Arizona Computer Science Standards

Sixth Grade – Eighth Grade

ARIZONA DEPARTMENT OF EDUCATION

HIGH ACADEMIC STANDARDS FOR STUDENTS

Adopted October 2018

Contents

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

Computing Systems	**Networking and the Internet**	**Data and Analysis**	**Algorithms and Programming**	**Impacts of Computing**
Devices	*Network Communication and Organization*	*Collection*	*Algorithms*	*Culture*
Hardware and Software	*Cybersecurity*	*Storage*	*Variables*	*Social Interactions*
Troubleshooting		*Visualization and Transformation*	*Control*	*Safety, Law, and Ethics*
		Inference and Models	*Modularity*	
			Program Development	

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

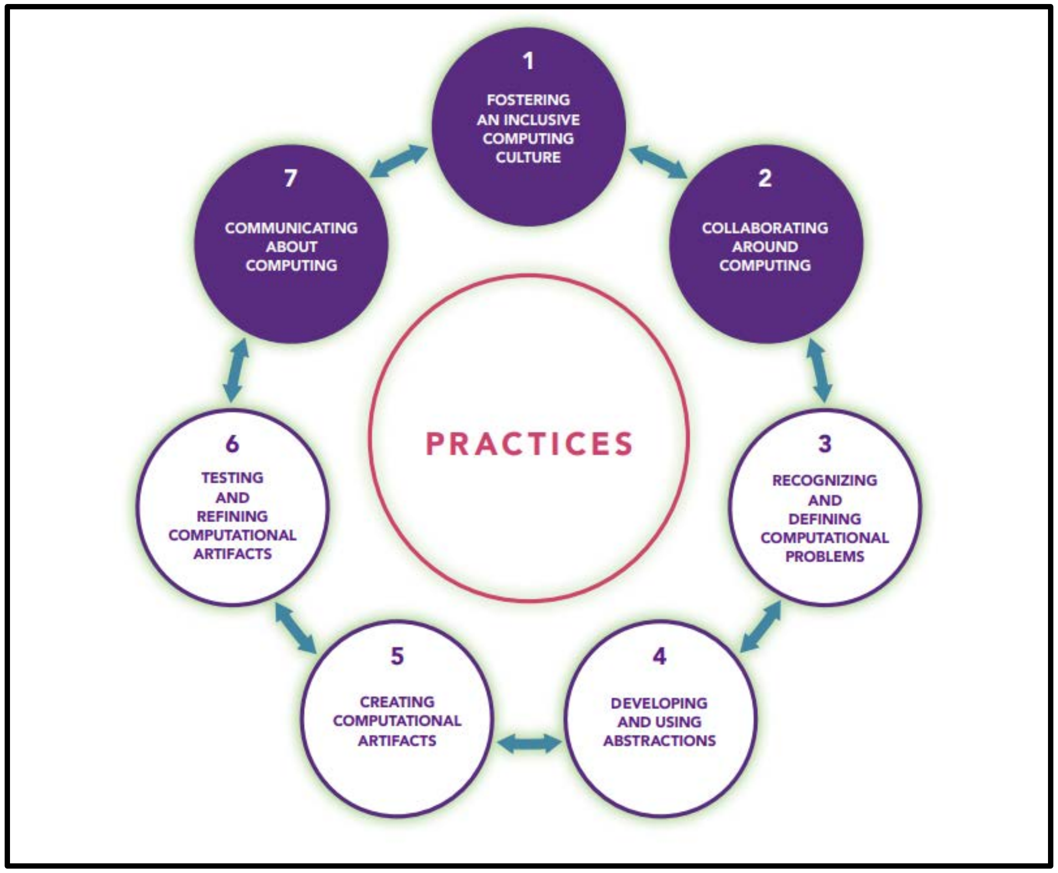
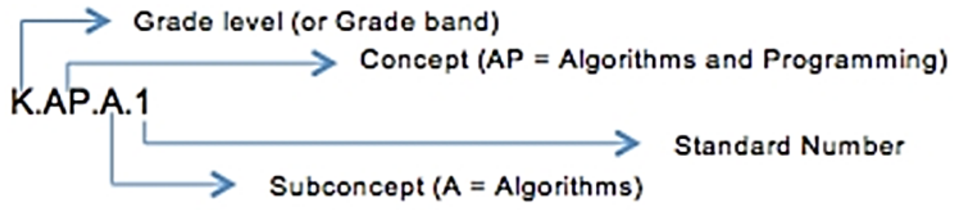

1
FOSTERING AN INCLUSIVE COMPUTING CULTURE
2
COLLABORATING AROUND COMPUTING
3
RECOGNIZING AND DEFINING COMPUTATIONAL PROBLEMS
4
DEVELOPING AND USING ABSTRACTIONS
5
CREATING COMPUTATIONAL ARTIFACTS
6
TESTING AND REFINING COMPUTATIONAL ARTIFACTS
7
COMMUNICATING ABOUT COMPUTING
PRACTICES

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


Grade level (or Grade band)
Concept (AP = Algorithms and Programming)
K.AP.A.1
Standard Number
Subconcept (A = Algorithms)

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

Sixth Grade

Students will deepen their understanding of computing devices as they begin to explore the application of computer science knowledge to real-world problems. The computer science literate student will explore how devices process data and address potential problems. They will investigate the process of data transmission and the need for security concerns. Individually and in small groups they will consider reliability and validity of computational models used to process and represent data. They will also identify possible solutions to programming challenges based on the user's needs. Successful students can implement programming skills using parameters to meet a project's goal and timeline. Computer science literate students will be able to identify the advantages and disadvantages of computing technologies in everyday activities, including bias, accessibility, and privacy.

