Arizona Computer Science Standards

High School

ARIZONA DEPARTMENT OF EDUCATION
HIGH ACADEMIC STANDARDS FOR STUDENTS
Adopted October 2018

Contents

Vision Statement 3
Introduction 3
Focus On Equity 4
Acknowledgements 4
Computer Science Essential Concepts and Subconcepts 5
Computer Science Practices for Students 7
How to Read the Arizona Computer Science Standards 10
High School 12
Concept: Computing Systems (CS) 12
Concept: Networks and the Internet (NI) 13
Concept: Data and Analysis (DA) 15
Concept: Algorithms and Programming (AP) 16
Concept: Impacts of Computing (IC) 18
Appendix A-Computer Science Glossary 21
Sources for definitions in this glossary: 27
Appendix B-Computer Science Practices 28

Vision Statement

Arizona's K-12 students will develop a foundation of computer science knowledge and learn new approaches to problem solving and critical thinking. Students will become innovative, collaborative creators and ethical, responsible users of computing technology to ensure they have the knowledge and skills to productively participate in a global society.

Introduction

Understanding problems, their potential solutions, and the technologies, techniques, and resources needed to solve them are vital for citizens of the 21st century. The State of Arizona has created computer science standards to further this understanding. These standards allow students to develop a foundation of computer science knowledge. By learning new approaches to problem solving that capture the power of computational thinking, students become users and creators of computing technology. The computer science standards will empower students to:

- Be informed citizens who can critically engage in public discussion on computer science related topics
- Develop as learners, users, and creators of computer science knowledge and artifacts
- Understand the role of computing in the world around them
- Learn, perform, and express themselves critically in a variety of subjects and interests

As the foundation for all computing, computer science is "the study of computers and algorithmic processes, including their principles, their hardware and software designs, their applications, and their impact on society" (Tucker et. al, 2006, p. 2). Computer science builds upon the concepts of computer literacy, educational technology, digital citizenship, and information technology. The previously listed concepts tend to focus more on using computer technologies as opposed to understanding why they work and how to create those technologies (K-12 Computer Science Framework, 2016). The differences and relationship with computer science are described below.

- ***Computer literacy*** refers to the general use of computers and programs, such as productivity software. 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.

Focus On Equity

Computer Science education has a history of significant access challenges, especially for early elementary students, students with disabilities, and women & minority students [csta-https://c.ymcdn.com/sites/www.csteachers.org/resource/resmgr/CSTAPolicyBrochure.pdf]. These Computer Science standards are intended to close the access gap for underserved populations and provide a foundation in computer science to *all* Arizona students.

All students and teachers have the opportunity to engage in rigorous, robust computer science standards. Each standard provides additional guidance and examples for implementation. Many standards include guidance and examples for use without computing devices, allowing for flexible implementation in lesson design and delivery.

The Arizona Computer Science Standards provide flexibility to allow students to demonstrate proficiency in multiple ways, thus providing a rigorous opportunity for engagement in computer science.

Acknowledgements

Numerous existing sets of standards and standards-related documents have been used in developing the Arizona Computer Science Standards. These include:

- The (Interim) CSTA K-12 Computer Science Standards, revised 2016 http://www.csteachers.org/?page=CSTA_Standards
- The K-12 Computer Science Framework https://k12cs.org/
- Approved or draft standards from the following states:
    - Nevada: http://www.doe.nv.gov/uploadedFiles/nde.doe.nv.gov/content/Standards_Instructional_Support/Nevada_Academic_Standards/Comp_Tech_Standards/DRAFTNevadaK-12ComputerScienceStandards.pdf
    - Wisconsin: https://dpi.wi.gov/sites/default/files/imce/computer-science/ComputerScienceStandardsFINALADOPTED.pdf

Computer Science Essential Concepts and Subconcepts

The Arizona Computer Science Standards for grades kindergarten through twelve are organized into five Essential Concepts:

- **Computing Systems:** This involves the interaction that people have 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.

- **Networks and the Internet (with Cybersecurity):** This involves the networks that connect computing systems. Computing devices 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. Networking and the Internet must also consider Cybersecurity. Cybersecurity, also known as information technology security, involves the protection of computers, networks, programs, and data from unauthorized or unintentional access, manipulation, or destruction. Many organizations, such as government, military, corporations, financial institutions, hospitals, and others collect, process, and store significant amounts of data on computing devices. That data is transmitted across multiple networks to other computing devices. The confidential nature of government, financial, and other types of data requires continual monitoring and protection for the sake of continued operation of vital systems and national security.

