

Utah 6-12 Computer Science Standards Fall 2019

DRAFT

Utah State Board of Education
ADA Compliant: December 2019

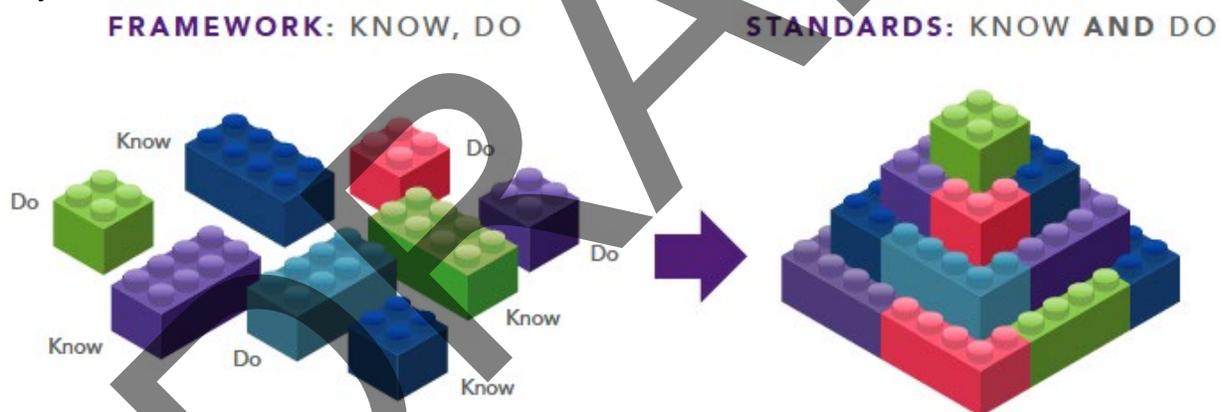


Introduction:

The Utah State Board of Education (USBE) formed a Computer Science Taskforce (Taskforce) to establish a Vision of Computer Science in the Public Education System. The Taskforce met multiple times to identify a strategic plan of recommendations to successfully carry out computer science education within the K-12 schools. In June 2018, the Taskforce's strategic priorities (steps) to accomplish the vision were presented to and subsequently accepted by the USBE. The priorities are as follows:

- Develop and implement statewide K-12 framework for computer science.
- Start early by engaging students at the elementary level.
- Develop a statewide strategy to communicate the value of computer science.
- Build capacity among educators at pre-service and in-service levels.
- Improve upon current course requirements to scaffold computer science learning K-12.
- Regardless of location, ensure students can access a majority of the 30+ computer science and IT courses currently offered (popular courses include Computer Programming 1, Computer Science Principles, Exploring Computer Science, Web Development 1, and Game Development).

The first strategic priority, *Utah Computer Science K-12 Framework¹*, has been developed and approved by the USBE. Implementation of the framework is the next step within the strategic priority.



Utah teachers, principals, district leaders, and university professors worked with the USBE in March 2019 to develop K-5 Computer Science Standards. The writers used the Utah Computer Science K-12 Framework² and the K-12 Computer Science Framework³ to identify important concepts and practices to inform the creation of standards for each grade level.

¹ Utah Computer Science K-12 Framework. (October 2018). Retrieved from: <https://schools.utah.gov/file/46d4ca37-9d23-414e-91fd-6640b6be9df6>

² Utah Computer Science K-12 Framework. (October 2018). Retrieved from: <https://schools.utah.gov/file/46d4ca37-9d23-414e-91fd-6640b6be9df6>

³ K-12 Computer Science Framework. (October 2016) Retrieved from: <https://k12cs.org/wp-content/uploads/2016/09/K%E2%80%9312-Computer-Science-Framework.pdf>

Utah Computer Science Vision Statement:

Each student in secondary public schools will have access to robust and varied computer science courses by 2022. All students will enter secondary schools with exposure to computational thinking and competencies in digital literacy. This begins in our elementary schools with competencies in keyboarding, appropriate and responsible use of technology, and basic coding principles.

Organization of Standards:

The Utah 6-12 Computer Science standards are organized into strands, which represent significant areas of learning within content areas. Within each strand are standards. A standard is an articulation of the demonstrated proficiency to be obtained. A standard represents an essential element of the learning that is expected. While some standards within a strand may be more comprehensive than others, all standards are essential for mastery.