Concept: Computing Systems (CS)

Subconcept: Devices (D)	
6.CS.D.1	**Compare computing device designs based on how humans interact with them.** *The study of human–computer interaction (HCI) can improve the design of devices, including both hardware and software. Teachers can guide students to consider usability through several lenses. For example, teachers can have students compare computing devices that have different methods of human interaction (keyboard/mouse/trackpad, touchscreen, voice commands, facial recognition/fingerprint sensing, etc)* *Practice(s): Recognizing and Defining Computational Problems: 3.3*
Subconcept: Hardware and Software (HS)	
6.CS.HS.1	**Explain how hardware and software can be used to collect and exchange data.** *Collecting and exchanging data involves input, output, storage, and processing. For example, students can describe how components of a device are used to collect data. Such components might include: accelerometer, Global Position System (GPS), microphone, fingerprint sensor, etc.* Practice(s): Creating Computational Artifacts: 5.1
Subconcept: Troubleshooting (T)	
6.CS.T.1	**Identify problems that can occur in computing devices and their components within a system.** *Since a computing device may interact with interconnected devices within a system, problems may not be due to the computing device itself but to devices or components connected to it. For example, students can discuss why the internet might not be working on their device. It could be airplane mode, no signal (Wi-Fi or mobile data), component malfunction, interference, etc.* Practice(s): Testing and Refining Computational Artifacts: 6.2

Concept: Networks and the Internet (NI)

Subconcept: Cybersecurity (C)	
6.NI.C.1	**Identify multiple methods of encryption to secure the transmission of information.** *Encryption can be as simple as letter substitution or as complicated as modern methods used to secure networks and the Internet. The students will identify different methods of encoding and decoding for encryptions used to hide or secure information. Examples of encryption methods could include: Substitution ciphers (mono-alphabetic or polyalphabetic) and Caesar ciphers.* *Practice(s):* Developing and Using Abstractions: 4.4
6.NI.C.2	**Identify different physical and digital security measures that protect electronic information.** *Information that is stored online is vulnerable to unwanted access. Examples of physical security measures to protect data include keeping passwords hidden, locking doors, making backup copies on external storage devices, and erasing a storage device before it is reused. Examples of digital security measures include secure router admin passwords, firewalls that limit access to private networks, and the use of a protocol such as HTTPS to ensure secure data transmission.* Practice(s): Communicating About Computing: 7.2
Subconcept: Network, Communication, and Organization (NCO)	
6.NI. NCO.1	**Discuss how protocols are used in transmitting data across networks and the Internet.** *Protocols are rules that define how messages are sent between computers. They determine how quickly and securely information is transmitted across networks and the Internet, as well as how to check for and handle errors in transmission. The priority at this level is understanding the purpose of protocols and how they enable secure and errorless communication. Knowledge of the details of how specific protocols work is not expected. For example, students could discuss their protocols or processes for communicating with their friends. They can discuss handshakes, turn-taking, whispering vs yelling, etc. The students can compare these protocols with how computers communicate.* Practice(s): Developing and Using Abstractions: 4.4

Concept: Data and Analysis (DA)

Subconcept: Collection, Visualization and Transformation (CVT)	
6.DA. CVT.1	**Compare different computational tools used to collect, analyze and present data that is meaningful and useful.** *As students continue to explore ways to gather, organize and present data visually to support a claim, they will need to understand when and how to transform data for this purpose. Examples of these computational tools could include Microsoft Excel and Google Sheets.*

	Practice(s): Testing and Refining Computational Artifacts: 6.3
Subconcept: Storage (S)	
6.DA.S.1	**Identify multiple encoding schemes used to represent data, including binary and ASCII.** *Students should explore the same data in multiple ways. For example, students could compare representations of-the same color using binary, RGB values, hex codes (low-level representations), or forms understandable by people, including words, symbols, and digital displays of the color (high-level representations).* Practice(s): Developing and Using Abstractions: 4.0
Subconcept: Inference and Models (IM)	
6.DA. IM.1	**Discuss the validity of a computational model based on the reliability of the data.** *A model may be a programmed simulation of events or a representation of how various data is related. In order to refine a model, students need to consider which data points are relevant, how data points relate to each other, and if the data is accurate. For example, students can discuss how valid a poll (political, social media, student poll) is based on how reliable the data is. Students will discuss if predictions can be made based on the poll.* Practice(s): Creating Computational Artifacts, Developing and Using Abstractions: 5.3, 4.4

Concept: Algorithms and Programming (AP)

Subconcept: Algorithms (A)	
6.AP.A.1	**Identify planning strategies such as flowcharts or pseudocode, to simulate algorithms that solve problems.** *Students should be able to select planning strategies to organize and sequence an algorithm that addresses a problem, even though they may not actually program the solutions. For example, students might express an algorithm that produces a recommendation for purchasing sneakers based on inputs such as size, colors, brand, comfort, and cost.* *Practice(s): Developing and Using Abstractions: 4.4, 4.1*
Subconcept: Variables (V)	
6.AP.V.1	**Identify variables that represent different data types and perform operations on their values.** *A variable is like a container with a name, in which the contents may change, but the name (identifier) does not. When planning and developing programs, students should decide when and how to declare and name new variables. Students should use naming conventions to improve program readability. For example, possible operations include adding points to the score, combining user input with words to make a sentence, changing the size of a picture, or adding a name to a list of people.*