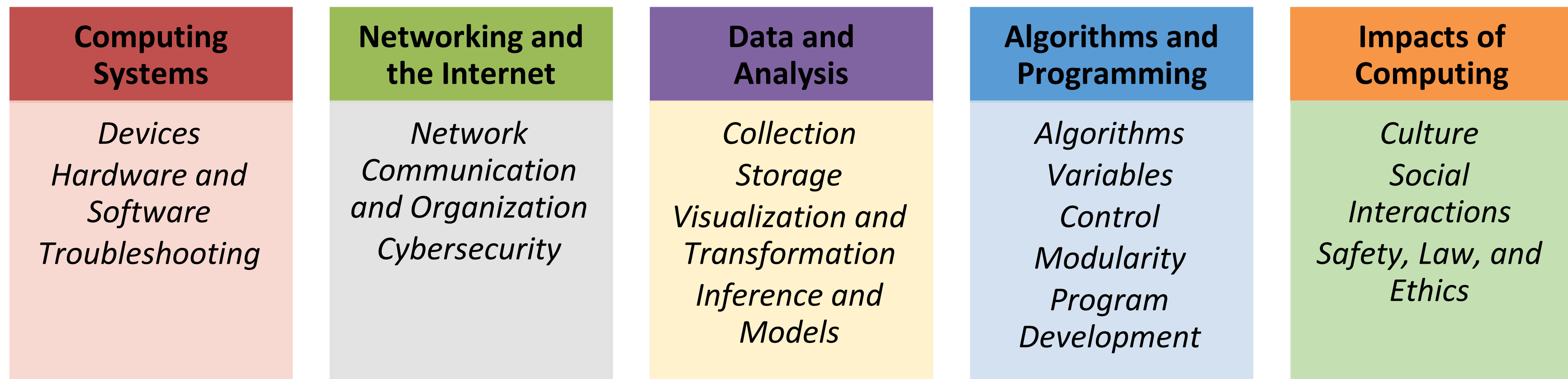
- **Data and Analysis:** This involves the data that exist and the computing systems that exist to process that data. The amount of digital data generated in the world is rapidly expanding, so the need to process data effectively is increasingly important. Data is collected and stored so that it can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

- **Algorithms and Programming:** Involves the use of algorithms. An algorithm is a sequence of steps designed to accomplish a specific task. Algorithms are translated into programs, or code, to provide instructions for computing devices. 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.

- **Impacts of Computing:** This involves the effect that computing has on daily life. 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.

Concepts are categories that represent major content areas in the field of computer science. They represent specific areas of disciplinary importance rather than abstract, general ideas. Each essential concept is supported by various subconcepts that represent specific ideas within each concept. Figure 1 provides a visual representation of the Essential Concepts and the supporting subconcepts.

Figure 1: Computer science essential concepts and subconcepts



Computer Science Practices for Students

The content of the Arizona Computer Science Standards is intended to support the following seven practices for students. The practices describe the behaviors and ways of thinking that computationally literate students use to fully engage in a data-rich and interconnected world.

- **Fostering an Inclusive Computing Culture:** Students will develop skills for building an inclusive and diverse computing culture, which 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.

- **Collaborating Around Computing:** Students will develop skills for collaborating around computing. Collaborative computing is the process of performing a computational task by working in pairs and on teams. Collaborative computing 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.

- **Recognizing and Defining Computational Problems:** Students will develop skills for 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.

- **Developing and Using Abstractions**: Students will develop skills for developing and using abstractions. Identifying patterns and extracting common features from specific examples to create generalizations form abstractions. Using generalized solutions and parts of solutions designed for broad reuse simplifies the development process by managing complexity.

- **Creating Computational Artifacts:** Students will develop skills for 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.

- **Testing and Refining Computational Artifacts:** Students will develop skills for 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.

- **Communicating About Computing**: Students will develop skills for 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.

See **Appendix B-Computer Science Practices,** for a complete, numbered listing of the sub-practices under each practice.

Regarding the previously listed practices, computational thinking is integrated throughout each one. Computational thinking is an approach to solving problems in a way that can be implemented with a computer. It involves the use of concepts, such as *abstraction, recursion, and iteration*, to process and analyze data, and to create real and virtual artifacts (Computer Science Teachers Association & Association for Computing Machinery, 2017). Computational thinking practices such as abstraction, modeling, and decomposition connect with computer science concepts such as algorithms, automation, and data visualization. Beginning with the elementary school grades and continuing through grade 12, students should develop a foundation of computer science knowledge and learn new approaches to problem solving that captures the power of computational thinking to become both users and creators of computing technology. Figure 2 is a visual representation of the essential practices along with computational thinking.

