computer Science Standard	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Physical Education	Health Education
K-2.IC.19	W.K.5, W.K.5, W.1.5, W.1.6, W.2.5, W.2.6	PI.K.2, PI.K.10, PI.K.12, PII.K.1, PII.K.2, PII.K.3, PII.K.4, PII.K.5, PII.K.6, PI.1.2, PI.1.10, PI.1.12, PII.1.1, PII.1.2, PII.1.3, PII.1.4, II.1.5, PII.1.6, PII.1.7, PI.2.2, PI.2.4, PI.2.10, PI.2.12, PII.2.1, PII.2.3, PII.2.4, PII.2.5, PII.2.6, PII.2.7	K.1.1, 1.1.2	*	*	*	2.3.1.A	2.5.2, 2.5.3, 2.5.4	K.1.1.S, K.1.6.S, K.1.7.S, K.4.2.M, K.7.2.M, K.8.1.M, 1.1.6.S, 1.8.1.S, 2.1.10.M

Computer Science Standard	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Physical Education	Health Education
K-2.IC.20	SL.K.4, SL.K.5, SL.K.6, SL.1.4, SL.1.5, SL.1.6, SL.2.4, SL.2.5, SL.2.6	P.I.K.1, P.I.K.3, P.I.K.9, P.I.K.11, P.I.K.12, P.II.K.1, P.II.K.2, P.II.K.3, P.II.K.4, P.II.K.5, P.I.1.3, P.I.1.9, P.I.1.11, P.I.1.12, P.II.1.1, P.II.1.2, P.II.1.3, P.II.1.4, P.II.1.5, P.II.1.6, P.II.1.7, P.I.2.1, P.I.2.3, P.I.2.4, P.I.2.9, P.I.2.11, P.I.2.12, P.II.2.1, P.II.2.2, P.II.2.3, P.II.2.4, P.II.2.5, P.II.2.6, P.II.2.7	*	*	*	*	1.1.3.F, 2.3.1.E	*	*

Note: Cross-disciplinary references from the interdisciplinary examples listed in the standards document may or may not be included on this table, as the connections listed above are meant to be general connections rather than aligned to specific examples.

Grade levels 3–5

Key: Asterisk symbol (*) = Not found.

Computer Science Standard	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Physical Education	Health Education
3-5.CS.1	SL.3.4, SL.4.4, SL.5.4	PI.3.9, PI.3.12, PI.4.9, PI.4.12, PI.5.9, PI.5.12	*	*	*	*	*	*	*
3-5.CS.2	*	*	*	*	*	*	*	*	*
3-5.CS.3	*	*	*	*	*	*	*	*	*
3-5.NI.4	*	*	*	*	*	*	*	*	*
3-5.NI.5	SL.3.4, SL.4.4, SL.5.4	PI.3.9, PI.3.12, PI.4.9, PI.4.12, PI.5.9, PI.5.12	*	*	*	*	3.3.1.B, 4.3.1.B, 5.3.1.C, 5.3.1.E, 5.3.1.F	*	3.1.4.M, 4.1.1.S
3-5.NI.6	*	*	*	*	*	4-PS4-3	*	*	*

Computer Science Standard	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Physical Education	Health Education
3-5.DA.7	W.3.6, W.4.6, W.5.6, SL.3.4, SL.4.4, SL.5.4	PI.3.9, PI.3.12, PI.4.9, PI.4.12, PI.5.9, PI.5.12	*	*	*	*	*	*	*
3-5.DA.8	*	*	*	3.MD.3, 4.MD.4, 5.MD.2	*	3-PS2-2, 3-LS3-1, 3-LS4-1, 3-ESS2-1, 4-ESS2-1, 4-ESS2-2, 5-PS1-2, 5-ESS1-2, 5-ESS2-2	*	*	*
3-5.DA.9	*	*	*	3.MD.3, 4.MD.4, 5.MD.2	*	3-PS2-2, 3-LS3-1, 3-LS4-1, 3-ESS2-1, 4-ESS2-1, 4-ESS2-2, 5-PS1-2, 5-ESS1-2, 5-ESS2-2	*	*	*

Computer Science Standard	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Physical Education	Health Education
3-5.AP.10	*	*	*	*	*	*	*	*	*
3-5.AP.11	*	*	*	*	*	*	*	*	*
3-5.AP.12	*	*	*	*	*	*	*	*	*
3-5.AP.13	*	*	*	*	*	*	*	*	*
3-5.AP.14	*	*	*	*	*	*	*	*	*
3-5.AP.15	*	*	*	*	*	3-5-ETS1-1, 3-5-ETS1-2	*	*	*
3-5.AP.16	*	*	*	*	*	*	3.1.4.A, 5.3.1.A, 5.3.1.B, 5.3.1.C	*	*
3-5.AP.17	*	*	*	*	*	3-5-ETS1-3	*	*	*
3-5.AP.18	*	*	*	*	Theatre 5.2.3, Theatre 5.5.2	*	*	3.5.6, 4.5.6, 5.5.5	*
3-5.AP.19	SL.3.4, SL.4.4, SL.5.4	PI.3.9, PI.3.12, PI.4.9, PI.4.12, PI.5.9, PI.5.12	*	*	*	*	*	*	*

Computer Science Standard	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Physical Education	Health Education
3-5.IC.20	SL.3.1, SL.4.1, SL.5.1	PI.3.1, PI.3.5, PI.4.1, PI.4.5, PI.5.1, PI.5.5	*	*	*	*	*	*	*
3-5.IC.21	W.3.7, W.4.7, W.5.7	*	*	*	*	3-5-ETS1-1, 3-5-ETS1-2, 3-5-ETS1-3	*	*	*
3-5.IC.22	*	*	*	*	*	*	*	*	*
3-5.IC.23	*	*	*	*	*	*	3.1.4.A, 5.3.1.B, 5.3.1.E	*	*

Note: Cross-disciplinary references from the interdisciplinary examples listed in the standards document may or may not be included on this table, as the connections listed above are meant to be general connections rather than aligned to specific examples.

Grade levels 6–8

Key: Asterisk symbol (*) = Not found.

Computer Science Standard	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Physical Education	Health Education
6-8.CS.1	*	*	*	7.SP.8	*	MS-ETS1-1	*	*	6.7.3.M
6-8.CS.2	*	*	*	8.F.5	*	MS-ETS1-1, MS-ETS1-3	*	*	*
6-8.CS.3	*	*	*	*	*	MS-ETS1-4	*	*	*
6-8.NI.4	*	*	*	*	*	*	*	*	*
6-8.NI.5	*	*	*	*	*	*	6.3.1.A, 6.3.1.B, 6.3.1.D	*	*
6-8.NI.6	*	*	*	*	*	*	*	*	*
6-8.DA.7	*	*	*	6.RP.3, 6.EE.9, 6.NS.1, 6.SP.4, 8.EE.5, 8.F.2, 8.F.4, 8.F.5, 8.SP.1	*	MS-ETS1-3	*	*	*
6-8.DA.8	*	*	*	7.SP.3, 8.SP.1, 8.SP.4, 7.SP.4, 8.F.4, 8.F.5, 8.SP.1, 6.SP.5, 7.RP.2, 7.SP.2, 7.SP.6	*	MS-PS1-2, MS-LS4-3, MS-ESS1-3, MS-ESS2-3, MS-ESS2-5, MS-ESS3-2, MS-ETS1-3	*	*	*

Computer Science Standard	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Physical Education	Health Education
6-8.DA.9	*	*	*	7EE.4, 7.SP.7, 8.SP.1, 8.SP.3, 8.SP.4, 8.G.9	*	MS-ETS1-4, MS-PS1-2, MS-ETS1-1, MS-ETS1-2, MS-ETS1-3	*	*	*
6-8.AP.10	*	*	7.6.7, 8.2.6, 8.5.3, 8.6.5	*	*	MS-ETS1-4	*	*	*
6-8.AP.11	*	*	*	6.EE.2, 6.EE.6, 7.EE.4, 8.EE.8, 8.F.1	*	*	*	*	*
6-8.AP.12	*	*	*	7.SP.8, 8.F.4, 8.SP.4,	*	MS-ETS1-4, MS-PS1-6	*	*	*
6-8.AP.13	*	*	7.6.4, 7.8.5, 7.9, 8.2.7, 8.3.6,	6.EE.6, 7.G.2, 8.EE.8,	*	MS-ETS1-1, MS-ETS1-2	*	*	*
6-8.AP.14	*	*	*	7.EE.4	*	*	*	*	*
6-8.AP.15	SL.6.1, SL.7.1, SL.8.1	PI.6.1, PI.6.5, PI.7.1, PI.7.5, PI.8.1, PI.8.5	*	*	*	MS-ETS1-1, MS-ETS1-4	*	*	*

Computer Science Standard	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Physical Education	Health Education
6-8.AP.16	RI.6.7, W.6.8, W.7.8, W.8.8	PI.6.10, PI.7.10, PI.8.10	*	*	*	*	6.1.4.B, 7-8.3.1.D	*	*
6-8.AP.17	*	*	*	6.EE.5	*	MS-ETS1-2, MS-ETS1-3, MS-ETS1-4	*	*	*
6-8.AP.18	*	*	*	*	*	*	*	8.5.5	*
6-8.AP.19	*	*	*	*	*	*	*	*	*
6-8.IC.20	*	*	7.3.5, 8.6.1	*	*	*	6.1.3.G	*	6.3.4.A, 7-8.2.1.N, 7-8.2.4.N, 7-8.2.2.G, 7-8.4.7.S, 7-8.2.4.P, 7-8.3.3.P
6-8.IC.21	RI.6.7, RI.7.7, RI.8.7, SL.6.1, SL.7.1, SL.8.1	PI.6.1, PI.6.5, PI.7.1, PI.7.5, PI.8.1, PI.8.5	*	*	*	*	*	*	6.1.5.M, 6.7.3.M, 7-8.1.5.M
6-8.IC.22	*	*	*	*	*	*	6.4.2.C	*	*

Computer Science Standard	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Physical Education	Health Education
6-8.IC.23	*	*	*	*	*	*	6.3.1.H, 7-8.3.1.A, 7-8.3.1.B, 7-8.3.1.E, 7-8.4.2.A, 7-8.4.2.B	*	*
6-8.IC.24	*	*	*	*	*	*	7-8.3.1.A, 7-8.3.1.B, 7-8.3.1.E, 7-8.4.2.A, 7-8.4.2.B	*	6.1.7.M

Note: Cross-disciplinary references from the interdisciplinary examples listed in the standards document may or may not be included on this table, as the connections listed above are meant to be general connections rather than aligned to specific examples.

Grade levels 9–12

There is a relationship between the computer science and content standards in other subject areas, particularly in the Earth and Human Activity (ESS3) and Engineering Design (ETS1) disciplinary core ideas (DCIs) of the Next Generation Science Standards (NGSS). Although the NGSS standards are more specific to scientific domains, both sets of standards ask students to generate, evaluate, and refine computational models or simulations, and decompose problems into smaller components to facilitate designing solutions.

High school mathematics content will support student learning in computer science. High school standards in Algebra and Functions will support students in understanding, creating, and refining computational models and applying encryption techniques in computer science. High school standards in Statistics and Probability, particularly standards about interpreting data and making inferences will provide conceptual grounding for computer science standards in the Data and Analysis concept area.

The table below provides a detailed look the connections between standards in core academic disciplines and computer science.

Key: Asterisk symbol (*) = Not found.