	Practice(s): Creating Computational Artifacts: 5.1, 5.2

Subconcept: Control (C)	
6.AP.C.1	**Design programs that combine control structures, including nested loops and compound conditionals.** *Control structures can be combined in many ways. Nested loops are loops placed within loops. Compound conditionals combine two or more conditions in a logical relationship (e.g., using AND, OR, and NOT), and nesting conditionals within one another allows the result of one conditional to lead to another. For example, when programming an interactive story, students could use a compound conditional within a loop to unlock a door only if a character has a key AND is touching the door.* *Practice(s): Creating Computational Artifacts: 5.1, 5.2*
Subconcept: Modularity (M)	
6.AP.M.1	**Decompose problems into parts to facilitate the design, implementation, and review of programs.** *In order to understand how programs are designed and used, problems should be broken down into smaller pieces that are easier to work with.* *Practice(s): Recognizing and Defining Computational Problems: 3.2*
6.AP.M.2	**Use procedures to organize code and make it easier to reuse.** *Students should compare procedures and/or functions that are used multiple times within a program to repeat groups of instructions. These procedures can be generalized by defining parameters that create different outputs for a wide range of inputs. For example, a procedure to draw a circle involves many instructions, but all of them can be invoked with one instruction, such as "drawCircle." By adding a radius parameter, the user can easily draw circles of different sizes.* *Practice(s): Developing and Using Abstractions: 4.1, 4.3*
Subconcept: Program Development (PD)	
6.AP.PD.1	**Seek and incorporate feedback from team members and users to refine a solution that meets user needs.** *Development teams that employ user-centered design create solutions (e.g., programs and devices) that can have a large societal impact, such as an app that allows people with speech difficulties to translate hard-to-understand pronunciation into understandable language. Students should seek diverse perspectives throughout the design process to improve their computational artifacts. For example, considerations of the end-user may include usability, accessibility, age-appropriate content, respectful language, user perspective, pronoun use, color contrast, and ease of use.* *Practice(s): Collaborating Around Computing, Fostering an Inclusive Computing Culture: 2.3, 1.1*
6.AP.PD.2	**Incorporate existing code into programs and give attribution.** *Building on the work of others enables students to produce more interesting and powerful creations. Students should use portions of code in their own programs and websites. For example, when creating a side-scrolling game, students may incorporate portions of code that create a realistic jump movement from another person's game. They may also import Creative Commons-licensed images to use in the background. Students should give attribution to the original creators to acknowledge their contributions.* *Practice(s): Developing and Using Abstractions, Creating Computational Artifacts, Communicating About Computing: 4.2, 5.2, 7.3*

6.AP.PD.3	**Test programs using a range of inputs and identify expected outputs.** *At this level, testing should become a deliberate process that is more iterative, systematic, and proactive. For example, having students enter data into Microsoft Excel or Google Sheets to see what outputs are produced.* *Practice(s): Testing and Refining Computational Artifacts: 6.1*
6.AP.PD.4	**Maintain a timeline with specific tasks while collaboratively developing computational artifacts.** *Collaboration is a common and crucial practice in program development. Often, many individuals and groups work on the interdependent parts of a project together. For example, students should assume pre-defined roles within their teams and manage the project workflow using structured timelines.* *Practice(s): Collaborating Around Computing: 2.2*
6.AP.PD.5	**Document programs in order to make them easier to follow, test, and debug.** *Documentation allows creators and others to more easily use and understand a program. Students should provide documentation for end users that explains their artifacts and how they function. For example, students could provide a project overview and clear user instructions. They should also incorporate comments into their programs and communicate their process throughout the design, development, and user experience phases.* *Practice(s): Communicating About Computing: 7.2*

Concept: Impacts of Computing (IC)

Subconcept: Culture (C)	
6.IC.C.1	**Identify some of the tradeoffs associated with computing technologies that can affect people's everyday activities and career options.** Advancements in computer technology are neither wholly positive nor negative. However, the ways that people use computing technologies have tradeoffs. Students should consider current events related to broad ideas, including privacy, communication, and automation. For example, driverless cars can increase convenience and reduce accidents, but they are also susceptible to hacking. The emerging industry will reduce the number of taxi and shared-ride drivers, but will create more software engineering and cybersecurity jobs. Practice(s): Communicating About Computing: 7.2
6.IC.C.2	**Identify issues of bias and accessibility in the design of existing technologies.** Students should identify, with teacher's guidance, how various technological tools have different levels of usability. For example, facial recognition software that works better for certain skin tones was likely developed with a homogeneous testing group and could be improved by sampling a more diverse population. For example, ways of improving accessibility of technological tools can include allowing a user to change font sizes and colors. This will make an interface usable for people with low vision and benefits users in situations, such as in bright daylight or a dark room. Practice(s): Fostering an Inclusive Computing Culture: 1.2