Figure 2: Essential Practices including computational thinking

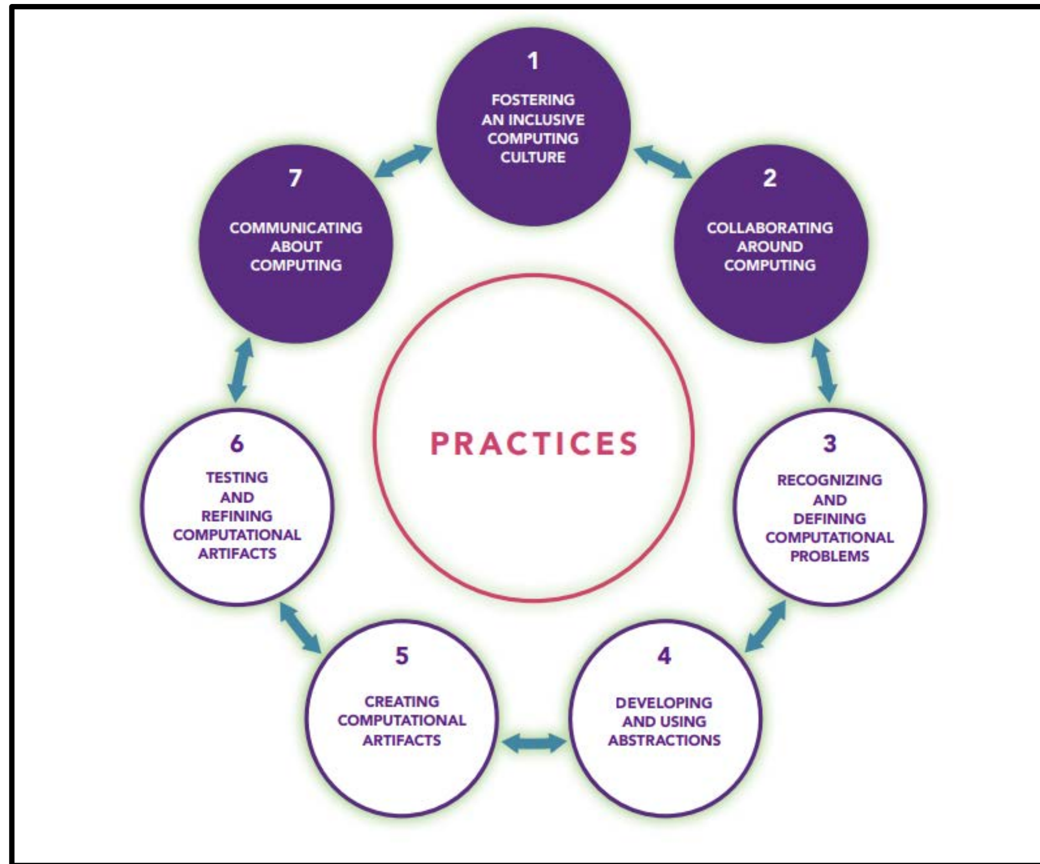
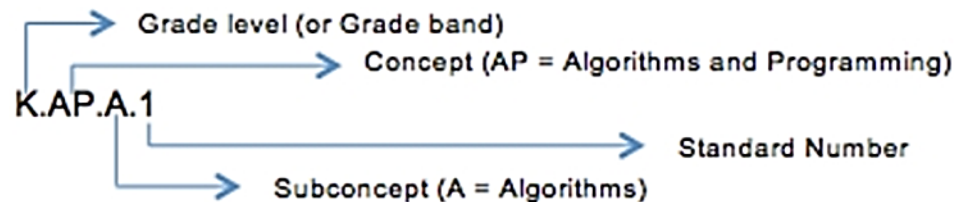


Figure 2: Practices- K-12 Computer Science Framework. (2016)

How to Read the Arizona Computer Science Standards

The Arizona Computer Science Standards are divided into Grades K, 1, 2, 3, 4, 5, 6, 7, 8, and 9-12. The standards are divided by the five main concepts. Those main concepts include computing systems, networks and the internet, data and analysis, algorithms and programming, and impacts of computing. Within each main concept there may be two to five sub concepts represented, such as algorithms, program development, variables, troubleshooting, or cybersecurity. Each standard will list the grade level, the concept, the subconcept, and the standard number. Figure 3 provides an example of the coding for a standard:

Figure 3: Standard Coding Scheme for Standards



Each standard appears like the example in Figure 4. This example shows two different subconcepts under the concept of Algorithms and Programming at the Kindergarten level.

Figure 4: Example of Complete Individual Standard

Subconcept: Algorithms	
K.AP.A.1	**With teacher assistance, model daily processes by following algorithms (sets of step-by-step instructions) to complete tasks.** *Routines, such as morning meeting, clean-up time, and dismissal, are examples of algorithms that are common in many early elementary classrooms. Just as people use algorithms to complete daily routines, they can program computers to use algorithms to complete different tasks. Algorithms are commonly implemented using a precise language that computers can interpret. For example, students begin to recognize daily step-by-step processes, such as brushing teeth or following a morning procedure, as "algorithms" that lead to an end result.* *Practice(s):* Developing and Using Abstractions: 4.4
Subconcept: Variables	
K.AP.V.1	**With teacher assistance, model the way programs store and manipulate data by using numbers or other symbols to represent information.** *Information in the real world can be represented in computer programs. Students could use thumbs up/down as representations of yes/no, use arrows when writing algorithms to represent direction, or encode and decode words using numbers, pictographs, or other symbols to represent letters or words.* Practice(s): Developing and Using Abstractions: 4.4

High School

Students build on K–8 experiences and learn more technical and sophisticated applications. Students refine their skills in differentiating problems or subproblems that are best solved by computing systems or digital tools and those that are best solved by humans. Students will further develop their computational thinking and problem solving skills to support the selection and appropriate use of technology. Computer science literate students will be able to explore how abstractions are integrated in computing systems within software and hardware. They can also evaluate security measures in order to protect sensitive data. Students will utilize data visualizations to represent and communicate real-world phenomena. Individually and in small groups they will consider the scalability and reliability of networks, including data storage; and collaboratively use algorithms to develop computational solutions to solve real-world challenges. Students will develop guidelines and strategies to solve computational problems. Successful students will implement computational solutions across disciplines, taking into consideration innovation, privacy, bias and equity, personal, ethical, social, economic, and cultural practices. High school students also have expanded learning opportunities in both the Career and Technical Education (CTE) and Advanced Placement (AP) programs

Concept: Computing Systems (CS)