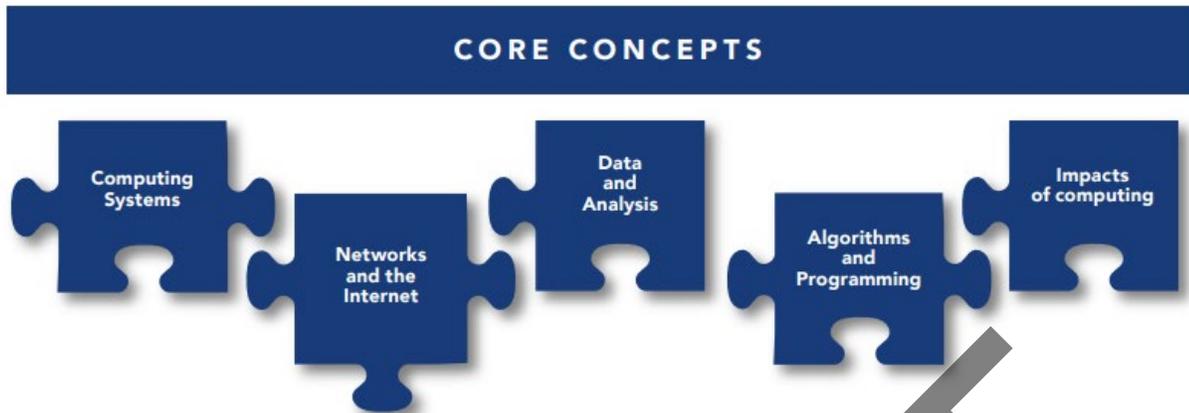
Within the standards there are words that are bold, underlined, and in green text. For example, look at Standard 6.NI.1.

Standard 6.NI.1 Explain potential security threats and practice practical measures to reduce these threats.

(Practice 4: Developing and Using Abstractions)

Green text demonstrates an alignment to the practice language that is highlighted at the conclusion of each standard.

The bold and underlined text, such as security threats, is included in the standards glossary at the conclusion of the document.

Strand Language⁴**Computing Systems (CS):**

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

Network and the Internet (NI):

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

⁴ K-12 Computer Science Framework. (October 2016) Retrieved from: <https://k12cs.org/wp-content/uploads/2016/09/K%E2%80%9312-Computer-Science-Framework.p>

Data and Analysis (DA):

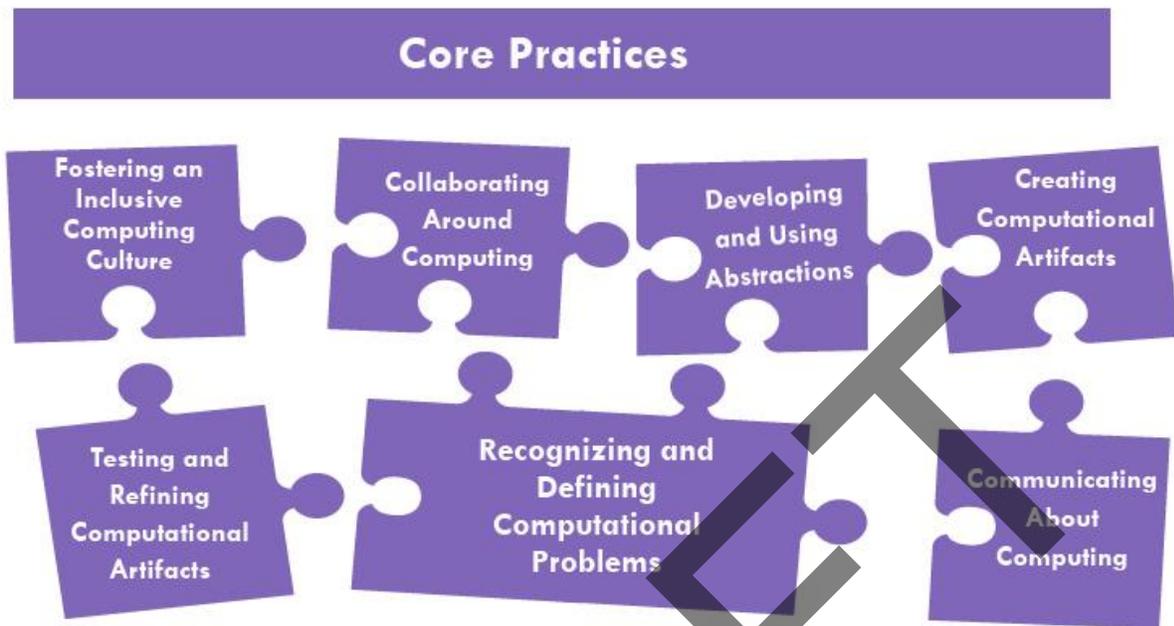
Computing systems exist to process data. The amount of digital data generated in the world is rapidly expanding, and the need to process data effectively is increasingly important. Data is collected and stored so it can be analyzed to better understand the world and make more accurate predictions.