Subconcept: Social Interactions (SI)	
6.IC.SI.1	**Identify the advantages of creating a computational product by collaborating with others using digital technologies.** Different digital technologies can be used to gather services, ideas, or content from a large group of people, especially from the online community. It can be done at the local level (e.g., classroom or school) or global level (e.g., age-appropriate online communities). For example, a group of students could combine animations to produce a digital community creation. They could also solicit feedback from many people though use of online communities and electronic surveys. Practice(s): Collaborating Around Computing, Creating Computational Artifacts: 2.4, 5.2
Subconcept: Safety, Law, and Ethics (SLE)	
6.IC. SLE.1	**Describe how some digital information can be public or can be kept private and secure.** Sharing information online can help establish, maintain, and strengthen connections between people. Students should consider current events related to broad ideas, including privacy, communication, and automation. For example, students can discuss how their privacy settings on social media affect who can view their information. Practice(s): Communicating About Computing: 7.2

Seventh Grade

Students will deepen their understanding of computing devices as they continue to explore the application of computer science knowledge to real-world problems. The computer science literate student will evaluate how devices process data and address potential problems through project development. They will investigate the need for security measures and protocols for data transmission. Successful students will be able to integrate reliable and valid computational models to process data that is meaningful and useful. They can evaluate possible solutions to programming challenges based on the user's needs. Students will implement programming skills using parameters to meet a project's goal and timeline. Computer science literate students will be able to compare and contrast possible solutions utilizing computing technologies to solve everyday challenges, taking into consideration bias, accessibility, and privacy.

Concept: Computing Systems (CS)

Subconcept: Devices (D)	
7.CS.D.1	**Identify some advantages, disadvantages, and consequences with the design of computer devices based on an analysis of how users interact with devices.** *The study of human–computer interaction (HCI) can improve the design of devices, including both hardware and software. Teachers can guide students to consider usability through several lenses, including accessibility, ergonomics, and learnability. For example, assistive devices provide capabilities such as scanning written information and converting it to speech.* *Practice(s): Recognizing and Defining Computational Problems: 3.3*
Subconcept: Hardware and Software (HS)	
7.CS.HS.1	**Design projects that combine hardware and software to collect and exchange data.** *Collecting and exchanging data involves input, output, storage, and processing. When possible, students should select the hardware and software components for their project designs by considering factors such as functionality, cost, size, speed, accessibility, and aesthetics. For example, components for a mobile app could include accelerometer, Global Position System (GPS), microphone, fingerprint sensor, etc.* Practice(s): Creating Computational Artifacts: 5.1
Subconcept: Troubleshooting (T)	
7.CS.T.1	**Evaluate strategies to fix problems with computing devices and their components within a system.** *Since a computing device may interact with interconnected devices within a system, problems may not be due to the computing device itself but to devices or components connected to it. For example, troubleshooting strategies include following a troubleshooting flow*

	diagram, making changes to software to see if hardware will work, checking connections and settings, and swapping in working components. Practice(s): Testing and Refining Computational Artifacts: 6.2

Concept: Networks and the Internet (NI)

Subconcept: Cybersecurity (C)	
7.NI.C.1	**Evaluate multiple methods of encryption for the secure transmission of information.** *Encryption can be as simple as letter substitution or as complicated as modern methods used to secure networks and the Internet. The students will examine the different levels of complexity used to hide or secure information. For example, students explore different methods of securing messages using methods such as Caesar ciphers or steganography (i.e., hiding messages inside a picture or other data).* *Practice(s):* Developing and Using Abstractions: 4.4
7.NI.C.2	**Explain how physical and digital security measures protect electronic information.** *Information that is stored online is vulnerable to unwanted access. Examples of physical security measures to protect data include keeping passwords hidden, locking doors, making backup copies on external storage devices, and erasing a storage device before it is reused. For example, digital security measures include secure router admin passwords, firewalls that limit access to private networks, and the use of a protocol such as HTTPS to ensure secure data transmission.* Practice(s): Communicating About Computing: 7.2
Subconcept: Network, Communication, and Organization (NCO)	
7.NI. NCO.1	**Compare and contrast models to understand the many protocols used for data transmission.** *Protocols are rules that define how messages are sent between computers. They determine how quickly and securely information is transmitted across networks and the Internet, as well as how to check for and handle errors in transmission. For example, students should examine how data is sent using protocols to choose the fastest path, to deal with missing information, and to deliver sensitive data securely. The priority at this level is understanding the purpose of protocols and how they enable secure and errorless communication. Knowledge of the details of how specific protocols work is not expected.* Practice(s): Developing and Using Abstractions: 4.4

Concept: Data and Analysis (DA)

Subconcept: Collection, Visualization and Transformation (CVT)	
7.DA. CVT.1	**Collect and analyze data using computational tools to create models that are meaningful and useful.** *As students continue to build on their ability to organize and present data visually to support a claim, they will need to understand when and how to transform data for this purpose. For example, students use computational tools such as Microsoft Excel or Google Sheets to solve a problem that is relevant and meaningful.* Practice(s): Testing and Refining Computational Artifacts: 6.3
Subconcept: Storage (S)	
7.DA.S.1	**Use multiple encoding schemes to represent data, including binary and ASCII.** *Students should represent the same data in multiple ways. For example, students could represent the same color using binary, RGB values, hex codes (low-level representations), as well as forms understandable by people, including words, symbols, and digital displays of the color (high-level representations).* Practice(s): Developing and Using Abstractions: 4.0
Subconcept: Inference and Models (IM)	
7.DA.IM. 1	**Use computational models and determine the reliability and validity of data they generate.** *A model may be a programmed simulation of events or a representation of how various data are related. To refine a model, students need to consider which data points are relevant, how data points relate to each other, and if the data are accurate. For example, students may make a prediction about how far a ball will travel based on a table of data related to the height and angle of a track.* Practice(s): Creating Computational Artifacts, Developing and Using Abstractions: 5.3, 4.4