Subconcept: Devices (D)	
HS.CS.D.1	**Explain how abstractions hide the underlying implementation details of computing systems embedded in everyday objects.** *Computing devices are often integrated with other systems, including biological, mechanical, and social systems. Students could explore how a medical device can be embedded inside a person to monitor and regulate his or her health, a hearing aid (a type of assistive device) can filter out certain frequencies and magnify others, a monitoring device installed in a motor vehicle can track a person's driving patterns and habits, or a facial recognition device can be integrated into a security system to identify a person. The usability, dependability, security, and accessibility of these devices, and the systems with which they are integrated are important considerations in their evolving design. Students are not expected to create integrated or embedded systems at this level.* *Practice(s):* Developing and Using Abstractions: 4.1
Subconcept: Hardware and Software (HS)	
HS.CS. HS.1	**Describe levels of abstraction and interactions between application software, system software, and hardware layers.** *At its most basic level, a computer is composed of physical hardware and electrical impulses. Multiple layers of software are built upon the hardware and interact with the layers above and below them to reduce complexity. System software manages a computing device's resources so that software can interact with hardware. For example, text editing software interacts with the operating system to receive input from the keyboard, convert the input to bits for storage, and interpret the bits as readable text to display on the monitor. System software is used on many different types of devices, such as smart TVs, assistive devices, virtual components, cloud*

	components, and drones. For example, students may explore the progression from voltage to binary signal to logic gates to adders and so on. Knowledge of specific, advanced terms for computer architecture, such as BIOS, kernel, or bus, is not expected at this level. *Practice(s):* Developing and Using Abstractions: 4.1
Subconcept: Troubleshooting (T)	
HS.CS.T.1	**Develop guidelines that convey systematic troubleshooting strategies that others can use to identify and fix errors.** *Troubleshooting complex problems involves the use of multiple sources when researching, evaluating, and implementing potential solutions. Troubleshooting also relies on experience, such as when people recognize that a problem is similar to one they have seen before or adapt solutions that have worked in the past. Examples of complex troubleshooting strategies include resolving connectivity problems, adjusting system configurations and settings, ensuring hardware and software compatibility, and transferring data from one device to another. Students could create a flow chart, a job aid for a help desk employee, or an expert system.* *Practice(s):* Testing and Refining Computational Artifacts: 6.2

Concept: Networks and the Internet (NI)

Subconcept: Cybersecurity (C)	
HS.NI.C.1	**Describe how sensitive data can be affected by malware and other attacks.** *Network security depends on a combination of hardware, software, and practices that control access to data and systems. Potential security problems, such as denial-of-service attacks, ransomware, viruses, worms, spyware, and phishing, present threats to sensitive data. Students might reflect on case studies or current events in which governments or organizations experienced data leaks or data loss as a result of these types of attacks.* *Practice(s): Communicating About Computing: 7.2*
HS.NI.C.2	**Recommend security measures to address various scenarios based on factors such as efficiency, feasibility, and ethical impacts.** *Security measures may include physical security tokens, two-factor authentication, and biometric verification. The timely and reliable access to data and information services by authorized users, referred to as availability, and is ensured through adequate bandwidth, backups, and other measures. Students should systematically evaluate different security measures based on the requirements or constraints of a situation, 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, and make recommendations based on their analysis.* *Practice(s): Recognizing and Defining Computational Problems: 3.3*
HS.NI.C.3	**Compare various security measures, considering tradeoffs between the usability and security of a computing system.** *Choosing security measures involves tradeoffs between the usability and security of the system. The needs of users and the sensitivity of data determine the level of security implemented. Students might discuss computer security policies in place at the local level that*

	present a tradeoff between usability and security, such as a web filter that prevents access to many educational sites but keeps the campus network safe. *Practice(s): Testing and Refining Computational Artifacts: 6.3*

Subconcept: Network, Communication, and Organization (NCO)	
HS.NI. NCO.1	**Evaluate the scalability and reliability of networks, by describing the relationship between routers, switches, servers, topology, and addressing.** *Each device is assigned an address that uniquely identifies it on the network. Routers function by comparing IP addresses to determine the pathways packets should take to reach their destination. Switches function by comparing MAC addresses to determine which computers or network segments will receive frames. Students could use online network simulators to experiment with these factors.* *Practice(s): Developing and Using Abstractions: 4.1*

Concept: Data and Analysis (DA)

Subconcept: Collection, Visualization and Transformation (CVT)	
HG.DA. CVT.1	**Create interactive data visualizations using software tools to help others better understand real-world phenomena.** *People use software tools or programming to create powerful, interactive data visualizations and perform a range of mathematical operations to transform and analyze data. Students should model phenomena as systems, with rules governing the interactions within the system and evaluate these models against real-world observations.* *Practice(s): Developing and Using Abstractions: 4.4*
Subconcept: Storage (S)	
HS.DA.S.1	**Translate between different bit representations of real-world phenomena, such as characters, numbers, and images.** *Most computing systems use different numerical representations of non-numerical data. For example, convert hexadecimal color codes to decimal numbers, or represent characters in their ASCII/Unicode representation.* *Practice(s): Developing and Using Abstractions: 4.1*
HS.DA.S.2	**Evaluate the tradeoffs in how and where data is stored.** *People make choices about how and where data is stored. Students might consider the cost, speed, reliability, accessibility, privacy, and integrity tradeoffs between storing photo data on a mobile device versus in the cloud. Students should evaluate whether a chosen solution is most appropriate for a particular problem.* *Practice(s): Recognizing and Defining Computational Problems: 3.3*

Subconcept: Inference and Models (IM)	
HS.DA. IM.1	**Analyze computational models to better understand real-world phenomena.** *Computational models make predictions about processes or phenomenon based on selected data and features that can be represented in a spreadsheet or other organizational software. The amount, quality, and diversity of data and the features chosen can affect the quality of a model and ability to understand a system. Predictions or inferences are tested to validate models. Students should model phenomena as systems, with rules governing the interactions within the system. Students should analyze and evaluate these models against real-world observations.* *Practice(s): Developing and Using Abstractions: 4.4*

Concept: Algorithms and Programming (AP)