CA CS Standard Identifier	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Health Education
9-12.CS.1	W.9-10.2 and W.11-12.2, L.9-10.6 and L.11-12.6, SL.9-10.4 and SL.11-12.4, WHST.9-10.2 and WHST.11-12.2	PI.9-12.9, PI.9-12.10, PI.9-12.12	*	*	*	*	*	*
9-12.CS.2	*	*	*	*	*	*	*	*
9-12.CS.3	*	*	*	*	*	*	*	*

CA CS Standard Identifier	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Health Education
9-12.NI.4	SL.9-10.1 and SL.11-12.1, SL.9-10.4 and SL.11-12.4	PI.9-12.1, PI.9-12.5, PI.9-12.9, PI.9-12.10, PI.9-12.12	*	*	*	*	*	*
9-12.NI.5	W.9-10.2 and W.11-12.2, L.9-10.6 and L.11-12.6, SL.9-10.4 and SL.11-12.4	PI.9-12.9, PI.9-12.10, PI.9-12.12	*	*	*	*	*	*
9-12.NI.6	*	*	*	*	*	*	*	*
9-12.NI.7	*	*	*	*	*	*	*	*
9-12.DA.8	*	*	*	*	*	*	*	*
9-12.DA.9	W.9-10.2 and W.11-12.2, L.9-10.6 and L.11-12.6, WHST.11-12.2, SL.9-10.4 and SL.11-12.4	PI.9-12.9, PI.9-12.10, PI.9-12.12	*	*	*	*	*	*
9-12.DA.10	RST.9-10.7, RST.11-12.7	*	*	S-ID.1, S-ID.5, S-ID.6	*	*	9-12 3.3	*

CA CS Standard Identifier	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Health Education
9-12.DA.11	*	*	*	S-IC.2	*	HS-PS3-1, HS-ESS3-3, HS-ESS3-6, HS-ETS1-4	9-12 3.3	*
9-12.AP.12	*	*	*	*	*	*	*	*
9-12.AP.13	*	*	*	*	*	*	*	*
9-12.AP.14	W.9-10.1 and W.11-12.1, L.9-10.6 and L.11-12.6, WHST.9-10.1 and WHST.11-12.1	PI.9-12.11, PI.9-12.12	*	*	*	*	*	*
9-12.AP.15	*	*	*	*	*	HS-ESS3-4	*	*
9-12.AP.16	*	*	*	*	*	HS-ETS1-2	*	*
9-12.AP.17	*	*	*	*	*	*	*	*
9-12.AP.18	*	*	*	*	*	*	*	*

CA CS Standard Identifier	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Health Education
9-12.AP.19	W.9-10.2 and W.11-12.2, L.9-10.6 and L.11-12.6, WHST.11-12.2, SL.9-10.4 and SL.11-12.4	PI.9-12.9, PI.9-12.10, PI.9-12.12	12.2.2	*	*	*	9-12 3.1	*
9-12.AP.20	*	*	*	*	*	*	*	*
9-12.AP.21	*	*	*	*	*	*	*	*
9-12.AP.22	W.9-10.2, L.9-10.6 and L.11-12.6, WHST.11-12.2, SL.9-10.4 and SL.11-12.4	PI.9-12.9, PI.9-12.10, PI.9-12.12	*	*	*	*	*	*
9-12.IC.23	RST.11-12.7, RST.11-12.9	*	*	*	Visual Arts 9-12.5.1	HS-ESS3-4	9-12 2.1, 9-12 2.2, 9-12 2.3, 9-12 3.2	*

CA CS Standard Identifier	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Health Education
9-12.IC.24	W.9-10.7 and W.11-12.7, W.9-10.8 and W.11-12.8, W.9-10.9 and W.11-12.9, L.9-10.6 and L.11-12.6	PI.9-12.9, PI.9-12.10, PI.9-12.12	*	*	*	*	*	*
9-12.IC.25	*	*	*	*	*	*	*	*
9-12.IC.26	RST.11-12.7, RST.11-12.9	*	*	*	Visual Arts 9-12.5.1	HS-ESS3-4	9-12 2.1, 9-12 2.2, 9-12 2.3, 9-12 3.2	*
9-12.IC.27	*	*	*	*	*	*	*	*
9-12.IC.28	W.9-10.2 and W.11-12.2, L.9-10.6 and L.11-12.6, WHST.11-12.2, SL.9-10.4 and SL.11-12.4	PI.9-12.9, PI.9-12.10, PI.9-12.12	12.2.2	*	*	*	9-12 3.1	*

CA CS Standard Identifier	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Health Education
9-12.IC.29	W.9-10.2 and W.11-12.2, L.9-10.6 and L.11-12.6, WHST.11-12.2, SL.9-10.4 and SL.11-12.4	PI.9-12.9, PI.9-12.10, PI.9-12.12	*	*	*	*	*	*
9-12.IC.30	RST.11-12.7, RST.11-12.9, WHST.9-10.1 and WHST.11-12.1	*	*	*	*	*	9-12 2.1, 9-12 2.2, 9-12 2.3, 9-12 3.1	*
9-12S.CS.1	*	*	*	*	*	*	*	*
9-12S.CS.2	W.9-10.2 and W.11-12.2, L.9-10.6 and L.11-12.6, WHST.11-12.2, SL.9-10.4 and SL.11-12.4	PI.9-12.9, PI.9-12.10, PI.9-12.12	*	*	*	*	*	*
9-12S.NI.3	*	*	*	*	*	*	*	*

CA CS Standard Identifier	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Health Education
9-12S.NI.4	W.9-10.2 and W.11-12.2, L.9-10.6 and L.11-12.6, WHST.11-12.2, SL.9-10.4 and SL.11-12.4	PI.9-12.9, PI.9-12.10, PI.9-12.12	*	*	*	*	*	*
9-12S.NI.5	*	*	*	*	*	*	9-12 3.1	*
9-12S.NI.6	*	*	*	*	*	*	*	*
9-12S.DA.7	*	*	*	S-IC.3	*	*	9-12 3.3	*
9-12S.DA.8	*	*	*	S-IC.1, S-IC.4, S-IC.5	*	HS-ESS3-5	9-12 3.3	*
9-12S.DA.9	RST.11-12.7, RST.11-12.9, WHST.9-10.1 and WHST.11-12.1	*	*	S-IC.2, S-IC.5	*	HS-LS2-1, HS-ESS1-4	9-12 2.1, 9-12 2.2, 9-12 2.3, 9-12 3.2	*
9-12S.AP.10	W.9-10.2 and W.11-12.2, L.9-10.6 and L.11-12.6	PI.9-12.10	*	*	*	*	*	*
9-12S.AP.11	*	*	*	*	*	*	*	*
9-12S.AP.12	*	*	*	*	*	*	*	*

CA CS Standard Identifier	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Health Education
9-12S.AP.13	*	*	*	*	*	*	*	*
9-12S.AP.14	*	*	*	*	*	*	*	*
9-12S.AP.15	*	*	*	*	*	*	*	*
9-12S.AP.16	*	*	*	*	*	*	*	*
9-12S.AP.17	*	*	*	*	*	*	*	*
9-12S.AP.18	*	*	*	*	*	*	*	*
9-12S.AP.19	*	*	*	*	*	*	*	*
9-12S.AP.20	*	*	*	*	*	*	*	*
9-12S.AP.21	*	*	*	*	*	*	9-12 3.1	*
9-12S.AP.22	*	*	*	*	*	*	*	*
9-12S.AP.23	*	*	*	*	*	*	*	*
9-12S.AP.24	*	*	*	*	*	*	*	*
9-12S.AP.25	*	*	*	*	*	*	*	*
9-12S.AP.26	*	*	*	*	*	*	*	*
9-12S.IC.27	RST.11-12.7, RST.11-12.9, WHST.9-10.1 and WHST.11-12.1	*	*	*	*	HS-ESS3-3, HS-ESS3-4, HS-ESS3-6, HS-ETS1-4	9-12 2.1, 9-12 2.2, 9-12 2.3, 9-12 3.2	*

CA CS Standard Identifier	ELA and Literacy	English Language Development	History–Social Studies	Mathematics	Visual and Performing Arts	Next Generation Science	Model School Library	Health Education
9-12S.IC.28	RST.11-12.7, RST.11-12.9	*	*	*	*	*	9-12 2.1, 9-12 2.2, 9-12 2.3, 9-12 3.2	*
9-12S.IC.29	RST.11-12.7, RST.11-12.9, WHST.9-10.1 and WHST.11-12.1	*	*	*	*	*	9-12 2.1, 9-12 2.2, 9-12 2.3, 9-12 3.2	*
9-12S.IC.30	RST.11-12.8, WHST.9-10.1 and WHST.11-12.1, SL.9-10.1 and SL.11-12.1	PI.9-12.11	12.2.2	*	*	*	9-12 2.1, 9-12 2.1, 9-12 2.2, 9-12 2.3, 9-12 3.1, 9-12 3.2	*

Note: Cross-disciplinary references from the interdisciplinary examples listed in the standards document may or may not be included on this table, as the connections listed above are meant to be general connections rather than aligned to specific examples.

Career Technical Education (CTE) Connections

California's Career Technical Education (CTE) standards are separated into 15 industry sectors, with separate career pathways defined within each sector, and were adopted by the SBE on January 16, 2013. The CS practices are synergistic with the CTE standards for career ready practices and can be adopted by both academic and CTE teachers.

Both the CTE and CS standards are founded in an academic discipline and provide career preparation. Both sets of practices emphasize clear communication, collaboration, understanding of diverse viewpoints, and problem-solving skills. There are two notable differences between the CS and the CTE standards. First, the CS standards identify foundational knowledge for all students—starting at the kindergarten level—in preparation for college and career readiness. Second, while the CTE standards emphasize specific technologies and industry protocols, the CS standards strive to be technology agnostic.

There are four CTE sectors and pathways that overlap with the CS standards:

- The Information and Communication Technologies (ICT) sector
- The Game Design and Integration pathway within the Arts, Media, and Entertainment (AME) sector
- The Telecommunications pathway within the Energy, Environment, and Utilities (EEU) sector
- The Public Safety pathway within the Public Services (PS) sector

CTE Standards for Career Ready Practice

The table below provides a detailed look at the connections between the CTE and CS standards and practices.

Key: Asterisk symbol (*) = Not found.

CTE Standards for Career Ready Practice	Computer Science Practices
1. Apply appropriate technical skills and academic knowledge	Alignment listed within individual sectors.
2. Communicate clearly, effectively, and with reason	**Practice 7.2** Describe, justify, and document computational processes and solutions using appropriate terminology consistent with the intended audience and purpose.
3. Develop an education and career plan aligned with personal goals	*

CTE Standards for Career Ready Practice	Computer Science Practices
4. Apply technology to enhance productivity	**Practice 2.4** Evaluate and select technological tools that can be used to collaborate on a project. **Practice 3.1** Identify complex, interdisciplinary, real-world problems that can be solved computationally. **Practice 3.3** Evaluate whether it is appropriate and feasible to solve a problem computationally.
5. Utilize critical thinking to make sense of problems and persevere in solving them	**Practice 3.2** Decompose complex real-world problems into manageable subproblems that could integrate existing solutions or procedures. **Practice 4.1** Extract common features from a set of interrelated processes or complex phenomena. **Practice 4.3** Create modules and develop points of interaction that can apply to multiple situations and reduce complexity. **Practice 5.1** Plan the development of a computational artifact using an iterative process that includes reflection on and modification of the plan, taking into account key features, time and resource constraints, and user expectations.
6. Practice personal health and understand financial literacy	*
7. Act as a responsible citizen in the workplace and the community	**Practice 7.3** Articulate ideas responsibly by observing intellectual property rights and giving appropriate attribution.
8. Model integrity, ethical leadership, and effective management	**Practice 2.2** Create team norms, expectations, and equitable workloads to increase efficiency and effectiveness. **Practice 2.3** Solicit and incorporate feedback from, and provide constructive feedback to, team members and other stakeholders.

CTE Standards for Career Ready Practice	Computer Science Practices
9. Work productively in teams while integrating cultural and global competence	**Practice 2.1** Cultivate working relationships with individuals possessing diverse perspectives, skills, and personalities.
10. Demonstrate creativity and innovation	**Practice 5.2** Create a computational artifact for practical intent, personal expression, or to address a societal issue.
11. Employ valid and reliable research strategies	**Practice 4.4** Model phenomena and processes and simulate systems to understand and evaluate potential outcomes. **Practice 6.1** Systematically test computational artifacts by considering all scenarios and using test cases. **Practice 7.1** Select, organize, and interpret large data sets from multiple sources to support a claim.
12. Understand the environmental, social, and economic impacts of decisions	**Practice 1.1** Include the unique perspectives of others and reflect on one's own perspectives when designing and developing computational products. **Practice 1.2** Address the needs of diverse end users during the design process to produce artifacts with broad accessibility and usability. **Practice 1.3** Employ self- and peer-advocacy to address bias in interactions, product design, and development methods.