Concept: Algorithms and Programming (AP)

Subconcept: Algorithms (A)	
7.AP.A.1	**Use planning strategies, such as flowcharts or pseudocode, to develop algorithms to address complex problems.** *Complex problems are problems that would be difficult for students to solve computationally. Students should use pseudocode and/or flowcharts to organize and sequence an algorithm that addresses a complex problem, even though they may not actually program the solutions. For example, students might follow an algorithm that produces a recommendation for purchasing sneakers based on inputs such as size, colors, brand, comfort, and cost.* *Practice(s): Developing and Using Abstractions: 4.4, 4.1*
Subconcept: Variables (V)	

7.AP.V.1	**Compare and contrast variables that represent different data types and perform operations on their values.** *A variable is like a container with a name, in which the contents may change, but the name (identifier) does not. When planning and developing programs, students should decide when and how to declare and name new variables. Students should use naming conventions to improve program readability. For example, possible operations include adding points to the score, combining user input with words to make a sentence, changing the size of a picture, or adding a name to a list of people.* *Practice(s): Creating Computational Artifacts: 5.1, 5.2*

Subconcept: Control (C)	
7.AP.C.1	**Design and develop programs that combine control structures, including nested loops and compound conditionals.** *Control structures can be combined in many ways. Nested loops are loops placed within loops. Compound conditionals combine two or more conditions in a logical relationship (e.g., using AND, OR, and NOT), and nesting conditionals within one another allows the result of one conditional to lead to another. For example, when programming an interactive story, students could use a compound conditional within a loop to unlock a door only if a character has a key AND is touching the door.* *Practice(s): Creating Computational Artifacts: 5.1*
Subconcept: Modularity (M)	
7.AP.M.1	**Decompose problems into parts to facilitate the design, implementation, and review of programs.** *In order to design, implement and evaluate programs students will break down problems into smaller parts. For example, students might code one part of a game at a time (sprites, motion, interaction, backgrounds, etc).* *Practice(s): Recognizing and Defining Computational Problems: 3.2*
7.AP.M.2	**Use procedures with parameters to organize code and make it easier to reuse.** *Students should use procedures and/or functions that are used multiple times within a program to repeat groups of instructions. These procedures can be generalized by defining parameters that create different outputs for a wide range of inputs. For example, a procedure to draw a circle involves many instructions, but all of them can be invoked with one instruction, such as "drawCircle." By adding a radius parameter, the user can easily draw circles of different sizes.* *Practice(s): Developing and Using Abstractions, Creating Computational Artifacts: 4.1, 4.3, 5.1, 5.2*
Subconcept: Program Development (PD)	
7.AP.PD.1	**Seek and incorporate feedback from team members and users to refine a solution that meets user needs.** *Development teams that employ user-centered design create solutions (e.g., programs and devices) that can have a large societal impact, such as an app that allows people with speech difficulties to translate hard-to-understand pronunciation into understandable language. Students should begin to seek diverse perspectives throughout the design process to improve their computational artifacts. Considerations of the end-user may include usability, accessibility, age-appropriate content, respectful language, user perspective, pronoun use, color contrast, and ease of use.* *Practice(s): Collaborating Around Computing, Fostering an Inclusive Computing Culture: 2.3, 1.1*
7.AP.PD.2	**Incorporate existing code and media into programs, and give attribution.** *Building on the work of others enables students to produce more interesting and powerful creations. Students should use portions of code and/or digital media in their own programs and websites. For example, when creating a side-scrolling game, students may incorporate portions of code that create a realistic jump movement from another person's game. They may also import Creative Commons-licensed images to use in the background. Students should give attribution to the original creators to acknowledge their contributions.* *Practice(s): Developing and Using Abstractions, Creating Computational Artifacts, Communicating About Computing: 4.2, 5.2, 7.3*

7.AP.PD.3	**Systematically test and refine programs using a range of possible inputs.** *At this level, testing should become a deliberate process that is more iterative, systematic, and proactive students should begin to test programs by considering potential errors, such as what will happen if a user enters invalid input (e.g., negative numbers and 0 instead of positive numbers).* *Practice(s): Testing and Refining Computational Artifacts: 6.1*
7.AP.PD.4	**Distribute and execute tasks while maintaining a project timeline when collaboratively developing computational artifacts.** *Collaboration is a common and crucial practice in program development. Often, many individuals and groups work on the interdependent parts of a project together. Students should assume pre-defined roles within their teams and manage the project workflow using structured timelines. With teacher guidance, they will begin to create collective goals, expectations, and equitable workloads. For example, students may divide the design stage of a game into planning the storyboard, flowchart, and different parts of the game mechanics. They can then distribute tasks and roles among members of the team and assign deadlines.* *Practice(s): Collaborating Around Computing: 2.2*
7.AP.PD.5	**Document programs to make them easier to follow, test, and debug.** *Documentation allows creators and others to more easily use and understand a program. Students should provide documentation for end users that explains their artifacts and how they function. For example, students could provide a project overview and clear user instructions. They should also incorporate comments into their programs and communicate their process throughout the design, development, and user experience phases.* *Practice(s): Communicating About Computing: 7.2*