Subconcept: Algorithms (A)	
HS.AP.A.1	**Create prototypes that use algorithms for practical intent, personal expression, or to address a societal issue** *A prototype is a computational artifact that demonstrates the core functionality of a product or process. Prototypes are useful for getting early feedback in the design process and can yield insight into the feasibility of a product. The process of developing computational artifacts embraces both creative expression and the exploration of ideas to create prototypes and solve computational problems. Students create artifacts that are personally relevant or beneficial to their community and beyond. Students should develop computational artifacts in response to a task or a computational problem that demonstrate the performance, reusability, and ease of implementation of an algorithm.* *Practice(s): Creating Computational Artifacts: 5.2*
Subconcept: Variables (V)	
HS.AP.V.1	**Use lists to simplify solutions, generalizing computational problems instead of repeatedly using simple variables.** *Students should be able to identify common features in multiple segments of code and substitute a single segment that uses lists (arrays) to account for the differences.* *Practice(s): Developing and Using Abstractions: 4.1*
Subconcept: Control (C)	
HS.AP.C.1	**Justify the selection of specific control structures and explain the benefits and drawbacks of choices made, when tradeoffs involve readability and program performance.** *Readability refers to how clear the program is to other programmers and can be improved through documentation. The discussion of performance is limited to a theoretical understanding of execution time; a quantitative analysis is not expected. Control structures at*

	this level may include conditional statements, loops, event handlers, and recursion. Students might compare several implementations of the same algorithm with different structures and discuss the tradeoffs of each implementation. *Practice(s): Recognizing and Defining Computational Problems: 5.2*
HS.AP.C.2	**Use events that initiate instructions to design and iteratively develop computational artifacts** *In this context, relevant computational artifacts include programs, mobile apps, or web apps for practical intent, personal expression, or to address a societal issue. Events can be user-initiated, such as a button press, or system-initiated, such as a timer firing. At previous levels, students have learned to create and call procedures. Here, students design procedures that are called by events.* *Practice(s): Creating Computational Artifacts: 5.1*
Subconcept: Modularity (M)	
HS.AP.M.1	**Decompose problems into smaller components using constructs such as procedures, modules, and/or objects.** *At this level, students should decompose complex problems into manageable subproblems that could potentially be solved with programs or procedures that already exist. A game program can be made up of objects representing different characters, methods defining how each behaves, and procedures for various events.* *Practice(s): Recognizing and Defining Computational Problems: 3.2*
HS.AP.M.2	**Use procedures within a program, combinations of data and procedures, or independent but interrelated programs to design and iteratively develop computational artifacts.** *Computational artifacts can be created by combining and modifying existing artifacts or by developing new artifacts. Complex programs are designed as systems of interacting procedures, each with a specific role, coordinating for a common overall purpose. The focus at this level is understanding a program as a system with relationships between procedures.* *Practice(s): Creating Computational Artifacts: 5.2*
Subconcept: Program Development (PD)	
HS.AP.PD.1	**Evaluate and refine computational artifacts to make them more usable and accessible.** *Testing and refinement is the deliberate and iterative process of improving a computational artifact. This process includes debugging (identifying and fixing errors) and comparing actual outcomes to intended outcomes. Students should respond to the changing needs and expectations of end users and improve the performance, reliability, usability, and accessibility of artifacts. At this level, students should work through a systematic process that includes feedback from broad audiences.* *Practice(s): Testing and Refining Computational Artifacts: 6.3*
HS.AP.PD.2	**Use team roles and collaborative tools to design and iteratively develop computational artifacts.** *Most software is developed in teams which can include pair programming or other collaborative structures. Team roles in pair programming are alternating driver and navigator but could be more specialized in larger teams. Students may choose to use collaborative tools to aid their team, such as a version control system or project management interface.* *Practice(s): Collaborating Around Computing: 2.1*

HS.AP.PD.3	**Document design decisions using text, graphics, presentations, and/or demonstrations in the development of complex programs.** *Complex programs are designed as systems of interacting modules, each with a specific role, coordinating for a common overall purpose. These modules can be procedures within a program; combinations of data and procedures; or independent, but interrelated, programs. Students might track their design decisions while developing a program, then choose a representation to communicate how each piece of their program contributes to the program as a whole.* *Practice(s): Communicating About Computing: 7.2*

Concept: Impacts of Computing (IC)

Subconcept: Culture (C)	
HS.IC.C.1	**Evaluate the ways access to computing impacts personal, ethical, social, economic, and cultural practices.** *Computing may improve, harm, or maintain practices. Equity deficits, such as minimal exposure to computing, access to education, and training opportunities, are related to larger, systemic problems in society. Students should be able to evaluate the accessibility of a product to a broad group of end users, such as people who lack access to broadband or who have various disabilities.* *Practice(s): Fostering an Inclusive Computing Culture: 1.2*
HS.IC.C.2	**Test and refine computational artifacts to reduce bias and equity deficits.** *Biases could include incorrect assumptions developers have made about their user base or data. Students should begin to identify potential bias during the design process to maximize accessibility in product design and become aware of professionally accepted accessibility standards to evaluate computational artifacts for accessibility.* *Practice(s): Fostering an Inclusive Computing Culture: 1.2*
HS.IC.C.3	**Demonstrate ways a given algorithm applies to problems across disciplines.** *Computation can share features with disciplines such as art and music by algorithmically translating human intention into an artifact. Students should be able to identify real-world problems that span multiple disciplines and can be solved computationally, such as increasing bike safety with new helmet technology.* *Practice(s): Recognizing and Defining Computational Problems: 3.1*