The CTE pathways also contain 11 knowledge and performance anchor standards that are common across each of the industry sectors:

1. Academics (sector-specific)
2. Communications
3. Career Planning and Management
4. Technology
5. Problem Solving and Critical Thinking
6. Health and Safety
7. Responsibility and Flexibility
8. Ethics and Legal Responsibilities
9. Leadership and Teamwork
10. Technical Knowledge and Skills
11. Demonstration and Application

All anchor standards are reinforced in the computer science content and practice standards except Academics (sector-specific), Career Planning and Management, and Health and Safety.

Information and Communication Technologies Sector

Note: Specific CTE standards are referenced in parentheses following each CS standard.

Key: Asterisk symbol (*) = Not found.

Information Support and Services Pathway Standard	Related 6–8 CS Standards	Related 9–12 Core CS Standards	Related 9–12 Specialty CS Standards
A3.0 Access and transmit information in a networked environment.	6-8.NI.4 Model the role of protocols in transmitting data across networks and the Internet (A3.1).	*	*
A5.0 Identify requirements for maintaining secure network systems.	6-8.NI.5 Explain potential security threats and security measures to mitigate those threats (A5.2, A5.3).	9-12.NI.6 Compare and contrast security measures to address various security threats (A5.2, A5.3).	9-12S.NI.5 Develop solutions to security threats (A5.2, A5.3).
A6.0 Diagnose and solve software, hardware, networking, and security problems.	6-8.CS.3 Systematically apply troubleshooting strategies to identify and resolve hardware and software problems in computing systems (A6.1, A6.2, A6.3).	9-12.CS.3 Develop guidelines that convey systematic troubleshooting strategies that others can use to identify and fix errors (A6.2, A6.3).	*
A8.0 Manage and implement information, technology, and communication projects.	6-8.CS.2 Design a project that combines hardware and software components to collect and exchange data (A8.1, A8.5).	*	*

Networking Pathway Standard	Related 6–8 CS Standards	Related 9–12 Core CS Standards	Related 9–12 Specialty CS Standards
B1.0 Identify and describe the principles of networking and the technologies, models, and protocols used in a network.	6-8.NI.4 Model the role of protocols in transmitting data across networks and the internet (B1.1, B1.3).	9-12.NI.4 Describe issues that impact network functionality (B1.1, B1.3, B1.4).	9-12S.NI.3 Examine the scalability and reliability of networks by describing the relationships between routers, switches, servers, topology, and addressing (B1.1, B1.3, B1.4, B1.5).
B2.0 Identify, describe, and implement network media and physical topologies.	*	*	9-12S.NI.3 Examine the scalability and reliability of networks by describing the relationships between routers, switches, servers, topology, and addressing (B2.3).
B4.0 Demonstrate proper network administration and management skills.	*	9-12.NI.4 Describe issues that impact network functionality (B4.1).	*
B6.0 Use and assess network communication applications and infrastructure.	*	9-12.NI.4 Describe issues that impact network functionality (B4.1).	9-12S.NI.3 Examine the scalability and reliability of networks by describing the relationships between routers, switches, servers, topology, and addressing (B2.3).

Networking Pathway Standard	Related 6–8 CS Standards	Related 9–12 Core CS Standards	Related 9–12 Specialty CS Standards
B8.0 Identify security threats to a network and describe general methods to mitigate those threats.	6-8.NI.5 Explain potential security threats and security measures to mitigate those threats (B8.1, B8.4, B8.5).	9-12.NI.6 Compare and contrast security measures to address various security threats (B8.1, B8.4, B8.5).	9-12S.NI.5 Develop solutions to security threats (B8.0).

Software and Systems Development Pathway Standard	Related 6–8 CS Standards	Related 9–12 Core CS Standards	Related 9–12 Specialty CS Standards
C1.0 Identify and apply the systems development process.	6-8.AP.18 Distribute tasks and maintain a project timeline when collaboratively developing computational artifacts (C1.4, C1.5).	9-12.AP.21 Design and develop computational artifacts working in team roles using collaborative tools (C1.4).	9-12S.AP.19 Plan and develop programs for broad audiences using a specific software life cycle process (C1.1). 9-12S.AP.25 Use version control systems, integrated development environments, and collaborative tools and practices (code documentation) while developing software within a group (C1.4, C1.5).
C2.0 Define and analyze systems and software requirements.	6-8.AP.15 Seek and incorporate feedback from team members and users to refine a solution that meets user needs (C2.3, C2.4, C2.5).	9-12.AP.18 Systematically design programs for broad audiences by incorporating feedback from users (C2.3, C2.4, C2.5).	9-12S.IC.27 Evaluate computational artifacts with regard to improving their beneficial effects and reducing harmful effects on society (C2.2).

Software and Systems Development Pathway Standard	Related 6–8 CS Standards	Related 9–12 Core CS Standards	Related 9–12 Specialty CS Standards
C2.0 Define and analyze systems and software requirements. *(continued)*	6-8.IC.21 Discuss issues of bias and accessibility in the design of existing technologies (C2.2). 6-8.IC.22 Collaborate with many contributors when creating a computational artifact (C2.3, C2.4, C2.5).	9-12.IC.24 Identify impacts of bias and equity deficit on design and implementation of computational artifacts and apply appropriate processes for evaluating issues of bias (C2.2).	
C3.0 Create effective interfaces between humans and technology.	6-8.CS.1 Design modifications to computing devices in order to improve the ways users interact with the devices (C3.2, C3.3). 6-8.CS.2 Design a project that combines hardware and software components to collect and exchange data (C3.1).	9-12.CS.1 Describe ways in which abstractions hide the underlying implementation details of computing systems to simplify user experiences (P4.1). 9-12.IC.23 Evaluate the ways computing impacts personal, ethical, social, economic, and cultural practices (C3.3). 9-12.IC.24 Identify impacts of bias and equity deficit on design and implementation of computational artifacts and apply appropriate processes for evaluating issues of bias (C3.3).	*

Software and Systems Development Pathway Standard	Related 6–8 CS Standards	Related 9–12 Core CS Standards	Related 9–12 Specialty CS Standards
C4.0 Develop software using programming languages.	6-8.DA.7 Represent data in multiple ways (C4.4). 6-8.AP.11 Create clearly named variables that store data, and perform operations on their contents (C4.4). 6-8.AP.12 Design and iteratively develop programs that combine control structures, including nested loops and compound conditionals (C4.9). 6-8.AP.19 Document programs in order to make them easier to use, read, test, and debug (C4.11).	9-12.DA.8 Translate between different representations of data abstractions of real-world phenomena, such as characters,numbers, and images (C4.4). 9-12.AP.13 Create more generalized computational solutions using collections instead of repeatedly using simple variables (C4.7). 9-12.AP.14 Justify the selection of specific control structures by identifying tradeoffs associated with implementation, readability, and performance (C4.9). 9-12.AP.15 Iteratively design and develop computational artifacts for practical intent, personal expression, or to address a societal issue by using events to initiate instructions (C4.9).	9-12S.AP.12 Implement searching and sorting algorithms to solve computational problems (C4.10). 9-12S.AP.13 Evaluate algorithms in terms of their efficiency (C4.10). 9-12S.AP.14 Compare and contrast fundamental data structures and their uses (C4.7). 9-12S.AP.26 Compare multiple programming languages and discuss how their features make them suitable for solving different types of problems (C4.1).

Software and Systems Development Pathway Standard	Related 6–8 CS Standards	Related 9–12 Core CS Standards	Related 9–12 Specialty CS Standards
C4.0 Develop software using programming languages. *(continued)*		9-12.AP.16 Decompose problems into smaller components through systematic analysis, using constructs such as procedures, modules, and/or classes (C4.8). 9-12.AP.17 Create computational artifacts using modular design (C4.9). 9-12.AP.22 Document decisions made during the design process using text, graphics, presentations, and/or demonstrations in the development of complex programs (C4.11).	
C5.0 Test, debug, and improve software development work.	6-8.AP.17 Systematically test and refine programs using a range of test cases (C5.4, C5.5, C5.6).	9-12.AP.20 Iteratively evaluate and refine a computational artifact to enhance its performance, reliability, usability, and accessibility (C5.4, C5.5, C5.6).	9-12S.AP.22 Develop and use a series of test cases to verify that a program performs according to its design specifications (C5.4, C5.5, C5.6). 9-12S.AP.24 Evaluate key qualities of a program through a process such as code review (C5.1).

Software and Systems Development Pathway Standard	Related 6–8 CS Standards	Related 9–12 Core CS Standards	Related 9–12 Specialty CS Standards
C6.0 Integrate a variety of media into development projects.	6-8.AP.16 Incorporate existing code, media, and libraries into original programs, and give attribution (C6.6).	*	*
C8.0 Develop databases.	6-8.DA.8 Collect data using computational tools and transform the data to make it more useful (C8.7).	9-12.DA.10 Create data visualizations to help others better understand real-world phenomena (C8.8).	9-12S.DA.7 Select and use data collection tools and techniques to generate data sets that reveal patterns or communicate information (C8.8).
C9.0 Develop software for a variety of devices, including robotics.	*	*	9-12S.AP.20 Develop programs for multiple computing platforms (C9.0).
C10. Develop intelligent computing.	*	*	9-12S.AP.10 Describe how artificial intelligence drives many software and physical systems (C10.2). 9-12S.AP.11 Implement an algorithm that uses artificial intelligence to overcome a simple challenge (C10.4).

Games and Simulation Pathway Standard	Related 6–8 CS Standards	Related 9–12 Core CS Standards	Related 9–12 Specialty CS Standards
D3.0 Create a working game or simulation individually or as part of a team.	6-8.AP.19 Document programs in order to make them easier to use, read, test, and debug (D3.2).	9-12.AP.22 Document decisions made during the design process using text, graphics, presentations, and/or demonstrations in the development of complex programs (D3.2, D3.4).	*
D5.0 Integrate music, sound, art, and animation as it applies to the environmental design of the game/simulation.	6-8.AP.16 Incorporate existing code, media, and libraries into original programs, and give attribution (D5.1).	*	*
D6.0 Explain the role and principles of event modeling and interface design and apply those principles in a game/simulation design and project.	6-8.DA.9 Test and analyze the effects of changing variables while using computational models (D6.1, D6.2). 6-8.AP.15 Seek and incorporate feedback from team members and users to refine a solution that meets user needs (D6.3).	9-12.DA.11 Refine computational models to better represent the relationships among different elements or data collected from a phenomenon or process (D6.1, D6.2). 9-12.AP.18 Systematically design programs for broad audiences by incorporating feedback from users (D6.3). 9-12.AP.20 Iteratively evaluate and refine a computational artifact to enhance its performance, reliability, usability, and accessibility (D6.3, D6.4).	*

Games and Simulation Pathway Standard	Related 6–8 CS Standards	Related 9–12 Core CS Standards	Related 9–12 Specialty CS Standards
D7.0 Acquire and apply appropriate programming skills for rendering a single player or multiuser game or simulation project, including program control, conditional branching, memory management, scorekeeping, timed event strategies, and implementation issues.	6-8.NI.4 Model the role of protocols in transmitting data across networks and the internet (D7.1). 6-8.NI.5 Explain potential security threats and security measures to mitigate threats (D7.1). 6-8.AP.17 Systematically test and refine programs using a range of test cases (D7.3). 6-8.AP.18 Distribute tasks and maintain a project timeline when collaboratively developing computational artifacts (D7.2). 6-8.AP.19 Document programs in order to make them easier to use, read, test, and debug (D7.3, D7.4).	9-12.NI.4 Describe issues that impact network functionality (D7.1). 9-12.NI.6 Compare and contrast security measures to address various security threats (D7.1). 9-12.AP.20 Iteratively evaluate and refine a computational artifact to enhance its performance, reliability, usability, and accessibility (D7.3). 9-12.AP.22 Document decisions made during the design process using text, graphics, presentations, and/or demonstrations in the development of complex programs (D7.2).	9-12S.NI.3 Examine the scalability and reliability of networks, by describing the relationship between routers, switches, servers, topology, and addressing (D7.1). 9-12S.NI.5 Develop solutions to security threats (D7.1). 9-12S.AP.20 Develop programs for multiple computing platforms (D7.1). 9-12.S.21 Identify and fix security issues that might compromise computer programs (D7.1). 9-12S.AP.22 Develop and use a series of test cases to verify that a program performs according to its design specifications (D7.3). 9-12S.AP.24 Evaluate key qualities of a program through a process such as a code review (D7.2).