Concept: Impacts of Computing (IC)

Subconcept: Culture (C)	
7.IC.C.1	**Explain how some of the tradeoffs associated with computing technologies can affect people's everyday activities and career options.** *Advancements in computer technology are neither wholly positive nor negative. However, the ways that people use computing technologies have tradeoffs. Students should consider current events related to broad ideas, including privacy, communication, and automation. For example, driverless cars can increase convenience and reduce accidents, but they are also susceptible to hacking. The emerging industry will reduce the number of taxi and shared-ride drivers but will create more software engineering and cybersecurity jobs.* Practice(s): Communicating About Computing: 7.2
7.IC.C.2	**Discuss how bias and accessibility issues can impact the functionality of existing technologies.** *Students should discuss the usability of various technology tools (e.g., apps, games, and devices) with the teacher's guidance. For example, facial recognition software that works better for certain skin tones was likely developed with a homogeneous testing group and could be improved by sampling a more diverse population.* Practice(s): Fostering an Inclusive Computing Culture: 1.2

Subconcept: Social Interactions (SI)	
7.IC.SI.1	**Describe the process for creating a computational product by collaborating with others using digital technologies.** *Crowdsourcing can be used as a platform to gather services, ideas, or content from a large group of people, especially from the online community. It can be done at the local level (e.g., classroom or school) or global level (e.g., age-appropriate online communities). For example, a group of students could combine animations to produce a digital community creation. They could also solicit feedback from many people though use of online communities and electronic surveys.* Practice(s): Collaborating Around Computing, Creating Computational Artifacts: 2.4, 5.2
Subconcept: Safety, Law, and Ethics (SLE)	
7.IC. SLE.1	**Identify the benefits and risks associated with sharing information digitally.** Sharing information online can help establish, maintain, and strengthen connections between people. For example, it allows artists and designers to display their talents and reach a broad audience. However, security attacks often start with personal information that is publicly available online. Social engineering is based on tricking people into revealing sensitive information and can be thwarted by being wary of attacks, such as phishing and spoofing. Practice(s): Communicating About Computing: 7.2

Eighth Grade

Students will apply their understanding of computing devices as they implement applications of computer science knowledge to real-world problems. The computer literate student will apply programming skills to process data and address problems using computing devices. They will implement security measures and protocols for data transmission to address vulnerabilities. They can also collect and represent reliable and valid computational models. Successful students can integrate possible solutions to programming challenges based on the user's needs. They will achieve this by implementing programming skills using parameters to meet a project's goal and timeline. Computer science literate students will be able to utilize computing technologies and develop possible solutions to solve everyday challenges, taking into consideration bias, accessibility, and privacy.

Concept: Computing Systems (CS)

Subconcept: Devices (D)	
8.CS.D.1	**Improve the design of computing devices based on an analysis of how users interact them, and consider unintended consequences.** *The study of human–computer interaction (HCI) can improve the design of devices, including both hardware and software. Students should make recommendations for existing devices (e.g., a laptop, phone, or tablet) or design their own components or interface (e.g., create their own controllers). Teachers can guide students to consider usability through several lenses, including accessibility, ergonomics, and learnability. For example, assistive devices provide capabilities such as scanning written information and converting it to speech.* *Practice(s): Recognizing and Defining Computational Problems: 3.3*
Subconcept: Hardware and Software (HS)	
8.CS.HS.1	**Design and evaluate projects that combine hardware and software components to collect and exchange data.** *Collecting and exchanging data involves input, output, storage, and processing. When possible, students should select the hardware and software components for their project designs by considering factors such as functionality, cost, size, speed, accessibility, and aesthetics. For example, components for a mobile app could include: accelerometer, GPS, and speech recognition. The choice of a device that connects wirelessly through a Bluetooth connection versus a physical USB connection involves a tradeoff between mobility and the need for an additional power source for the wireless device.* Practice(s): Creating Computational Artifacts: 5.1

Subconcept: Troubleshooting (T)	
8.CS.T.1	**Systematically identify and develop strategies to fix problems with computing devices and their components.** *Since a computing device may interact with interconnected devices within a system, problems may not be due to the computing device itself but to devices or components connected to it. Just as pilots use checklists to troubleshoot problems with aircraft systems, students should use a similar, structured process to troubleshoot problems with computing systems and ensure that potential solutions are not overlooked. Examples of troubleshooting strategies include following a troubleshooting flow diagram, making changes to software to see if hardware will work, checking connections and settings, and swapping in working components.* Practice(s): Testing and Refining Computational Artifacts: 6.2

Concept: Networks and the Internet (NI)

Subconcept: Cybersecurity (C)	
8.NI.C.1	**Apply multiple methods of encryption to model the secure transmission of information.** Encryption can be as simple as letter substitution or as complicated as modern methods used to secure networks and the Internet. Students should encode and decode messages using a variety of encryption methods, and they should understand the different levels of complexity used to hide or secure information. For example, students could secure messages using methods such as Caesar cyphers or steganography (i.e., hiding messages inside a picture or other data). They can also model more complicated methods, such as public key encryption, through unplugged activities. *Practice(s):* Developing and Using Abstractions: 4.4
8.NI.C.2	**Evaluate how various physical and digital security measures protect electronic information and how a lack of such measures could lead to vulnerabilities.** Information that is stored online is vulnerable to unwanted access. Examples of physical security measures to protect data include keeping passwords hidden, locking doors, making backup copies on external storage devices, and erasing a storage device before it is reused. Examples of digital security measures include secure router admin passwords, firewalls that limit access to private networks, and the use of a protocol such as HTTPS to ensure secure data transmission. Examples of vulnerabilities include password strength, awareness of how data is used, as well as threats to personal and professional data. Practice(s): Communicating About Computing: 7.2