Subconcept: Social Interactions (SI)	
HS.IC.SI.1	**Analyze the impact of collaborative tools and methods that increase social connectivity.** *Many aspects of society, especially careers, have been affected by the degree of communication afforded by computing. The increased connectivity between people in different cultures and in different career fields has changed the nature and content of many careers. Students should explore different collaborative tools and methods used to solicit input from team members, classmates, and others, such as participation in online forums or local communities. For example, students could compare ways different social media tools could help a team become more cohesive.*

	Practice(s): Collaborating Around Computing: 2.4

Subconcept: Safety, Law, and Ethics (SLE)	
HS.IC. SLE.1	**Explain the beneficial and harmful effects that intellectual property laws can have on innovation.** *Laws govern many aspects of computing, such as privacy, data, property, information, and identity. These laws can have beneficial and harmful effects, such as expediting or delaying advancements in computing and protecting or infringing upon people's rights. International differences in laws and ethics have implications for computing. For examples, laws that mandate the blocking of some file-sharing websites may reduce online piracy but can restrict the right to access information. Students should be aware of intellectual property laws and be able to explain how they are can be used to protect the interests of innovators or can be potentially be misused.* *Practice(s): Communicating About Computing: 7.3*
HS.IC. SLE.2	**Explain the privacy concerns related to the collection and generation of data through automated processes that may not be evident to users.** *Data can be collected and aggregated across millions of people, even when they are not actively engaging with or physically near the data collection devices. This automated and non-evident collection can raise privacy concerns, such as social media sites mining an account even when the user is not online. Students might review situations where this automated collection has led to unintended consequences or accidental breaches in privacy.* *Practice(s): Communicating About Computing: 7.2*
HS.IC. SLE.3	**Evaluate the social and economic implications of privacy in the context of safety, law, or ethics.** *Laws govern many aspects of computing, such as privacy, data, property, information, and identity. International differences in laws and ethics have implications for computing. Students might review case studies or current events which present an ethical dilemma when an individual's right to privacy is at odds with the safety, security, or wellbeing of a community.* *Practice(s): Communicating About Computing: 7.3*

Appendix A-Computer Science Glossary

The following glossary includes definitions of commonly-used computer science terms and was borrowed (with permission) from the K–12 Computer Science Framework. This section is intended to increase teacher understanding and use of computer science terminology.

Abstraction: Pulling out specific difference to make one solution work for multiple problems.

- (Process): The process of reducing complexity by focusing on the main idea. By hiding details irrelevant to the question at hand and bringing together related and useful details, abstraction reduces complexity and allows one to focus on the problem. In elementary classrooms, abstraction is hiding unnecessary details to make it easier to think about a problem.
- (Product): A new representation of a thing, a system, or a problem that helpfully reframes a problem by hiding details irrelevant to the question at hand.

Algorithm: A step-by-step process to complete a task. A list of steps to finish a task. A set of instructions that can be performed with or without a computer.

For example, the collection of steps to make a peanut butter and jelly sandwich is an algorithm.

App: A type of application software, often designed to run on a mobile device, such as a smartphone or tablet computer (also known as a mobile application).

Artifact: Anything created by a human. See "computational artifact" for the computer science-specific definition.

ASCII: (American Standard Code for Information Interchange) is the most common format for text files in computers and on the Internet. In an ASCII file, each alphabetic, numeric, or special character is represented with a 7-bit binary number (a string of seven 0s or 1s). 128 possible characters are defined.

Automation: The process of linking disparate systems and software in such a way that they become self-acting or self-regulating.

Backup: The process of making copies of data or data files to use in the event the original data or data files are lost or destroyed.

Binary: A system of notation representing data using two symbols (usually 1 and 0).

Block-based programming language: Any programming language that lets users create programs by manipulating "blocks" or graphical programming elements, rather than writing code using text. (Sometimes called visual coding, drag and drop programming, or graphical programming blocks)

Bug: An error in a software program. It may cause a program to unexpectedly quit or behave in an unintended manner. The process of removing errors (bugs) is called debugging.

Cloud: Remote servers that store data and are accessed from the Internet.

Code: Any set of instructions expressed in a programming language. One or more commands or algorithm(s) designed to be carried out by a computer. See also: program

Command: An instruction for the computer. Many commands put together make up algorithms and computer programs.

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.

Computational models: Used to make predictions about processes or phenomenon based on selected data and features that can be represented by organizational software.

Computational thinking: Mental processes and strategies that include: decomposition (breaking larger problems into smaller, more manageable problems), pattern matching (finding repeating patterns), abstraction (identifying specific changes that would make one solution work for multiple problems), and algorithms (a step-by-step set of instructions that can be acted upon by a computer).

Computer science: The study of computers and algorithmic processes including their principles, hardware and software design, their applications, and their impact on society.

Conditionals: Statements that only run under certain conditions or situations.

Data: Information. Often, quantities, characters, or symbols that are the inputs and outputs of computer programs.

Debugging: Finding and fixing errors in programs.

Decompose: Break a problem down into smaller pieces.

Decryption: The process of taking encoded or encrypted text or other data and converting it back into text that you or the computer can read and understand.

Digital divide: the gulf between those who have ready access to computers and the Internet, and those who do not.

Encryption: The process of encoding messages or information in such a way that only authorized parties can read it.

Event: An action that causes something to happen

Execution: The process of executing an instruction or instruction set.

For loop: A loop with a predetermined beginning, end, and increment (step interval)

Function: A type of procedure or routine. Some programming languages make a distinction between a function, which returns a value, and a procedure, which performs some operation, but does not return a value. Note: This definition differs from that used in math. A piece of code that you can easily call over and over again. Functions are sometimes called 'procedures.'