Games and Simulation Pathway Standard	Related 6–8 CS Standards	Related 9–12 Core CS Standards	Related 9–12 Specialty CS Standards
D7.0 Acquire and apply appropriate programming skills for rendering a single player or multiuser game or simulation project, including program control, conditional branching, memory management, scorekeeping, timed event strategies, and implementation issues. *(continued)*			9-12S.AP.25 Use version control systems, integrated development environments (IDEs), and collaborative tools and practices (e.g., code documentation) while developing software within a group (D7.2).
D8.0 Acquire and apply appropriate artificial intelligence (AI) techniques used by the game development industry.	*	*	9-12S.AP.10 Describe how artificial intelligence drives many software and physical systems (D8.1). 9-12S.AP.11 Implement an algorithm that uses artificial intelligence to overcome a simple challenge (D8.2).

Other Sectors

Note: Specific CTE standards are referenced in parentheses following each CS standard.

Key: Asterisk symbol (*) = Not found.

Arts, Media, and Entertainment: Game Design and Integration Pathway Standard	Related 6–8 CS Standards	Related 9–12 Core CS Standards	Related 9–12 Specialty CS Standards
D2.0 Analyze the core tasks and challenges of video game design and explore the methods used to create and sustain player immersion.	6-8.AP.13 Decompose problems and subproblems into parts to facilitate the design, implementation, and review of programs (D2.2).	9-12.AP.16 Decompose problems into smaller components through systematic analysis, using constructs such as procedures, modules, and/or classes (D2.2).	9-12S.AP.16 Analyze a large-scale computational problem and identify generalizable patterns or problem components that can be applied to a solution (D2.2).
D3.0 Acquire and apply appropriate game programming concepts and skills to develop a playable video game.	6-8.AP.11 Create clearly named variables that store data and perform operations on their contents (D3.1). 6-8.AP.12 Design and iteratively develop programs that combine control structures, including nested loops and compound conditionals (D3.1).	9-12.DA.8 Translate between different representations of data abstractions of real-world phenomena, such as characters, numbers, and images (D3.1). 9-12.AP.13 Create more generalized computational solutions using collections instead of repeatedly using simple variables (D3.1). 9-12.AP.14 Justify the selection of specific control structures by identifying tradeoffs associated with implementation, readability, and performance (D3.1).	9-12S.AP.14 Compare and contrast fundamental data structures and their uses (D3.1).

Arts, Media, and Entertainment: Game Design and Integration Pathway Standard	Related 6–8 CS Standards	Related 9–12 Core CS Standards	Related 9–12 Specialty CS Standards
D3.0 Acquire and apply appropriate game programming concepts and skills to develop a playable video game. *(continued)*		9-12.AP.15 Iteratively design and develop computational artifacts for practical intent, personal expression, or to address a societal issue by using events to initiate instructions (D3.1).	
D4.0 Students will demonstrate mastery of game art and multimedia, including music, sound, art, and animation.	6-8.AP.16 Incorporate existing code, media, and libraries into original programs, and give attribution (D4.8). 6-8.IC.23 Compare tradeoffs associated with licenses for computational artifacts to balance the protection of the creators' rights and the ability for others to use and modify (D4.8).	9-12.AP.19 Explain the limitations of licenses that restrict use of computational artifacts when using resources such as libraries (D4.8). 9-12.IC.28 Explain the beneficial and harmful effects that intellectual property laws can have on innovation (D4.8).	9-12S.IC.30 Debate laws and regulations that impact the development and use of software (D4.8).
D5.0 Demonstrate an understanding of testing techniques used to evaluate, assess, rate, and review quality assurance of video games.	6-8.AP.17 Systematically test and refine programs using a range of test cases (D5.2). 6-8.IC.21 Discuss issues of bias and accessibility in the design of existing technologies (D5.4).	9-12.AP.20 Iteratively evaluate and refine a computational artifact to enhance its performance, reliability, usability, and accessibility (D5.2).	9-12S.AP.22 Develop and use a series of test cases to verify that a program performs according to its design specifications (D5.2).

Arts, Media, and Entertainment: Game Design and Integration Pathway Standard	Related 6–8 CS Standards	Related 9–12 Core CS Standards	Related 9–12 Specialty CS Standards
D6.0 Understand the general procedures, documentation, and requirements of large-scale game design projects. Examine and categorize the significant processes in the production of games.	6-8.AP.18 Distribute tasks and maintain a project timeline when collaboratively developing computational artifacts (D6.4). 6-8.AP.19 Document programs in order to make them easier to use, read, test, and debug (D6.7).	9-12.AP.21 Design and develop computational artifacts working in team roles using collaborative tools (D6.1, D6.4). 9-12.AP.22 Document decisions made during the design process using text, graphics, presentations, and/or demonstrations in the development of complex programs (D6.7).	9-12S.AP.25 Use version control systems, integrated development environments, and collaborative tools and practices (code documentation) while developing software within a group (D6.1, D6.4).
D8.0 Understand the impact of games and the role of play in human culture. Analyze the ethics and global impact of the game industry.	6-8.IC.20 Compare tradeoffs associated with computing technologies that affect people's everyday activities and career options (D8.4, D8.5).	9-12.IC.23 Evaluate the ways computing impacts personal, ethical, social, economic, and cultural practices (D8.3, D8.4, D8.5, D8.6).	9-12S.IC.27 Evaluate computational artifacts with regard to improving their beneficial effects and reducing harmful effects on society (D8.3, D8.4, D8.5, D8.6, D8.7).

Arts, Media, and Entertainment: Game Design and Integration Pathway Standard	Related 6–8 CS Standards	Related 9–12 Core CS Standards	Related 9–12 Specialty CS Standards
D10.0 Students will build a game that demonstrates teamwork and project management by creating a game design production plan that describes the game play, outcomes, controls, rewards, interface, and artistic style of a video game.	6-8.AP.15 Seek and incorporate feedback from team members and users to refine a solution that meets user needs (D10.2). 6-8.AP.17 Systematically test and refine programs using a range of test cases (D10.6). 6-8.AP.18 Distribute tasks and maintain a project timeline when collaboratively developing computational artifacts (D10.0). 6-8.AP.19 Document programs in order to make them easier to use, read, test, and debug (D10.1, 10.6).	9-12.AP.18 Systematically design programs for broad audiences by incorporating feedback from users (D10.2). 9-12.AP.20 Iteratively evaluate and refine a computational artifact to enhance its performance, reliability, usability, and accessibility (D10.6). 9-12.AP.21 Design and develop computational artifacts working in team roles using collaborative tools (D10.0). 9-12.AP.22 Document decisions made during the design process using text, graphics, presentations, and/or demonstrations in the development of complex programs (D10.1).	9-12S.AP.22 Develop and use a series of test cases to verify that a program performs according to its design specifications (D10.6). 9-12S.AP.25 Use version control systems, integrated development environments, and collaborative tools and practices (code documentation) while developing software within a group (D10.0).

Energy, Environment, and Utilities: Telecommunications Pathway Standard	Related 6–8 CS Standards	Related 9–12 Core CS Standards	Related 9–12 Specialty CS Standards
C1.0 Understand the basic principles and concepts that impact the telecommunications industry, including systems, concepts, and regulations.	6-8.IC.20 Compare tradeoffs associated with computing technologies that affect people's everyday activities and career options (C1.1, C1.2).	9-12.NI.5 Describe the design characteristics of the internet (C1.1).	9-12.NI.4 Explain how the characteristics of the internet influence the systems developed on it (C1.1).
C2.0 Demonstrate understanding and use of the basic and emerging technologies that impact the telecommunications industry.	6-8.NI.4 Model the role of protocols in transmitting data across networks and the internet (C2.9).	*	*
C3.0 Examine the role and functions of satellites in telecommunications.	6-8.NI.4 Model the role of protocols in transmitting data across networks and the internet (C3.3, C3.8).	9-12.NI.4 Describe issues that impact network functionality (C3.3, C3.8).	9-12S.NI.3 Examine the scalability and reliability of networks, by describing the relationship between routers, switches, servers, topology, and addressing (C3.3, C3.8).
C4.0 Research the components, interaction, and operations of wireless telecommunications systems.	6-8.NI.4 Model the role of protocols in transmitting data across networks and the internet (C4.4).	9-12.NI.4 Describe issues that impact network functionality (C4.4).	9-12S.NI.3 Examine the scalability and reliability of networks, by describing the relationship between routers, switches, servers, topology, and addressing (C4.4).

Energy, Environment, and Utilities: Telecommunications Pathway Standard	Related 6–8 CS Standards	Related 9–12 Core CS Standards	Related 9–12 Specialty CS Standards
C5.0 Research the components, interaction, and operations of fixed-wire telecommunications systems.	6-8.NI.4 Model the role of protocols in transmitting data across networks and the internet (C5.3).	9-12.NI.4 Describe issues that impact network functionality (C5.3).	9-12S.NI.3 Examine the scalability and reliability of networks, by describing the relationship between routers, switches, servers, topology, and addressing (C5.3).
C6.0 Consider privacy and security issues of the telecommunications systems.	6-8.NI.5 Explain potential security threats and security measures to mitigate those threats (C6.2). 6-8.IC.24 Compare tradeoffs between allowing information to be public and keeping information private and secure (C6.2).	9-12.NI.6 Compare and contrast security measures to address various security threats (C6.20).	9-12S.NI.3 Examine the scalability and reliability of networks by describing the relationships between routers, switches, servers, topology, and addressing (C6.1).

Public Services Public Safety Pathway Standard	Related 6–8 CS Standards	Related 9–12 Core CS Standards	Related 9–12 Specialty CS Standards
A8.0 Demonstrate an understanding of the functions and career opportunities within the US Department of Homeland security (DHS).	6-8.NI.5 Explain potential security threats and security measures to mitigate threats (A8.5). 6-8.NI.6 Apply multiple methods of information protection to model the secure transmission of information (A8.5).	9-12.NI.6 Compare and contrast security measures to address various security threats (A8.4). 9-12.NI.7 Compare and contrast cryptographic techniques to model the secure transmission of information (A8.4).	9-12S.NI.5 Develop solutions to security threats (A8.6). 9-12S.NI.6 Analyze cryptographic techniques to model the secure transmission of information (A8.6).

Health Science and Medical Technology Biotechnology Pathway Standard	Related 6–8 CS Standards	Related 9–12 Core CS Standards	Related 9–12 Specialty CS Standards
A5.0 Integrate computer skills into program components.	6-8.DA.8 Collect data using computational tools and transform the data to make it more useful (A5.2).	9-12.DA.10 Create data visualizations to help others better understand real-world phenomena (A5.2).	9-12S.DA.7 Select and use data collection tools and techniques to generate data sets (A5.2).

Connections to Postsecondary Education

California State University/University of California Freshman Minimum Admission Requirements

The California State University (CSU) and University of California (UC) systems currently accept some computer science courses to fulfill freshman minimum admission requirements in category "c" (mathematics), "d" (laboratory science), or "g" (college-preparatory elective) (Alliance for California Computing Education for Students and Schools 2019). Alignments between the California computer science standards and the California mathematics and science standards and practices are described in detail in Section IV: Interdisciplinary Connections.