Subconcept: Network, Communication, and Organization (NCO)	
8.NI. NCO.1	**Develop models to illustrate the role of protocols in transmitting data across networks and the Internet.** Protocols are rules that define how messages are sent. They determine how quickly and securely information is transmitted across networks and the Internet, as well as how to check for and handle errors in transmission. Students should model how data is sent using protocols to choose the fastest path, to deal with missing information, and to deliver sensitive data securely. For example, students can be given a data transmission scenario and asked to determine which protocol should be used and why. The priority at this level is understanding the purpose of protocols and how they enable secure and errorless communication. Knowledge of the details of how specific protocols work is not expected. Practice(s): Developing and Using Abstractions: 4.4

Concept: Data and Analysis (DA)

Subconcept: Collection, Visualization and Transformation (CVT)	
8.DA. CVT.1	**Collect data using computational tools and transform the data to make it more meaningful and useful.** *As students continue to build on their ability to organize and present data visually to support a claim, they will need to understand when and how to transform data for this purpose. Students should transform data to remove errors, highlight or expose relationships, and/or make it easier for computers to process. Data cleaning is an important transformation for ensuring consistent format and reducing noise and errors (e.g., removing irrelevant responses in a survey). An example of a transformation that highlights a relationship is representing males and females as percentages of a whole instead of as individual counts.* Practice(s): Testing and Refining Computational Artifacts: 6.3
Subconcept: Storage (S)	
8.DA.S.1	**Represent data using multiple encoding schemes including binary and ASCII.** *Data representations occur at multiple levels of abstraction, from the physical storage of bits to the arrangement of information into organized formats (e.g., tables). Students should represent the same data in multiple ways. For example, students could represent the same color using binary, RGB values, hex codes (low-level representations), as well as forms understandable by people, including words, symbols, and digital displays of the color (high-level representations).* Practice(s): Developing and Using Abstractions: 4.0

Subconcept: Inference and Models (IM)	
8.DA.IM.1	**Design computational models and evaluate them based on the reliability and validity of the data they generate.** *A model may be a programmed simulation of events or a representation of how various data is related. To refine a model, students need to consider which data points are relevant, how data points relate to each other, and if the data is accurate. For example, students may make a prediction about how far a ball will travel based on a table of data they designed related to the height and angle of a track. The students could then test and refine their model by comparing predicted versus actual results and considering whether other factors are relevant (e.g., size and mass of the ball). Additionally, students could refine game mechanics based on tests to make the game more balanced or fair.* Practice(s): Creating Computational Artifacts, Developing and Using Abstractions: 5.3, 4.4

Concept: Algorithms and Programming (AP)

Subconcept: Algorithms (A)	
8.AP.A.1	**Develop planning strategies, such as flowcharts or pseudocode, to develop algorithms to address complex problems.** *Complex problems are problems that would be difficult for students to solve computationally. Students should use pseudocode and/or flowcharts to organize and sequence an algorithm that addresses a complex problem, even though they may not actually program the solutions. For example, students might express an algorithm that produces a recommendation for purchasing sneakers based on inputs such as size, colors, brand, comfort, and cost. Testing the algorithm with a wide range of inputs and users allows students to refine their recommendation algorithm and to identify other inputs they may have initially excluded.* *Practice(s): Developing and Using Abstractions: 4.4, 4.1*
Subconcept: Variables (V)	
8.AP.V.1	**Create named variables that represent different data types and perform operations on their values.** *A variable is like a container with a name, in which the contents may change, but the name (identifier) does not. When planning and developing programs, students should decide when and how to declare and name new variables. Students should use naming conventions to improve program readability. Examples of operations include adding points to the score, combining user input with words to make a sentence, changing the size of a picture, or adding a name to a list of people.* *Practice(s): Creating Computational Artifacts: 5.1, 5.2*