GPS: Abbreviation for "Global Positioning System." GPS is a satellite navigation system used to determine the ground position of an object.

Hacking: Appropriately applying ingenuity. Cleverly solving a programming problem. Using a computer to gain unauthorized access to data within a system.

Hardware: The physical components that make up a computing system, computer, or computing device.

Hierarchy: An organizational structure in which items are ranked according to levels of importance.

HTTP: (Hypertext Transfer Protocol) is the set of rules for transferring files (text, graphic images, sound, video, and other multimedia files) on the World Wide Web.

HTTPS: A web transfer protocol that encrypts and decrypts user page requests as well as the pages that are returned by the Web server. The use of HTTPS protects against eavesdropping and attacks.

Input: The signals or instructions sent to a computer.

Internet: The global collection of computer networks and their connections, all using shared protocols to communicate. A group of computers and servers that are connected to each other.

Iterative: Involving the repeating of a process with the aim of approaching a desired goal, target, or result.

Logic (Boolean): Boolean logic deals with the basic operations of truth values: AND, OR, NOT and combinations thereof.

Loop: A programming structure that repeats a sequence of instructions as long as a specific condition is true.

Looping: Repetition, using a loop. The action of doing something over and over again.

Lossless: Data compression without loss of information.

Lossy: Data compression in which unnecessary information is discarded.

Memory: Temporary storage used by computing devices.

Model: A representation of (some part of) a problem or a system. (Modeling: the act of creating a model.) Note: This definition differs from that used in science.

Nested loop: A loop within a loop, an inner loop within the body of an outer loop.

Network: A group of computing devices (personal computers, phones, servers, switches, routers, and so on) connected by cables or wireless media for the exchange of information and resources.

Operating system: Software that communicates with the hardware and allows other programs to run. An operating system (or "OS") is comprised of system software, or the fundamental files a computer needs to boot up and function. Every desktop computer, tablet, and smartphone includes an operating system that provides basic functionality for the device.

Operation: An action, resulting from a single instruction that changes the state of data.

Packets: Small chunks of information that have been carefully formed from larger chunks of information.

Pair programming: A technique in which two developers (or students) team together and work on one computer. The terms "driver" and "navigator" are often used for the two roles. In a classroom setting, teachers often specify that students switch roles frequently, or within a specific period of time.

Paradigm (programming): A theory or a group of ideas about how something should be done, made, or thought about. A philosophical or theoretical framework of any kind. Common programming paradigms are object-oriented, functional, imperative, declarative, procedural, logic, and symbolic.

Parallelism: The simultaneous execution on multiple processors of different parts of a program.

Parameter: A special kind of variable used in a procedure to refer to one of the pieces of data provided as input to the procedure. These pieces of data are called arguments. An ordered list of parameters is usually included in the definition of a subroutine so each

time the subroutine is called, its arguments for that call can be assigned to the corresponding parameters. An extra piece of information that you pass to a function to customize it for a specific need.

Pattern matching: Finding similarities between things.

Persistence: Trying again and again, even when something is very hard.

Piracy: The illegal copying, distribution, or use of software.

Procedure: An independent code module that fulfills some concrete task and is referenced within a larger body of source code. This kind of code item can also be called a function or a subroutine. 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. A procedure may also be referred to as a function, subroutine, routine, method, or subprogram.

Processor: The hardware within a computer or device that executes a program. The CPU (central processing unit) is often referred to as the brain of a computer.

Program (n): A set of instructions that the computer executes in order to achieve a particular objective. Program (v): To produce a program by programming. An algorithm that has been coded into something that can be run by a machine.

Programming (v): The craft of analyzing problems and designing, writing, testing, and maintaining programs to solve them. The art of creating a program.

Protocol: The special set of rules that end points in a telecommunication connection use when they communicate. Protocols specify interactions between the communicating entities.

Prototype: An early approximation of a final product or information system, often built for demonstration purposes.

Pseudocode: A detailed yet readable description of what a computer program or algorithm must do, expressed in a formally-styled natural language rather than in a programming language.

Remix: making changes to an existing procedure.

RGB: (red, green, and blue) Refers to a system for representing the colors to be used on a computer display. Red, green, and blue can be combined in various proportions to obtain any color in the visible spectrum.

Routing; router; routing: Establishing the path that data packets traverse from source to destination. A device or software that determines the routing for a data packet.

Run program: Cause the computer to execute the commands written in a program.

Security: The protection against access to, or alteration of, computing resources, through the use of technology, processes, and training.

Servers: Computers that exist only to provide things to others.

Simulate: To imitate the operation of a real world process or system over time.

Simulation: Imitation of the operation of a real world process or system over time.

Software: Programs that run on a computer system, computer, or other computing device.

SMTP: A standard protocol for sending emails across the Internet. The communication between mail servers, by default, uses port 25. IMAP: a mail protocol used for accessing email on a remote web server from a local client.

Storage: A place (usually a device) into which data can be entered, in which it can be held, and from which it can be retrieved at a later time. 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.

String: A sequence of letters, numbers, and/or other symbols. A string might represent a name, address, or song title. Some functions commonly associated with strings are length, concatenation, and substring.

Structure: The concept of encapsulation without specifying a particular paradigm.

Subroutine: A callable unit of code, a type of procedure.

Switch: A high-speed device that receives incoming data packets and redirects them to their destination on a local area network (LAN).

System: A collection of elements or components that work together for a common purpose. A collection of computing hardware and software integrated for the purpose of accomplishing shared tasks.