For a computer science course to meet the requirements for category "c", it must be designed to give students five core competencies:

1. A view that mathematics is not just a collection of definitions, algorithms and/or theorems to memorize and apply, but rather is a coherent and tightly organized body of knowledge that provides a way to think about and understand a broad array of phenomena.
2. A proclivity to put time and thought into using mathematics to grasp and solve unfamiliar problems.
3. A view that mathematics models reality and students should have the capacity to use mathematical models to guide their understanding of the world around us.
4. An awareness of special goals of mathematics, such as clarity and brevity, parsimony, universality and objectivity.
5. Confidence and fluency in handling formulas and computational algorithms: understanding their motivation and design, predicting approximate outcomes and computing them—mentally, on paper or with technology, as appropriate.

For more information on category "c" course requirements, see https://hs-articulation.ucop.edu/guide/a-g-subject-requirements/c-mathematics/.

For a computer science course to meet the requirements for category "d", it must align with the eight practices of science and engineering identified in the California Next Generation Science Standards. For more information on category "d" course requirements, see https://hs-articulation.ucop.edu/guide/a-g-subject-requirements/d-science/.

Students and parents can search the University of California's "a-g" approved course list to determine if a high school's computer science course satisfies any of the minimum admission requirements at ~~https://www.ucop.edu/agguide/~~. [No longer valid] School administrators can submit computer science courses for approval through the University of California's A-G Course Management Portal at https://hs-articulation.ucop.edu/agcmp/login#/.

Advanced Placement (AP)

Students who receive a score of 3 or higher on any Advanced Placement (AP) computer science exam can receive credit (University of California Admissions n.d.). Currently, there are two AP computer science exams.

AP Computer Science Principles (CSP) is a newer course that covers the fundamental principles of computer science. It is equivalent to a first-semester introductory college computer science course. AP CSP emphasizes computational thinking skills and computer science practices similar to those in the California computer science standards. The course covers seven "Big Ideas" in computing: creativity, abstraction, data and information, algorithms, programming, the internet, and global impact. There is significant overlap between the learning objectives of AP CSP and the California computer science standards for 9–12 (core and specialty). The California computer science standards contain a few additional standards that cover advanced topics in networking and computer programming. For more information about AP CSP, please visit the College Board website at https://apstudents.collegeboard.org/courses/ap-computer-science-principles.

AP Computer Science A (CS A) is a more traditional course that covers the fundamentals of Java programming. It is an introductory course for all students interested in the fundamentals of programming, not just those who intend to major in computer science in college. The topic outline for AP CS A contains six major topics: object-oriented program design, program implementation, program analysis, standard data structures, standard algorithms, and computing in context. The AP CS A course outline does not include topics that are included in the Networks and the Internet and Data and Analysis concept areas of the California computer science standards. In addition, practices like inclusion, collaboration, and communicating about computing are de-emphasized in AP CS A. For more information about AP CS A, please visit the College Board website at https://apstudent.collegeboard.org/apcourse/ap-computer-science-a/course-details.

Neither the AP CSP course nor the AP CS A course cover all the 9–12 core CS standards. The table below describes the alignment between the content of the AP computer science courses and the CA CS standards for 9–12. Both AP courses contain more specific learning goals than those outlined here; this table provides a general overview of areas where there is the greatest overlap.

Key: Asterisk symbol (*) = Not found.

CA CS Standard Identifier	**CA CS Standard**	**AP CSP Learning Objectives (alignments to Essential Knowledge are given by the letters in parentheses)**	**AP CS A Topic Outline**
9-12.CS.1	Describe ways in which abstractions hide the underlying implementation details of computing systems to simplify user experiences.	LO 2.1.1 Describe the variety of abstractions used to represent data (A, B, C, D, E, F, G). LO 2.2.3 Identify multiple levels of abstractions that are used when writing programs (A, B, C, D, E, F, G, H, I, J, K). LO 6.1.1 Explain the abstractions in the internet and how the internet functions (A, B, C, D, E, F, G, H, I).	IV C. Classes
9-12.CS.2	Compare levels of abstraction and interactions between application software, system software, and hardware.	LO 2.2.2 Use multiple levels of abstraction to write programs (A, B). LO 2.2.3 Identify multiple levels of abstractions that are used when writing programs (B, C, D, E, F, G, H, I, J, K).	*
9-12.CS.3	Develop guidelines that convey systematic troubleshooting strategies that others can use to identify and fix errors.	LO 5.4.1 Evaluate the correctness of a program (E).	III. B. Debugging
9-12.NI.4	Describe issues that impact network functionality.	LO 6.3.1 Identify existing cybersecurity concerns and potential options to address these issues with the internet and the systems built on it (A, B, C, D, E, F).	*

CA CS Standard Identifier	**CA CS Standard**	**AP CSP Learning Objectives (alignments to Essential Knowledge are given by the letters in parentheses)**	**AP CS A Topic Outline**
9-12.NI.5	Describe the design characteristics of the internet.	LO 6.1.1 Explain the abstractions in the internet and how the internet functions (A, B, C, D, F). LO 6.2.1 Explain the characteristics of the internet and the systems built on it (A, D).	*
9-12.NI.6	Compare and contrast security measures to address various security threats.	LO 6.3.1 Identify existing cybersecurity concerns and potential options to address these issues with the internet and the systems built on it (C, D, E, F, G, H).	VI. D. Social and ethical ramifications of computer use
9-12.NI.7	Compare and contrast cryptographic techniques to model the secure transmission of information.	LO 6.3.1 Identify existing cybersecurity concerns and potential options to address these issues with the internet and the systems built on it (H, I, J, K, L).	*
9-12.DA.8	Translate between different representations of data abstractions of real-world phenomena, such as characters, numbers, and images.	LO 2.1.1 Describe the variety of abstractions used to represent data (A, B, C). LO 2.1.2 Explain how binary sequences are used to represent digital data (D, E, F).	*
9-12.DA.9	Describe tradeoffs associated with how data elements are organized and stored.	LO 3.1.1 Find patterns and test hypotheses about digitally processed information to gain insight and knowledge (B, C). LO 3.3.1 Analyze how data representation, storage, security, and transmission of data involve computational manipulation of information (C, G, H).	V. A. Operations on data structures

CA CS Standard Identifier	**CA CS Standard**	**AP CSP Learning Objectives (alignments to Essential Knowledge are given by the letters in parentheses)**	**AP CS A Topic Outline**
9-12.DA.10	Create data visualizations to help others better understand real-world phenomena.	LO 3.1.3 Explain the insight and knowledge gained from digitally processed data by using appropriate visualizations, notations, and precise language (A, B, C, D, E).	*
9-12.DA.11	Refine computational models to better represent the relationships among different elements of data collected from a phenomenon or process.	LO 2.3.1 Use models and simulations to represent phenomena (D). LO 2.3.2 Use models and simulations to formulate, refine, and test hypotheses (A).	*
9-12.AP.12	Design algorithms to solve computational problems using a combination of original and existing algorithms.	LO 4.1.1 Develop and algorithm for implementation in a program (E, F, G, H).	V. A. Operations on data structures V. B. Searching V. C. Sorting
9-12.AP.13	Create more generalized computational solutions using collections instead of repeatedly using simple variables.	LO 5.5.1 Employ appropriate mathematical and logical concepts in programming (H, I, J).	V. A. Operations on data structures V. B. Searching V. C. Sorting IV. D. Lists IV. E. Arrays

CA CS Standard Identifier	**CA CS Standard**	**AP CSP Learning Objectives (alignments to Essential Knowledge are given by the letters in parentheses)**	**AP CS A Topic Outline**
9-12.AP.14	Justify the selection of specific control structures by identifying tradeoffs associated with implementation, readability, and performance.	LO 5.4.1 Evaluate the correctness of a program (I, J). LO 5.5.1 Employ appropriate mathematical and logical concepts in programming (D, E, F, G, I).	II. B. Programming constructs
9-12.AP.15	Iteratively design and develop computational artifacts for practical intent, personal expression, or to address a societal issue by using events to initiate instructions.	LO 1.2.1 Create a computational artifact for creative expression (E). LO 5.1.1 Develop a program for creative expression, to satisfy personal curiosity, or to create new knowledge (A, B). LO 5.1.2 Develop a correct program to solve problems (A).	II. A. Implementation techniques
9-12.AP.16	Decompose problems into smaller subproblems through systematic analysis, using constructs such as procedures, modules, and/or classes.	LO 2.2.1 Develop an abstraction when writing a program or creating other computational artifacts (A, B, C). LO 2.2.2 Use multiple levels of abstraction to write programs (A–B). LO 5.3.1 Use abstraction to manage complexity in programs (A–G, L).	I. A. Program and class design II. A. Implementation techniques IV. C. Classes

CA CS Standard Identifier	**CA CS Standard**	**AP CSP Learning Objectives (alignments to Essential Knowledge are given by the letters in parentheses)**	**AP CS A Topic Outline**
9-12.AP.17	Create computational artifacts using modular design.	LO 1.2.1 Create a computational artifact for creative expression (B). LO 1.2.3 Create a new computational artifact by combining or modifying existing artifacts (A). LO 5.1.2 Develop a correct program to solve problems (B, C).	I. A. Program and class design II. A. Implementation techniques
9-12.AP.18	Systematically design programs for broad audiences by incorporating feedback from users.	LO 1.2.5 Analyze the correctness, usability, functionality, and suitability of computational artifacts (A–D). LO 5.1.2 Develop a correct program to solve problems (G, H).	*
9-12.AP.19	Explain the limitations of licenses that restrict use of computational artifacts when using resources such as libraries.	LO 7.3.1 Analyze the beneficial and harmful impacts of computing (N–Q).	VI. C. Legal issues and intellectual property
9-12.AP.20	Iteratively evaluate and refine a computational artifact to enhance its performance, reliability, usability, and accessibility.	LO 1.2.5 Analyze the correctness, usability, functionality, and suitability of computational artifacts (A–D). LO 5.1.2 Develop a correct program to solve problems (A, H).	III. A. Testing III. B. Debugging
9-12.AP.21	Design and develop computational artifacts working in team roles using collaborative tools.	LO 1.2.4 Collaborate in the creation of computational artifacts (A, B). LO 5.1.3 Collaborate to develop a program (C).	*

CA CS Standard Identifier	**CA CS Standard**	**AP CSP Learning Objectives (alignments to Essential Knowledge are given by the letters in parentheses)**	**AP CS A Topic Outline**
9-12.AP.22	Document decisions made during the design process using text, graphics, presentations, and/or demonstrations in the development of complex programs.	LO 5.1.2 Develop a correct program to solve problems (D–F).	*
9-12.IC.23	Evaluate the ways computing impacts personal, ethical, social, economic, and cultural practices.	LO 7.3.1 Analyze the beneficial and harmful impacts of computing (A–Q).	VI. D. Social and ethical ramifications of computer use
9-12.IC.24	Identify impacts of bias and equity deficit on design and implementation of computational artifacts and apply appropriate processes for evaluating issues of bias.	LO 7.4.1 Explain the connections between computing and real-world contexts, including economic, social, and cultural contexts (A, C, D).	VI. D. Social and ethical ramifications of computer use
9-12.IC.25	Demonstrate ways a given algorithm applies to problems across disciplines.	LO 5.2.1 Explain how programs implement algorithms (J).	*
9-12.IC.27	Use collaboration tools and methods to increase connectivity with people of different cultures and careers.	LO 1.2.4 Collaborate in the creation of computational artifacts (C, E). LO 5.1.3 Collaborate to develop a program (B, C, F).	*
9-12.IC.28	Explain the beneficial and harmful effects that intellectual property laws can have on innovation.	LO 7.3.1 Analyze the beneficial and harmful effects of computing (N, O, P, Q).	VI. C. Legal issues and intellectual property