Subconcept: Control (C)	
8.AP.C.1	**Design and iteratively develop programs that combine control structures, including nested loops and compound conditionals.** *Control structures can be combined in many ways. Nested loops are loops placed within loops. Compound conditionals combine two or more conditions in a logical relationship (e.g., using AND, OR, and NOT), and nesting conditionals within one another allows the result of one conditional to lead to another. For example, when programming an interactive story, students could use a compound conditional within a loop to unlock a door only if a character has a key AND is touching the door.* *Practice(s): Creating Computational Artifacts: 5.1, 5.2*
Subconcept: Modularity (M)	
8.AP.M.1	**Decompose problems into parts to facilitate the design, implementation, and review of programs.** In order to design, implement and evaluate programs, students will break down problems into smaller parts. For example, students might code one part of a game at a time (sprites, motion, interaction, backgrounds, etc). *Practice(s): Recognizing and Defining Computational Problems: 3.2*
8.AP.M.2	**Create procedures with parameters to organize code and make it easier to reuse.** *Students should create procedures and/or functions that are used multiple times within a program to repeat groups of instructions. These procedures can be generalized by defining parameters that create different outputs for a wide range of inputs. For example, a procedure to draw a circle involves many instructions, but all of them can be invoked with one instruction, such as "drawCircle." By adding a radius parameter, the user can easily draw circles of different sizes.* *Practice(s): Developing and Using Abstractions: 4.1, 4.3*
Subconcept: Program Development (PD)	
8.AP.PD.1	**Seek and incorporate feedback from team members and users to refine a solution that meets user needs.** *Development teams that employ user-centered design create solutions (e.g., programs and devices) that can have a large societal impact, such as an app that allows people with speech difficulties to translate hard-to-understand pronunciation into understandable language. Students should begin to seek diverse perspectives throughout the design process to improve their computational artifacts. Considerations of the end-user may include usability, accessibility, age-appropriate content, respectful language, user perspective, pronoun use, color contrast, and ease of use.* *Practice(s): Collaborating Around Computing, Fostering an Inclusive Computing Culture: 2.3, 1.1*
8.AP.PD.2	**Incorporate existing code, media, and libraries into original programs, and give attribution.** *Building on the work of others enables students to produce more interesting and powerful creations. Students should use portions of code, algorithms, and/or digital media in their own programs and websites. At this level, they may also import libraries and connect to web application program interfaces (APIs). For example, when creating a side-scrolling game, students may incorporate portions of code that create a realistic jump movement from another person's game. They may also import Creative Commons-licensed images to use in the background. Students should give attribution to the original creators to acknowledge their contributions.* *Practice(s): Developing and Using Abstractions, Creating Computational Artifacts, Communicating About Computing: 4.2, 5.2, 7.3*

8.AP.PD.3	**Systematically test and refine programs using a range of possible inputs.** *At this level, testing should become a deliberate process that is more iterative, systematic, and proactive. Students should begin to test programs by considering potential errors, such as what will happen if a user enters invalid input (e.g., negative numbers and 0 instead of positive numbers).* *Practice(s): Testing and Refining Computational Artifacts: 6.1*
8.AP.PD.4	**Distribute and execute tasks while maintaining a project timeline when collaboratively developing computational artifacts.** *Collaboration is a common and crucial practice in program development. Often, many individuals and groups work on the interdependent parts of a project together. Students should assume pre-defined roles within their teams and manage the project workflow using structured timelines. With teacher guidance, they will begin to create collective goals, expectations, and equitable workloads. For example, students may divide the design stage of a game into planning the storyboard, flowchart, and different parts of the game mechanics. They can then distribute tasks and roles among members of the team and assign deadlines.* *Practice(s): Collaborating Around Computing: 2.2*
8.AP.PD.5	**Document programs to make them easier to follow, test, and debug.** *Documentation allows creators and others to more easily use and understand a program. Students should provide documentation for end users that explains their artifacts and how they function. For example, students could provide a project overview and clear user instructions. They should also incorporate comments into their programs and communicate their process throughout the design, development, and user experience phases.* *Practice(s): Communicating About Computing: 7.2*

Concept: Impacts of Computing (IC)

Subconcept: Culture (C)	
8.IC.C.1	**Compare and contrast tradeoffs associated with computing technologies that affect people's everyday activities and career options.** Advancements in computer technology are neither wholly positive nor negative. However, the ways that people use computing technologies have tradeoffs. Students should consider current events related to broad ideas, including privacy, communication, and automation. For example, driverless cars can increase convenience and reduce accidents, but they are also susceptible to hacking. The emerging industry will reduce the number of taxi and shared-ride drivers, but will create more software engineering and cybersecurity jobs. Practice(s): Communicating About Computing: 7.2

8.IC.C.2	**Develop a solution to address an issue of bias or accessibility in the design of existing technologies.** Students should test and discuss the usability of various technology tools (e.g., apps, games, and devices) with the teacher's guidance. For example, facial recognition software that works better for certain skin tones was likely developed with a homogeneous testing group and could be improved by sampling a more diverse population. When discussing accessibility, students may notice that allowing a user to change font sizes and colors will not only make an interface usable for people with low vision but also benefits users in various situations, such as in bright daylight or a dark room. Practice(s): Fostering an Inclusive Computing Culture: 1.2
Subconcept: Social Interactions (SI)	
8.IC.SI.1	**Collaborate with contributors by using digital technologies when creating a computational product.** Crowdsourcing can be used as a platform to gather services, ideas, or content from a large group of people, especially from the online community. It can be done at the local level (e.g., classroom or school) or global level (e.g., age-appropriate online communities). For example, a group of students could combine animations to produce a digital community creation. They could also solicit feedback from many people though use of online communities and electronic surveys. Practice(s): Collaborating Around Computing, Creating Computational Artifacts: 2.4, 5.2
Subconcept: Safety, Law, and Ethics (SLE)	
8.IC. SLE.1	**Evaluate the benefits and risks associated with sharing information digitally.** Sharing information online can help establish, maintain, and strengthen connections between people. For example, it allows artists and designers to display their talents and reach a broad audience. However, security attacks often start with personal information that is publicly available online. Social engineering is based on tricking people into revealing sensitive information and can be thwarted by being wary of attacks, such as phishing and spoofing. For example, students could brainstorm reasons why individuals would want to share information online and the potential risks of doing so. Practice(s): Communicating About Computing: 7.2

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.