Topology: The physical and logical configuration of a network; the arrangement of a network, including its nodes and connecting links. A logical topology details how devices appear connected to the user. A physical topology is how devices are actually interconnected with wires and cables.

Troubleshooting: A systematic approach to problem solving that is often used to find and resolve a problem, error, or fault within software or a computer system.

User: A person for whom a hardware or software product is designed (as distinguished from the developers).

Variable: A symbolic name that is used to keep track of a value that can change while a program is running. Variables are not just used for numbers. They can also hold text, including whole sentences ("strings"), or the logical values "true" or "false." A variable has a data type and is associated with a data storage location; its value is normally changed during the course of program execution. A placeholder for a piece of information that can change. Note: This definition differs from that used in math.

Wearable computing: Miniature electronic devices that are worn under, with or on top of clothing.

Sources for definitions in this glossary:

CAS-Prim: Computing at School. Computing in the national curriculum: A guide for primary teachers (http://www.computingatschool.org.uk/data/uploads/CASPrimaryComputing.pdf)

Code.org: Creative Commons License (CC BY-NC-SA 4.0) (https://code.org/curriculum/docs/k-5/glossary)

Computer Science Teachers Association: CSTA K–12 Computer Science Standards (2011) https://csta.acm.org/Curriculum/sub/K12Standards.html

FOLDOC: Free On-Line Dictionary of Computing. (http://foldoc.org/)

MA-DLCS: Massachusetts Digital Literacy and Computer Science Standards, Glossary (Draft, December 2015)

NIST/DADS: National Institute of Science and Technology Dictionary of Algorithms and Data Structures. (https://xlinux.nist.gov/dads//)

Techopedia: Techopedia. (https://www.techopedia.com/dictionary)

TechTarget: TechTarget Network. (http://www.techtarget.com/network)

TechTerms: Tech Terms Computer Dictionary. (http://www.techterms.com)

Appendix B-Computer Science Practices

There are seven core practices of computer science. The practices naturally integrate with one another and contain language that intentionally overlaps to illuminate the connections among them. Unlike the core concepts, the practices are not delineated by grade bands. Conversely, like the core concepts, they are meant to build upon each other. (Adapted from: K-12 Computer Science Framework, 2016)

Practices-All practices list skills that students should be able to incorporate by the end of 12th grade
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.
1.1. Include the unique perspectives of others and reflect on one's own perspectives when designing and developing computational products.
1.2. Address the needs of diverse end users during the design process to produce artifacts with broad accessibility and usability.
1.3. Employ self- and peer-advocacy to address bias in interactions, product design, and development methods.
Practice 2. Collaborating Around Computing: Collaborative computing is the process of performing a computational task by working in pairs and on teams. Because it involves asking for the contributions and feedback of others, effective collaboration can lead to better outcomes than working independently. Collaboration requires individuals to navigate and incorporate diverse perspectives, conflicting ideas, disparate skills, and distinct personalities. Students should use collaborative tools to effectively work together and to create complex artifacts.
2.1. Cultivate working relationships with individuals possessing diverse perspectives, skills, and personalities.
2.2. Create team norms, expectations, and equitable workloads to increase efficiency and effectiveness.
2.3. Solicit and incorporate feedback from, and provide constructive feedback to, team members and other stakeholders.
2.4. Evaluate and select technological tools that can be used to collaborate on a project.

Practice 3. Recognizing 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.
3.1. Identify complex, interdisciplinary, real-world problems that can be solved computationally.
3.2. Decompose complex real-world problems into manageable subproblems that could integrate existing solutions or procedures.
3.3. Evaluate whether it is appropriate and feasible to solve a problem computationally.
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.
4.1. Extract common features from a set of interrelated processes or complex phenomena.
4.2. Evaluate existing technological functionalities and incorporate them into new designs.
4.3. Create modules and develop points of interaction that can apply to multiple situations and reduce complexity.
4.4. Model phenomena and processes and simulate systems to understand and evaluate potential outcomes.
Practice 5. Creating Computational Artifacts: The process of developing computational artifacts embraces both creative expression and the exploration of ideas to create prototypes and solve computational problems. Students create artifacts that are personally relevant or beneficial to their community and beyond. Computational artifacts can be created by combining and modifying existing artifacts or by developing new artifacts. Examples of computational artifacts include programs, simulations, visualizations, digital animations, robotic systems, and apps.
5.1. Plan the development of a computational artifact using an iterative process that includes reflection on and modification of the plan, taking into account key features, time and resource constraints, and user expectations.
5.2. Create a computational artifact for practical intent, personal expression, or to address a societal issue.
5.3. Modify an existing artifact to improve or customize it.
Practice 6. Testing 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.

6.1. Systematically test computational artifacts by considering all scenarios and using test cases.
6.2. Identify and fix errors using a systematic process.
6.3. Evaluate and refine a computational artifact multiple times to enhance its performance, reliability, usability, and accessibility.
Practice 7. Communicating About Computing: Communication involves personal expression and exchanging ideas with others. In computer science, students communicate with diverse audiences about the use and effects of computation and the appropriateness of computational choices. Students write clear comments, document their work, and communicate their ideas through multiple forms of media. Clear communication includes using precise language and carefully considering possible audiences.
7.1. Select, organize, and interpret large data sets from multiple sources to support a claim.
7.2. Describe, justify, and document computational processes and solutions using appropriate terminology consistent with the intended audience and purpose.
7.3. Articulate ideas responsibly by observing intellectual property rights and giving appropriate attribution.