California Computer Science Standards - Introduction

Subsection Links

"Computer science is no more about computers than astronomy is about telescopes."
Anonymous, at times attributed to Edsger Dijkstra, 1970

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 K12 Computer Science Framework at https://k12cs.org/. 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 to 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 as well as examples for classroom application. Examples are not meant to be prescriptive nor 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 9–12 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 grades K–12, 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 have often taken 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 including: 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 of the introduction to the standards.

The computer science core concepts are infused with purpose and relevance via the computer science core practices. The core practices focus on how students *interact with* computer science, the ways in which they apply conceptual knowledge. It is these core practices that enable students to experience computer science as a creative process, moving them past the role of users of computing technology toward active creators and innovators, engaged 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, and (7) Communicating About Computing. Standards integrate computer science practices with concept statements.

Practice	+ Concept	= Standard
Creating Computational Artifact 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. *By the end of Grade 8, Sub-Concept Hardware & Software*	**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 K12 Computer Science Framework at https://k12cs.org/. These themes 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. In order to accurately reflect the field, the standards are interdisciplinary in nature to ensure that each and 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 standards alignment charts, according to grade level. The standards also complement the California Standards for Career Technical Education (CTE) Pathways.

Why Computer Science?

Computer science is an essential component of a broad and comprehensive education, containing necessary foundational concepts and corresponding practices that ensure 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 K12 Computer Science Framework at https://k12cs.org/, 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. While the impact of computer science on multiple areas of the human endeavor continues to increase rapidly, computer science education has not kept pace with this increased influence on society. The standards are an integral part of college and career

readiness for all students. 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 exam, 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 (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.

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 each and every student, grades K–12. Computer science core concepts and practices prepare each and every student for college and career, even if they do not pursue a computer science degree or occupation. The computer science core practices help to develop lifelong learners that persevere in processes of creative problem solving in a way that promotes inclusion and celebrates diversity. Computer science core concepts provide foundational knowledge of computing principles, providing 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 now, 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 fosters personal fulfillment by motivating students to become creative 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 each and every student. 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 & Gallup, 2015).

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 college and career as both critical consumers, and also thoughtful creators and innovators of computing technology.

Issues of Equity

California schools house the largest, most diverse population of students in the United States (California History-Social Studies Framework, 2016). As such, it is vital 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 each and every student develops foundational conceptual knowledge and proficiency in computer science practices to provide the skills to responsibly and productively participate in a world in which digital technologies are broadly integrated. More than availability of computer science classes, equity requires leaders and educators to carefully consider 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 merely to a select few, or as an elective for interested students. Equity in computer science calls upon 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). Computer science education rates at the K–8 level are even more dismal. While 60 percent of California's student population is Latinx or African American, these students comprise only 16 percent of students taking AP CS A

and 15 percent of the technology workforce (College Board, 2016; EEOC, 2016). While female students are 49 percent of the population, they comprised only 27 percent of AP CS A test takers in the state of California (College Board, 2016). 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 & Gallup, 2017).

The standards are designed for each and every student, 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. In order 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 to educators, curriculum leaders, and administrators regarding flexible implementation options to ensure universal access to the standards are 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. It is this inclusive, user-centric mindset within computer science that has 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 celebration of diversity.

Problem Solving and the 4 Cs

College and careers of the future require students to problem solve, communicate, think critically, create, and collaborate. These skills are echoed throughout the California State Standards in English Language Arts, English Language Development, Mathematics, Next Generation Science Standards, the California History-Social Studies Framework, California Career Technical Education Anchor Standards, California Visual and Performing Arts Content Standards, and California Health Education Standards. 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 4 Cs. More detail follows regarding computer science practice connections to problem solving and the 4 Cs. For more detail regarding computer science in the context of equity, reference the previous section entitled **Issues of Equity**.

Equity ≈ Practice 1: Fostering an Inclusive Computing Culture

Collaboration ≈ Practice 2: Collaborating Around Computing

Problem Solving ≈ Practice 3: Recognizing & Defining Computational Problems

Critical Thinking ≈ Practice 4: Developing & Using Abstractions

Creativity ≈ Practice 5: Creating Computational Artifacts and **Practice 6**: Testing & Refining Computational Artifacts

Communication ≈ Practice 7: Communicating About Computing

Collaboration in computer science fosters contributions and feedback from others, which may result in improved outcomes as opposed to working independently. California Computer Science core practice 2, *Collaborating About 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 use of technological tools to support collaborative work. In the computer science industry, development teams collaborate together, soliciting feedback and providing feedback to others, in order to meet common goals.

Problem solving is a foundation of computer science. By nature, computer science exists to solve problems for people and the world. Computer scientists create algorithms, step-by-step instructions for a program, to design potential solutions for end users. The study of computer science core concepts promotes empathy, urging students to identify authentic problems and create solutions to increase accessibility

and/or functionality based on users' individual needs. The California Computer Science standards emphasize problem solving as a necessary practice in the study of computer science. 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 decompose these problems and develop potential solutions for the betterment of society.

Critical thinking is a key component of computer science, as exhibited in computer science 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 practice ideation (development of 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. California computer science core practice 5, *Creating Computational Artifacts*, emphasizes the need for students to iteratively create when engaging in computer science learning. Creation is an iterative process. It is this iteration that leads to innovation. California Computer Science 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. California Computer Science Standards 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 to communicate in varying language registers. According to the California 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 ELA/ELD Framework, 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. In order to provide universal access to computer science instruction for each and every student, we must thoroughly define computer science. Computer science can be defined as “the study of computers and algorithmic processes, including their principles, their hardware and software designs, their applications, and their impact on society” (Tucker et. al, 2006, p. 2). Computer science is the science of computing. The term computer science is often misconstrued with other technological and digital terminology.