CA CS Standard Identifier	**CA CS Standard**	**AP CSP Learning Objectives (alignments to Essential Knowledge are given by the letters in parentheses)**	**AP CS A Topic Outline**
9-12.IC.29	Explain the privacy concerns related to the collection and generation of data through automated processes.	LO 3.2.2 Determine how large data sets impact the use of computational processes to discover information and knowledge (D). LO 3.3.1 Analyze how data representation, storage, security, and transmission of data involve computational manipulation of information (A, B, F).	VI. B. Privacy
9-12.IC.30	Evaluate the social and economic implications of privacy in the context of safety, law, or ethics.	LO 7.3.1 Analyze the beneficial and harmful impacts of computing (G, H).	VI. B. Privacy VI. D. Social and ethical ramifications of computer use
9-12S.CS.1	Illustrate ways computing systems implement logic, input, and output through hardware components.	LO 2.2.3 Identify multiple levels of abstractions that are used when writing programs (E–I).	*
9-12S.CS.2	Categorize and describe the different functions of operating system software.	*	*
9-12S.NI.3	Examine the scalability and reliability of networks, by describing the relationship between routers, switches, servers, topology, and addressing.	LO 6.2.2 Explain how the characteristics of the internet influence the systems built on it (A, B).	*

CA CS Standard Identifier	**CA CS Standard**	**AP CSP Learning Objectives (alignments to Essential Knowledge are given by the letters in parentheses)**	**AP CS A Topic Outline**
9-12S.NI.4	Explain how the characteristics of the internet influence the systems developed on it.	LO 6.1.1 Explain the abstractions in the internet and how the internet functions (A–D, F). LO 6.2.1 Explain the characteristics of the internet and the systems built on it (A, D). LO 6.2.2 Explain how the characteristics of the internet influence the systems built on it (A–K).	*
9-12S.NI.5	Develop solutions to security threats.	LO 6.3.1 Identify existing cybersecurity concerns and potential options to address these issues with the internet and the systems built on it (C, G).	*
9-12S.NI.6	Analyze cryptographic techniques to model the secure transmission of information.	LO 6.3.1 Identify existing cybersecurity concerns and potential options to address these issues with the internet and the systems built on it (I, K, L).	*
9-12S.DA.7	Select and use data collection tools and techniques to generate data sets.	LO 3.2.2 Determine how large data sets impact the use of computational processes to discover information and knowledge (B, C).	*
9-12S.DA.8	Use data analysis tools and techniques to identify patterns in data representing complex systems.	LO 3.2.1 Extract information from data to discover and explain connections or trends (C–F).	*
9-12S.DA.9	Evaluate the ability of models and simulations to test and support the refinement of hypotheses.	LO 2.3.2 Use models and simulations to formulate, refine, and test hypotheses (A–H).	*

CA CS Standard Identifier	**CA CS Standard**	**AP CSP Learning Objectives (alignments to Essential Knowledge are given by the letters in parentheses)**	**AP CS A Topic Outline**
9-12S.AP.10	Describe how artificial intelligence drives many software and physical systems.	*	*
9-12S.AP.11	Implement an algorithm that uses artificial intelligence to overcome a simple challenge.	*	*
9-12S.AP.12	Implement searching and sorting algorithms to solve computational problems.	LO 4.1.1 Develop and algorithm for implementation in a program (A–C). LO 4.1.2 Express an algorithm in a language (G). LO 4.2.4 Evaluate algorithms analytically and empirically for efficiency, correctness, and clarity (H).	V. B. Searching V. C. Sorting
9-12S.AP.13	Evaluate algorithms in terms of their efficiency.	LO 4.2.1 Explain the difference between algorithms that run in a reasonable time and those that do not run in a reasonable time (B, C). LO 4.2.4 Evaluate algorithms analytically and empirically for efficiency, correctness, and clarity (D, G).	III. E. Algorithm analysis
9-12S.AP.14	Compare and contrast fundamental data structures and their uses.	LO 5.3.1 Use abstraction to manage complexity in programs (K, L).	IV. D. Lists IV. E. Arrays
9-12S.AP.15	Demonstrate the flow of execution of a recursive algorithm.	*	II. B. Programming constructs

CA CS Standard Identifier	**CA CS Standard**	**AP CSP Learning Objectives (alignments to Essential Knowledge are given by the letters in parentheses)**	**AP CS A Topic Outline**
9-12S.AP.16	Analyze a large-scale computational problem and identify generalizable patterns or problem components that can be applied to a solution.	LO 2.2.1 Develop an abstraction when writing a program or creating other computational artifacts (A, B, C).	*
9-12S.AP.17	Construct solutions to problems using student-created components, such as procedures, modules, and/or objects.	LO 5.3.1 Use abstraction to manage complexity in programs (A–G, L).	I. A. Program and class design
9-12S.AP.18	Demonstrate code reuse by creating programming solutions using libraries and APIs.	LO 5.3.1 Use abstraction to manage complexity in programs (M, N, O).	I. A. Program and class design
9-12S.AP.19	Plan and develop programs for broad audiences using a specific software life cycle process.	LO 5.1.2 Develop a correct program to solve problems (A).	*
9-12S.AP.20	Develop programs for multiple computing platforms.	*	*
9-12S.AP.21	Identify and fix security issues that might compromise computer programs.	*	*
9-12S.AP.22	Develop and use a series of test cases to verify that a program performs according to its design specifications.	LO 1.2.5 Analyze the correctness, usability, functionality, and suitability of computational artifacts (A, B).	III A. Testing

CA CS Standard Identifier	**CA CS Standard**	**AP CSP Learning Objectives (alignments to Essential Knowledge are given by the letters in parentheses)**	**AP CS A Topic Outline**
9-12S.AP.23	Modify an existing program to add additional functionality and discuss intended and unintended implications.	LO 5.1.2 Develop a correct program to solve problems (B, C).	*
9-12S.AP.24	Evaluate key qualities of a program through a process such as a code review.	LO 5.4.1 Evaluate the correctness of a program (A, B, C).	*
9-12S.AP.25	Use version control systems, integrated development environments (IDEs), and collaborative tools and practices (e.g., code documentation) while developing software within a group.	LO 1.2.1 Create a computational artifact for creative expression (C). LO 1.2.4 Collaborate in the creation of computational artifacts (A, B). LO 5.1.3 Collaborate to develop a program (C, D, E, F).	*
9-12S.AP.26	Compare multiple programming languages and discuss how their features make them suitable for solving different types of problems.	LO 2.2.3 Identify multiple levels of abstractions that are used when writing programs (A, B).	*
9-12S.IC.27	Evaluate computational artifacts with regard to improving their beneficial effects and reducing harmful effects on society.	LO 5.1.1 Develop a program for creative expression, to satisfy personal curiosity, or to create new knowledge (E, F).	VI. D. Social and ethical ramifications of computer use

CA CS Standard Identifier	**CA CS Standard**	**AP CSP Learning Objectives (alignments to Essential Knowledge are given by the letters in parentheses)**	**AP CS A Topic Outline**
9-12S.IC.28	Evaluate how computational innovations that have revolutionized aspects of our culture might evolve.	LO 7.1.1 Explain how computing innovations affect communication, interaction, and cognition (A–O).	VI. D. Social and ethical ramifications of computer use
9-12S.IC.29	Evaluate the impact of equity, access, and influence on the distribution of computing resources in a global society.	LO 7.4.1 Explain the connections between computing and real-world contexts, including economic, social, and cultural contexts (A–E).	VI. D. Social and ethical ramifications of computer use
9-12S.IC.30	Debate laws and regulations that impact the development and use of software.	LO 7.3.1 Analyze the beneficial and harmful impacts of computing (N–Q).	VI. C. Legal issues and intellectual property

International Baccalaureate

Computer science is considered an experimental science in the International Baccalaureate (IB) Diploma Programme curriculum, alongside biology, chemistry, design technology, physics, and environmental systems and societies (International Baccalaureate 2019). Computer science is offered as a standard level (SL) course, intended for students with no previous experience with computer science, and as a higher level (HL) course, intended for students with some prior exposure to programming. Both the SL and the HL course emphasize computational thinking and include the following four topics:

1. System fundamentals
2. Computer organization
3. Networks
4. Computational thinking, problem-solving, and programming

Students at SL and HL will also choose to study one of the following options:

A. Databases

B. Modeling and simulation

C. Web science

D. Object-oriented programming

The HL course includes three additional topics as more in-depth content for the selected option:

1. Abstract data structures
2. Resource management
3. Control

There is significant overlap between the IB Diploma Programme computer science courses and the California computer science standards, but neither the SL nor the HL course addresses the full set of California computer science standards. The IB courses include some objectives that are covered in the K–8 computer science standards and explore topics, such as system fundamentals and computer organization, in much greater depth. The IB syllabus changes frequently—teachers should attend an IB workshop to obtain detailed curriculum information. A general overview of the degree of alignment between the IB course topics and the CA CS standards is provided in the table below.

Key: Asterisk symbol (*) = Not found.

IB CS Topic	Degree of Alignment with CA CS Standards	Example 9–12 CA CS Standards Aligned to CS Topic
Topic 1: System fundamentals	Low or none	*
Topic 2: Computer organization	Moderate	9-12S.CS.1, 9-12.CS.2
Topic 3: Networks	Moderate	9-12.NI.4, 9-12S.NI.3
Topic 4: Computational thinking, problem-solving, and programming	Moderate	9-12.AP.12, 9-12S.AP.10, 9-12S.AP.11, 9-12S.AP.12, 9-12S.AP.13
Topic 5: Abstract data structures	High	9-12.AP.13, 9-12S.AP.14, 9-12S.AP.15
Topic 6: Resource management	Low or none	*
Topic 7: Control	Low or none	*
Option A: Databases	Low or none	*
Option B: Modeling and simulation	Moderate	9-12.DA.10, 9-12S.DA.7, 9-12S.DA.8
Option C: Web science	Moderate	9-12.NI.5, 9-12S.NI.4
Option D: Object-oriented programming	High	9-12.AP.16, 9-12.AP.17, 9-12S.AP.16, 9-12S.AP.17, 9-12S.AP.18
Case study	Low or none	*

Glossary

The glossary includes definitions of terms used in the statements in the framework. These terms are defined for readers of the framework and are not necessarily intended to be the definitions or terms that are seen by students. Definitions are adapted from the K–12 Computer Science Framework glossary at https://k12cs.org/glossary.

Term	Definition
abstraction	*(process)*: The process of hiding detail associated with an idea or phenomenon to reduce complexity and facilitate communication. By hiding some details, abstraction allows one to focus on relevant details without including background details or explanations. *(product)*: A representation of an idea or phenomenon that hides details irrelevant to the question at hand. Abstractions can be associated with levels, where abstractions that are used to define other abstractions are commonly referred to as lower level abstractions, and abstractions built upon lower level abstractions are considered higher level abstractions.
accessibility	The extent to which products, devices, services, or environments can be used by people who experience disabilities. Accessibility standards that are generally accepted by professional groups include the Web Content Accessibility Guidelines (WCAG) 2.0, Accessible Rich Internet Applications (ARIA) standards, Section 508 Standards, and Section 225 Guidelines for Telecommunications Equipment (Wikipedia n.d.).
algorithm	A sequence of instructions designed to complete a specific task.
analog	The defining characteristic of analog data is that it is represented in a continuous, physical way. Whereas digital data is represented in discrete binary form, analog data uses continuous representations, such as the surface grooves on a vinyl record, the magnetic tape of a VCR cassette, or electromagnetic waves sent to an analog speaker (Techopedia 2019).
app	An app is computer software, or a program, most commonly a small, specific one used for mobile devices. The term app originally referred to any mobile or desktop application, but as more app stores have emerged to sell mobile apps to smartphone and tablet users, the term has evolved to refer to small programs that can be downloaded and installed all at once. Also known as a *mobile application* (Techopedia 2019).