Algorithms and Programming (AP):

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

Impacts of Computing (IC):

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

Practice Language⁵:**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, backgrounds, and abilities. Incorporating these perspectives involves understanding the personal, ethical, social, economic, and cultural contexts in which people operate. Considering the needs of diverse users during the design process is essential to producing inclusive computational products.

By the end of Grade 12, students should be able to:

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

At all grade levels, students should recognize that the choices people make when they create artifacts are based on personal interests, experiences, and needs. Students who are well-versed in fostering an inclusive computing culture should be able to differentiate backgrounds and skill sets and know when to call upon others, such as to seek out knowledge about potential end users or intentionally seek input from people with diverse backgrounds.

2. Address the needs of diverse end users during the design process to produce artifacts with broad accessibility and usability.

At any level, students should recognize that users of technology have different needs and preferences and that not everyone chooses to use, or is able to use, the same technology products. At the higher grades, students should become aware of professionally accepted accessibility standards and should be

⁵ K-12 Computer Science Framework. (October 2016) Retrieved from: <https://k12cs.org/wp-content/uploads/2016/09/K%E2%80%9312-Computer-Science-Framework.pdf>

able to evaluate computational artifacts for accessibility. Students should also begin to identify potential bias during the design process to maximize accessibility in product design. For example, they can test an app and recommend to its designers that it respond to verbal commands to accommodate users who are blind or have physical disabilities.

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

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

Practice 2: Collaborating Around Computing

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

By the end of Grade 12, students should be able to:

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

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

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

After students have had experience cultivating working relationships within teams, they should gain experience working in particular team roles. As students' progress, they should become less dependent on the teacher assigning roles and become more adept at assigning roles within their teams. For example, they should decide together how to take turns in different roles. Eventually, students should independently organize their own teams and create common goals, expectations, and equitable workloads. They should also manage project workflow using agendas and timelines and should evaluate workflow to identify areas for improvement.

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

At any level, students should ask questions of others and listen to their opinions. As they progress in school, students should provide and receive feedback related to computing in constructive ways. For example, pair programming is a collaborative process that promotes giving and receiving feedback. Older

students should engage in active listening by using questioning skills and should respond empathetically to others. As they progress, students should be able to receive feedback from multiple peers and should be able to differentiate opinions. Eventually, students should seek contributors from various environments. These contributors may include end users, experts, or general audiences from online forums.

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

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

Practice 3: Recognizing and Defining Computational Problems

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

By the end of Grade 12, students should be able to:

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

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

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

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

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

After students have had some experience breaking problems down and identifying subproblems that can be solved computationally, they should begin to evaluate whether a computational solution is the most

appropriate solution for a specific problem. For example, students might question whether using a computer to determine whether someone is telling the truth would be advantageous. As students' progress, they should systematically evaluate the feasibility of using computational tools to solve given problems or subproblems, such as through a cost-benefit analysis. Eventually, students should include more factors in their evaluations, such as how efficiency affects feasibility or whether a proposed approach raises ethical concerns.

Practice 4: Developing and Using Abstractions

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

By the end of Grade 12, students should be able to:

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

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

2. Evaluate existing technological functionalities and incorporate them into new designs.

At all levels, students should be able to use well defined abstractions that hide complexity. Just as a car hides operating details, such as the mechanics of the engine, a computer program's "move" command relies on hidden details that cause an object to change location on the screen. As they progress, students should incorporate predefined functions into their designs, understanding that they do not need to know the underlying implementation details of the abstractions that they use. Eventually, students should understand the advantages of, and be comfortable using, existing functionalities (abstractions) including technological resources created by other people, such as libraries and application programming interfaces (APIs). Students should be able to evaluate existing abstractions to determine which should be incorporated into designs and how they should be incorporated. For example, students could build powerful apps by incorporating existing services, such as online databases that return geolocation coordinates of street names or food nutrition information.

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

After students have had some experience identifying patterns, decomposing problems, using abstractions, and taking advantage of existing resources, they should begin to develop their own abstractions. As they progress, students should take advantage of opportunities to develop generalizable modules. For example, students could write more efficient programs by designing procedures that are used multiple times in the program. These procedures can be generalized by defining parameters that create different outputs for a wide range of inputs. Later, students should be able to design systems of interacting modules, each with a well-defined role, that coordinate to accomplish a common goal. Within an object-oriented programming context, module design may include defining the interactions among objects. At this stage, these modules, which combine both data and procedures, can be designed and

documented for reuse in other programs. Additionally, students can design points of interaction, such as a simple user interface, either text or graphical, that reduces the complexity of a solution and hides lower level implementation details.