Computer Science **IS**	Computer Science **IS NOT**
• a theory and practice that allows you to program a computer to do what you want it to • 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 • a skill that teaches students how to use computers to create, not just consume	• learning how to type or use a mouse • learning to use word processing, spreadsheet, or presentation software (e.g., Word, PowerPoint, Google Docs & Drive) • learning how to build or repair computers • playing video games • learning Smarter Balanced (SBAC) skills

Adapted from CS First at https://csfirst.withgoogle.com/en/home

While computer science is not computer literacy, educational technology, digital citizenship, and information technology, it builds upon these and goes further in complexity and depth. According to the K12 Computer Science Framework at https://k12cs.org/, further differentiators are as follows:

- ***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 calls upon students to understand why and how computing technologies work, and then build upon that conceptual knowledge by creating computational artifacts.

In accordance with the K12 Computer Science Framework at https://k12cs.org/, 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 of the standards. 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 included 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 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 California Computer Science 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 California Computer Science Standards. Core concept definitions and subconcept descriptions are taken from K12 Computer Science Framework at https://k12cs.org/.

Core Concept CS

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 (D)	Subconcept Hardware and Software (HS)	Subconcept Troubleshooting (T)
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.	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.	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 NI

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 (NCO)	Subconcept Cybersecurity (CS)
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.	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 DA

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

Subconcept Storage (S)	Subconcept Collection, Visualization, and Transformation (CVT)	Subconcept Inference and Models (IM)

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.	Data is collected with both computational and non-computational tools and processes. In early grades, students learn how data about themselves and their world is collected and used. As they progress, students learn the effects of collecting data with computational and automated tools. 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.	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 AP

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 (A)	Subconcept Variables (V)	Subconcept Control (C)	Subconcept Modularity (M)	Subconcept Program Development (PD)

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.	Computer programs store and manipulate data using variables. In early grades, students learn that different types of data, such as words, numbers, or pictures, can be used in different ways. As they progress, students learn about variables and ways to organize large collections of data into data structures of increasing complexity.	Control structures specify the order in which instructions are executed within an algorithm or program. In early grades, students learn about sequential execution and simple control structures. As they progress, students expand their understanding to combinations of structures that support complex execution.	Modularity 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.	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 IC

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 (C)	Subconcept Social Interactions (SI)	Subconcept Safety, Law, and Ethics (SLE)
Computing influences culture—including belief systems, language, relationships, technology, and institutions—and culture shapes how people engage with and access computing. In early grades, students learn how computing can be helpful and harmful. As they progress, students learn about tradeoffs associated with computing and potential future impacts of computing on global societies.	Computing 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.	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 California Computer Science Standards represent how students *do* computer science while building conceptual knowledge of the key content areas. According to K12 Computer Science Framework at https://k12cs.org/, 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 K12 Computer Science Framework at https://k12cs.org/. Practice statements describe what students should be able to do by the end of grade 12.

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 12 students should be able to:

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

    - At all grade levels, students should recognize that the choices people make when they create artifacts are based on personal interests, experiences, and needs. Young learners should begin to differentiate their technology preferences from the technology preferences of others. Initially, students should be presented with perspectives from people with different backgrounds, ability levels, and points of view. As students progress, they should independently seek diverse perspectives throughout the design process for the purpose of improving their computational artifacts. Students who are well-versed in fostering an inclusive computing culture should be able to differentiate backgrounds and 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 12 students should be able to:

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

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

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

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

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

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

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

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

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

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 12 students should be able to:

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

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

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

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

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

- After students have had some experience breaking problems down (P3.2) and identifying 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 12 students should be able to:

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

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

2. Evaluate existing technological functionalities and incorporate 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 12 students should be able to:

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

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

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

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

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

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

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 12 students should be able to:

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

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

2. Identify and fix errors using a systematic process.

    - At any grade level, students should be able to identify and fix errors in programs (debugging) and use strategies to solve problems with computing systems (troubleshooting). Young students could use trial and error to fix simple errors. For example, a student may try reordering the sequence of commands in a program. In a hardware context, students

could try to fix a device by resetting it or checking whether it is connected to a network. As students progress, they should become more adept at debugging programs and begin to consider logic errors: cases in which a program works, but not as desired. In this way, students will examine and correct their own thinking. For example, they might step through their program, line by line, to identify a loop that does not terminate as expected. Eventually, older students should progress to using more complex strategies for identifying and fixing errors, such as printing the value of a counter variable while a loop is running to determine how many times the loop runs.

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

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

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 12 students should be able to:

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

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

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

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

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

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

References and Attributions

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

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

College Board (2016). AP State Report: California.

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

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

EEOC (2016). Diversity in High Tech.

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

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

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

Google & Gallup. (2016). *Trends in the state of computer science in U.S. K–12 schools.* Retrieved from http://g.co/cseduresearch.

Google & Gallup. (2017). *Computer Science Learning: Closing the Gap: Rural and Small Town School Districts.* Retrieved from http://g.co/cseduresearch.

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

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

Level Playing Field Institute (2015). Path Not Found: Disparities in Access to CS Courses in California High Schools.

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

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

California Department of Education: August 2018