Term	Definition
Application Programming Interface (API)	A set of commands, procedures, protocols, and objects that programmers can use to create or interact with software (Tech Terms 2019). A good API makes it easier to develop a computer program by providing building blocks for common operations, which are then put together by the programmer. Documentation for the API is usually provided to facilitate usage.
argument	A value passed to a procedure and stored in a parameter. *See also* parameter.
array	Arrays are commonly used in computer programs to organize data in a sequence, where individual data values can be easily accessed and/or modified. Each element in an array is identified by an index.
artifact	*See* computational artifact.
audience	Expected end users of a computational artifact or system. *See also* end user.
authentication	The verification of the identity of a person or process (FOLDOC 2019).
automate	To run or operate (something, such as a factory or system) by using machines, computers, etc., instead of people to do the work (Merriam-Webster 2019).
automation	The creation of technology and its application to perform tasks that were previously performed by humans (Techopedia 2019).
bandwidth	The maximum data transfer rate of a network or internet connection. It measures how much data can be sent over a specific connection in a given amount of time (Tech Terms 2019).
binary	A numeric system that only uses two digits: 0 and 1. Computers operate in binary, meaning data is represented and calculations are performed using only zeros and ones. Binary is also called “base 2” (Tech Terms 2019).
biometric verification	An authentication process used to confirm a claimed identity through uniquely identifiable biological traits, such as fingerprints and hand geometry. It is designed to allow a user to prove his or her identity by supplying a biometric sample and associated unique identification code in order to gain access to a secure environment (Techopedia 2019).

Term	Definition
Boolean	A type of data or expression with two possible values: true and false (FOLDOC, n.d.).
bug	An error in a software program. It may cause a program to unexpectedly quit or behave in an unintended manner (Tech Terms 2019). The process of finding and correcting errors (bugs) is called debugging. *See also* debugging.
camel case	Camel case (also "CamelCase" or "dromedary case") is a naming convention in which the first letter of each word in a compound word is capitalized. Some programming languages do not allow the use of spaces in the names of procedures, variables, or other entities. Therefore, programmers often use CamelCase to define portions of their code. For example, employeeID, employeeFirstName, employeeLastName, and employeeAddress (Tech Terms 2019).
cloud; **cloud computing**	Cloud computing is a type of internet-based computing that relies on sharing computer resources. Instead of installing software applications on local servers or devices, the applications and services are offered over the internet, from data centers all over the world. These data centers are collectively referred to as the "cloud" (Tech Terms 2019).
code	**code** *(n)*: Any set of instructions expressed in a programming language (TechTarget 2019). **code** *(v)*: To write instructions for a computer using a programming language (TechTarget 2019). See also program; programming.
collection	A set of variables used to store and process related data. The data is accessible through the use of a single variable and functionality defined for that particular collection. Examples of collections include arrays, sets, and lists.
comment	In HTML, a comment is information designers can add to the HTML for reference. Comments are not viewed by users within a browser, but rather are only visible when viewing the HTML source code. HTML comments are written as the following, where you add your own textual note between the characters: <!-- your information here --> (Webopedia 2019).

Term	Definition
complexity	The minimum amount of resources, such as memory, time, or messages, needed to solve a problem or execute an algorithm (Black 2004).
component	Computers are made up of many different parts, or components, such as a motherboard and hard drive. Each of these parts is made up of smaller parts, also called components. For example, a motherboard includes a circuit board, electrical connectors, and resistors. These and other components work together to make the motherboard function (Tech Terms 2019).
compose	To make or create by putting together parts of elements (The Free Dictionary n.d.) *See also* decompose.
compound conditional	Compound conditionals combine two or more conditions in a logical relationship, such as AND, OR, and NOT. See also conditional.
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, p. 11).
computational model; computational modeling	Computational modeling is the use of computers to simulate and study the behavior of complex systems. A computational model contains numerous variables that characterize the system being studied. Simulation is done by adjusting each of these variables alone or in combination and observing how the changes affect the outcomes. The results of model simulations help researchers make predictions about what will happen in the real system that is being studied in response to changing conditions (NIBIB 2016).

Term	Definition
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, p.3). Computational thinking plays a key role in the computer science practices described in the K–12 Computer Science Framework (2016) and encompasses the following practices: 3. Recognizing and Defining Computational Problems 4. Developing and Using Abstractions 5. Creating Computational Artifacts 6. Testing and Refining Computational Artifacts
computer	A machine or device that performs processes, calculations, and operations based on instructions provided by a software or hardware program (Techopedia 2019). *See also* computing device.
computer science	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. 2006).
computing	Any goal-oriented activity requiring, benefiting from, or creating computers. Computing is a family of disciplines that includes computer science, electrical engineering, and information systems (Tucker et al. 2006).
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.

Term	Definition
conditional	A programming language feature that determines the flow of control of a program. A conditional can appear in the form of a conditional statement (if–then), conditional expression (Boolean expression), or conditional construct (functional programming).
configuration	*(process)*: Defining the options that are provided when installing or modifying hardware and software or the process of creating the configuration (product) (TechTarget 2019). *(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 2019).
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 2019).
control	*(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.
Creative Commons license	One of several public copyright licenses that enable the free distribution of an otherwise copyrighted work. A Creative Commons (CC) license is used when an author wants to give people the right to share, use, and build upon a work that they have created. A CC license provides an author flexibility (for example, they might choose to allow only non-commercial uses of their own work) and protects the people who use or redistribute an author's work from concerns of copyright infringement as long as they abide by the conditions that are specified in the license by which the author distributes the work. (Wikipedia n.d.). *See also* license.
cryptography	Cryptography is a technique for transforming information on a computer in such a way that it becomes unreadable by anyone except authorized parties. Cryptography is useful for supporting secure communication of data across networks. Examples of cryptographic methods include hashing, symmetric encryption/decryption (private key), and asymmetric encryption/decryption (public key/private key).

Term	Definition
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 2019).
data	The raw representation of variables. Data can be collected and used for reference or analysis. Data can be digital or non-digital and can be in many forms, including numbers, text, show of hands, images, sounds, or video (Computing at School 2013).
data structure	A format for storing and accessing data within a computer program for the appropriate access and modification of the data.
data type	A classification of data that is distinguished by its attributes and the types of operations that can be performed on it. Some common data types are integer, string, Boolean (true or false), and floating-point.
debugging	The process of finding and correcting errors (bugs) in programs (quoted in Massachusetts Department of Elementary and Secondary Education 2016, 37). *See also* bug.
declare a variable	In computer programming, declaring a variable determines the name and, in some programming languages, its data type. Programmers declare variables by writing the name of the variable into code (Techopedia 2019). *See also* variable.
decompose; **decomposition**	**decompose** *(v)*: To break down into components. **decomposition** *(n)*: The act of breaking down a problem or system into components (quoted in Massachusetts Department of Elementary and Secondary Education 2016, 37).

Term	Definition
denial-of-service (DoS) attack	A cyberattack where the perpetrator seeks to make a computer system unavailable to its intended users. Denial of service is typically accomplished by flooding the computer system with excessive and/or invalid requests in an attempt to overload the system and prevent some or all legitimate requests from being fulfilled (Techopedia 2019).
design	The creation of a plan or convention for the construction of an object, system, or measurable human interaction (as in pseudocode and prototypes) (Wikipedia n.d.).
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 2019).
digital	A characteristic of electronic technology that uses binary digits 0 and 1, to generate, store, and process data.
digital citizenship	The norms of appropriate, responsible behavior with regard to the use of technology (Massachusetts Department of Elementary and Secondary Education 2016, 37).
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. (Black 2004).
encapsulation	The technique of combining data and the procedures that act on it to create a type (FOLDOC 2019). *See also* structure.
encoding	The process of converting data into a format required for information processing. In computer technology, encoding is the process of applying a specific code, such as letters, symbols, and/or numbers, to data for conversion (Techopedia 2019).
encryption	The conversion of electronic data into another form, called ciphertext, which cannot be easily understood by anyone except authorized parties (TechTarget 2019).
end user (or user)	A person for whom a hardware or software product is designed (as distinguished from the developers) (TechTarget 2019). *See also* audience.

Term	Definition
evaluation	1. Converting a programming statement into a value. 2. The process of examining the extent to which a computational artifact meets specified properties and goals (FOLDOC 2019).
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 2019).
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 2019).
firewall	A network security system that monitors and controls incoming and outgoing network traffic based on predetermined security rules. A firewall typically establishes a barrier between a trusted internal network and an untrusted outside network, such as the Internet. Firewalls can be implemented as hardware, software, or a combination of both (Webopedia 2019).
floating point number	The computing term used for representing non-integer numbers, such as numbers with decimal points.
function	*See definition for* procedure.
garbage collection	In computer science, garbage collection (GC) is a form of automatic memory management. The garbage collector, or just collector, attempts to reclaim garbage, or memory occupied by objects that are no longer in use by the program (Wikipedia n.d.)
hardware	The physical components that make up a computing system, computer, or computing device (quoted in Massachusetts Department of Elementary and Secondary Education 2016, 38).
hierarchy	An organizational structure in which items are ranked according to levels of importance (TechTarget 2019).
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 2019).

Term	Definition
identifier	The programmer-defined, unique name of a program element (such as a variable or procedure). An identifier name should indicate the meaning and usage of the element being named (Techopedia 2019).
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.
input	Data sent to a computer program.
integrated development environment (IDE)	A software system for supporting the process of writing software. Such a system may include a syntax-directed editor, graphical tools for program entry, and integrated support for compiling and running the program and relating compilation errors back to the source (FOLDOC 2019).
integrity	The overall completeness, accuracy, and consistency of data (Techopedia 2019).
internet	The global collection of computer networks and their connections, all using shared protocols to communicate (Computing at School 2013, 27).
Internet Protocol (IP) address	A unique numerical label that is used to identify and locate each device connected to the internet (Tech Terms 2019).
iterative	Involving the repeating of a process with the aim of approaching a desired goal, target, or result (quoted in Massachusetts Department of Elementary and Secondary Education 2016, 39).
library	A collection of resources used by computer programs. These resources can include message templates, pre-written code, and specifications. The distinguishing feature is that a library is organized for the purposes of being reused by independent programs or sub-programs (Wikipedia n.d.).
license	An official permission or permit to do, use, or own something (as well as the document of that permission or permit). A public license is a license by which a copyright holder can grant additional copyright permissions to any and all persons in the general public as licensees. By applying a public license to a work, copyright holders give permission for others to copy or change their work in ways that would otherwise infringe copyright law (Wikipedia n.d.). *See also* Creative Commons license.

Term	Definition
list	A data structure holding many values, possibly of different types, which is usually accessed sequentially, working from the head to the end of the tail—an "ordered list" (FOLDOC 2019).
loop	A programming structure that repeats a sequence of instructions as long as a specific condition is true (Tech Terms 2019).
Media Access Control (MAC) address	A hardware identification number that uniquely identifies each device on a network. The MAC address is manufactured into every network card, such as an ethernet card or a Wi-Fi card, and therefore cannot be changed (Tech Terms 2019).
memory	Internal storage hardware used by computers to store and access data.
model	A representation (abstraction) of some part of a problem or a system (quoted in Massachusetts Department of Elementary and Secondary Education 2016, 39). *Note: This definition differs from that used in science.*
modularity	The characteristic of a software or 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 2019).
module	Any of a number of distinct but interrelated units from which a program may be built up or into which a complex activity may be analyzed. 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).
nest(ed)	To embed one object in another object. Nesting is quite common in programming, where different logic structures sequence, selection and loop) are combined (i.e., nested in one another). For example, nested loops are loops placed within loops, and nested conditionals allow the result of one conditional to lead to another (Webopedia 2019).
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.

Term	Definition
object; **object-oriented programming (OOP)**	Object-oriented programming (OOP) refers to a type of computer programming (software design) in which programmers define data types that encapsulate both data and operations (procedures) that are applied to the data structure. In this way, the data structure becomes an object that includes both data and procedures. Programmers can create relationships between one object and another. For example, objects can inherit characteristics from other objects (Webopedia 2019).
operation	An action, resulting from a single instruction, that changes the state of data (Free Dictionary n.d.).
packet	The unit of data sent over a network (Tech Terms 2019).
pair programming	Pair programming is a software development technique in which two programmers work together using a single computer. One programmer acts as the driver by entering code, while the second programmer acts as the navigator, providing insight and feedback on the code as it is entered. The programmers switch roles on a regular basis. *See also* program; programming.
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 (quoted in Massachusetts Department of Elementary and Secondary Education 2016, 39).
phishing	The attempt to obtain sensitive information, such as usernames, passwords, and credit card details, by disguising as a trustworthy entity in an electronic communication. Phishing is typically carried out by email or instant messaging, and often directs users to enter personal information at a fake website which looks identical to the legitimate one except for the URL (Wikipedia n.d.).
physical security token	Devices used to gain access to an electronically restricted resource. The token is used in addition to, or in place of, a password. It acts like an electronic key to access something. An example of a physical security token is a wireless keycard opening a locked door (Wikipedia n.d.).
piracy	The illegal copying, distribution, or use of software (TechTarget 2019).

Term	Definition
procedure	An independent code module that fulfills some concrete task and is referenced within a larger body of program code. The fundamental role of a procedure is to offer a single point of reference for some small goal or task that the developer or programmer can trigger by invoking the procedure itself (Techopedia 2019). Note: In these standards, 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 Dictionaries 2019).
program; programming	**program** *(n)*: A set of instructions that the computer executes to achieve a particular objective (quoted in Massachusetts Department of Elementary and Secondary Education 2016, 40). **program** *(v)*: To produce a program by programming. **programming**: The craft of analyzing problems and designing, writing, testing, and maintaining programs to solve them (quoted in Massachusetts Department of Elementary and Secondary Education 2016, 40).
protocol	The special set of rules used by endpoints in a telecommunication connection when they communicate. Protocols specify interactions between the communicating entities (TechTarget 2019).
prototype	An early approximation of a final product or information system, often built for demonstration purposes (TechTarget 2019).
pseudocode	An informal high-level description of a computer program or other algorithm (Wikipedia n.d.). Pseudocode does not require strict syntax and uses natural language. The purpose of using pseudocode is that it is easier for people to understand than conventional programming language code. It is commonly used in computer program development for sketching out the structure of the program before actual coding takes place (TechTarget 2019).
public key encryption	An encryption technique that uses pairs of keys: public keys, which may be widely known, and private keys, which are known only to the owner. In a public key encryption system, any sender can encrypt a message using the public key of the receiver, but the message can only be decrypted with the receiver's private key (Wikipedia n.d.).

Term	Definition
ransomware	A type of malicious software that blocks access to a victim's computer or certain files unless a ransom is paid (Wikipedia n.d.).
recursion; recursive procedures	A powerful problem-solving approach where the problem solution is built on solutions of smaller instances of the same problem. A base case, which returns a result without referencing itself, must be defined, otherwise infinite recursion will occur. Procedures that incorporate recursion are called recursive procedures. *See also* procedure.
redundancy	A system design in which a component is duplicated, so if it fails, there will be a backup (TechTarget 2019).
reliability	An attribute of any system that consistently produces the same results given the same configuration, preferably meeting or exceeding its requirements (FOLDOC 2019).
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 and Burke 2014).
router	A device or software that determines the path that data packets travel from source to destination (TechTarget 2019).
scalability	The capability of a network or a program to handle a growing amount of work or its potential to be enlarged to accommodate that growth (Wikipedia n.d.).
scrape a web page	Scraping a web page is the process of automatically mining data or collecting information from the World Wide Web. Current web scraping solutions range from the ad-hoc, requiring human effort, to fully automated systems that are able to convert entire web sites into structured information, with limitations (Wikipedia n.d.).
security	*See* cybersecurity.
server	A computer program or device that provides services to other computer programs or devices (TechTarget 2019).
set	A data type that can store a collection of values, without any particular order, and no repeated values (Wikipedia n.d.).

Term	Definition
simulate; simulation	**simulate** *(v)*: To imitate the operation of a real-world process or system. **simulation** *(n)*: Imitation of the operation of a real-world process or system (Massachusetts Department of Elementary and Secondary Education 2016, 40).
social engineering	A non-technical method of breaking into a secured computer system. Victims of social engineering are tricked into releasing information that they do not realize will be used to attack a computer network. Phishing is a type of security attack that relies on social engineering (Techopedia 2019). *See also* phishing.
software	Programs that run on a computing system, computer, or other computing device.
spyware	Software that aims to gather information about a person or organization without their knowledge, that may send such information to another entity without the consumer's consent, or that asserts control over a device without the consumer's knowledge (Wikipedia n.d.).
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 2019). *(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 2019).
storyboard	A storyboard is a graphic organizer in the form of illustrations or images displayed in sequence for the purpose of pre-visualizing a motion picture, animation, motion graphic, interactive media sequence, or other computational artifact (Wikipedia n.d.).
string	A sequence of letters, numbers, and/or other symbols (TechTarget 2019). A string might represent, for example, a name, address, or song title. Some procedures commonly associated with strings are length, concatenation, and substring.

Term	Definition
structure	A general term used in the framework to discuss the concept of encapsulation without specifying a particular programming methodology. *See also* encapsulation.
subroutine	*See* procedure.
switch	A high-speed device that receives incoming data packets and redirects them to their destination on a local area network (LAN) (Techopedia 2019).
system	A collection of elements or components that work together for a common purpose (TechTarget 2019). *See also* 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 (Software Testing Fundamentals 2019).
testing	The process of verifying that a software program works as expected (Wikipedia n.d.).
topology	The physical and logical configuration of a network; the arrangement of a network, including its nodes and connecting links. A logical topology is the way devices appear connected to the user. A physical topology is the way they are actually interconnected with wires and cables (PCMag 2019).
transmission	Data transmission is the process of sending digital or analog data over a communication medium to one or more computing, network, communication or electronic devices (Techopedia 2019).
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 2019).
two-factor authentication	A security system that requires more than one method of authentication from independent categories of credentials to verify the user's identity for a login or other transaction. Multi-factor authentication (which includes two-factor authentication) combines two or more independent credentials: what the user knows (password), what the user has (physical security token), and what the user is (biometric verification) (TechTarget 2019).

Term	Definition
unplugged	An approach to teaching computer science without computers where concepts are presented through engaging activities and puzzles by using cards, crayons, and active playing (Hazzan, Lapidot, and Ragonis 2015, 110).
user	*See* end user.
variable	A memory location that stores a value. A variable is associated with a symbolic name (or identifier) and a data type. Variables are not just used for numbers; they can also hold text, including whole sentences (strings) or logical values (true or false). The value of a variable is normally changed during the course of program execution (Computing at School 2013; Reges and Stepp 2014; Techopedia 2019; Wikipedia n.d.). *Note: This definition differs from that used in math and statistics.* *See also* declare a variable.
version control software	Version control is used to manage multiple versions of computer files and programs. Version control software provides two primary data management capabilities: It allows users to (1) lock files so they can only be edited by one person at a time, and (2) track changes to files (TechTerms 2019).
virus	A type of malicious software program that, when executed, replicates itself by modifying other computer programs and inserting its own code. When this replication succeeds, the affected computers are then said to be "infected" with a computer virus (Wikipedia n.d.).
web filter	A program that can screen an incoming web page to determine whether some or all of it should not be displayed to the user. The filter checks the origin or content of a web page against a set of rules provided by the company or person who installed the web filter. A web filter allows an enterprise or individual to block out pages from websites that are likely to include objectionable content (TechTarget 2019).
worm	A type of computer virus that replicates itself to spread to uninfected computers, but does not alter any files on the computer. However, worms can still cause havoc by multiplying so many times that it takes up all the computer's available memory or hard disk space (Tech Terms 2019).

References

Some definitions came directly from these sources, while others were excerpted or adapted to include content relevant to this framework.

Alliance for California Computing Education for Students and Schools. 2019. Frequently Asked Questions: Computer Science and Science. ~~http://access-ca.org/frequently-asked-questions-computer-science-and-science/~~. [Preceding link no longer available]

Black, P. E. 2004. https://xlinux.nist.gov/dads/.

Center for Applied Special Technology (CAST). 2019. The UDL Guidelines. https://udlguidelines.cast.org/?utm_medium=web&utm_campaign=none&utm_source=cast-about-udl.

Computing at School. 2013. *Computing in the national curriculum: A guide for primary teachers.* Belford, UK: Newnorth Print. https://www.computingatschool.org.uk/resource-library/2014/september/computing-in-the-national-curriculum-a-guide-for-primary-teachers.

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

The Free Dictionary. n.d. https://www.thefreedictionary.com/.

Free On-line Dictionary of Computing (FOLDOC). n.d. https://foldoc.org.

Hazzan, O., T. Lapidot, and N. Ragonis. 2015. *Guide to Teaching Computer Science: An Activity-Based Approach.* 2nd ed., 2014 edition. London: Springer.

International Baccalaureate. 2019. Computer Science. https://www.ibo.org/programmes/diploma-programme/curriculum/sciences/computer-science/.

Israel, M., T. A. Lash, and G. Jeong. 2017. Utilizing the Universal Design for Learning Framework in K-12 Computer Science Education. Project TACTIC: Teaching All Computational Thinking through Inclusion and Collaboration. Adapted from University of Illinois, Creative Technology Research Lab website: https://CTRL.education.illinois.edu/TACTICal.html.

K–12 Computer Science Framework. 2016. https://k12cs.org/wp-content/uploads/2016/09/K%E2%80%9312-Computer-Science-Framework.pdf.

K–12 Computer Science Framework. n.d. Glossary. https://k12cs.org/glossary.

Kafai, Y. and Q. Burke. 2014. Connected Code: Why Children Need to Learn Programming. Cambridge, MA: MIT Press.

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. https://d-miller.github.io/DRK12/topic1/2301.pdf.

Massachusetts Department of Elementary and Secondary Education. 2016. 2016 Massachusetts Digital Literacy and Computer Science (DLCS) Curriculum Framework. https://www.doe.mass.edu/frameworks/dlcs.pdf.

Merriam-Webster. 2019. https://www.merriam-webster.com/.

National Council for the Social Studies (NCSS). 2013. 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. https://www.socialstudies.org/system/files/c3/C3-Framework-for-Social-Studies.pdf.

National Institute of Biomedical Imaging and Bioengineering (NIBIB). 2016. Computational Modeling. https://www.nibib.nih.gov/science-education/science-topics/computational-modeling.

Oxford Dictionaries. 2019. https://www.oxforddictionaries.com/us.

Papert, S. 1980. *Mindstorms: Children, Computers, and Powerful Ideas*. New York: Basic Books.

PCMag. 2019. https://www.pcmag.com/encyclopedia/term/46301/logical-vs-physical-topology.

Reges, S. and M. Stepp. 2014. *Building Java Programs: A back to basics approach*. Pearson Education.

Software Testing Fundamentals. 2019. ~~https://softwaretestingfundamentals.com/test-case/~~. [Preceding link not valid]

Tech Terms. 2019. Tech Terms Computer Dictionary. https://techterms.com/.

Techopedia. 2019. Technology Dictionary. https://www.techopedia.com/dictionary.

TechTarget. 2019. ~~https://www.techtarget.com/network~~. [No longer valid] Tucker, A., D. McCowan, F. Deek, C. Stephenson, J. Jones, and A.
Verno. 2006. *A Model Curriculum for K–12 Computer Science: Final report of the ACM K–12 task force curriculum committee.* 2nd ed. New York, NY: Association for Computing Machinery.

University of California Admissions. n.d. AP credits. ~~https://admission.universityofcalifornia.edu/counselors/exam-credit/ap-credits/index.html~~. [Preceding link not valid]

Webopedia. 2019. https://www.webopedia.com.

Wikipedia. n.d. https://www.wikipedia.org.

Page 262 intentionally blank.

ISBN 978-0-8011-1809-8