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

Students at all grade levels should be able to represent patterns, processes, or phenomena. As they progress, students should understand that computers can model real-world phenomena, and they should use existing computer simulations to learn about real-world systems. For example, they may use a preprogrammed model to explore how parameters affect a system, such as how rapidly a disease can spread. Older students should model phenomena as systems, with rules governing the interactions within the system. Students should analyze and evaluate these models against real-world observations. For example, students might create a simple producer–consumer ecosystem model using a programming tool. Eventually, they could progress to creating more complex and realistic interactions between species, such as predation, competition, or symbiosis, and evaluate the model based on data gathered from nature.

Practice 5: Creating Computational Artifacts

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

By the end of Grade 12, students should be able to:

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

At any grade level, students should participate in project planning and the creation of brainstorming documents. As learning progresses, students should systematically plan the development of a program or artifact and intentionally apply computational techniques, such as decomposition and abstraction, along with knowledge about existing approaches to artifact design. Students should be capable of reflecting on and, if necessary, modifying the plan to accommodate end goals.

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

Students at all grade levels should develop artifacts in response to a task or a computational problem. As they progress, student expressions should become more complex and of increasingly broader significance. Eventually, students should engage in independent, systematic use of design processes to create artifacts that solve problems with social significance by seeking input from broad audiences.

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

At all grade levels, students should be able to examine existing artifacts to understand what they do. As they progress, students should attempt to use existing solutions to accomplish a desired goal. For example, students could attach a programmable light sensor to a physical artifact they have created to

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

Practice 6: Testing and Refining Computational Artifacts

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

By the end of Grade 12, students should be able to:

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

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

2. Identify and fix errors using a systematic process.

At any grade level, students should be able to identify and fix errors in programs (debugging) and use strategies to solve problems with computing systems (troubleshooting). As students' progress, they should become more adept at debugging programs and begin to consider logic errors: cases in which a program works, but not as desired. In this way, students will examine and correct their own thinking. For example, they might step through their program, line by line, to identify a loop that does not terminate as expected. Eventually, older students should progress to using more complex strategies for identifying and fixing errors, such as printing the value of a counter variable while a loop is running to determine how many times the loop runs.

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

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

Practice 7: Communicating About Computing

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

carefully considering possible audiences.

By the end of Grade 12, students should be able to:

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

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

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

At any grade level, students should be able to talk about choices they make while designing a computational artifact. Students should provide documentation for end users that explains their artifacts and how they function, and they should both give and receive feedback. For example, students could provide a project overview and ask for input from users. As students' progress, they should incorporate clear comments in their product and document their process using text, graphics, presentations, and demonstrations.

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

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

DRAFT

Grade 6

Computing Systems (CS):

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

Standard 6.CS.1 Utilize troubleshooting strategies to resolve hardware and software issues in a logical order. (*Practice 4: Developing and Using Abstractions*)

Students will be able to utilize a step-by-step approach to identify and resolve problems with hardware and software. For example, a checklist can be used to ensure that possible solutions are not overlooked such as checking for writing conventions before finalizing a writing assignment. Students may refer to the order of operations when solving a math equation. Students may search for technical information online when solving problems. A flow diagram may be used to determine possible next steps.

Network and the Internet (NI):

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

Standard 6.NI.1 Explain potential security threats and practice protective measures to reduce these threats. (*Practice 4: Developing and Using Abstractions*)

Students will recognize and explain the existence of threats and protect their personal information using appropriate security measures. Students identify multiple methods for protecting their data and articulate the value and appropriateness for each method. For example, students should develop habits such as logging off devices and maintaining hidden, strong, evolving passphrases. Also, understanding how to use cybersecurity to protect personal and sensitive data.

Data and Analysis (DA):

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

Standard 6.DA.1 Represent a single data set in multiple ways using words, symbols, manipulatives, charts, diagrams, and visuals. (*Practice 4: Developing and Using Abstractions.*)

Students will represent data in multiple ways using abstraction. For example, convert letters into binary

code, location into GPS coordinates or ideas and phrases into emojis. Students may represent a location as a string “New Zealand” or a numeric input (longitude/latitude geolocation). Another example could be representing colors using binary, hexadecimal, or words.

Algorithms and Programming (AP):

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

Standard 6.AP.1 Design and illustrate **algorithms** to efficiently solve complex problems by utilizing **pseudocode** and/or other descriptive methods. (*Practice 3: Recognizing and defining computational problems*)

Students will decompose or design algorithms (how to instructions) utilizing pseudocode to solve complex problems. Students will be able to decompose a real-world problem and illustrate the decision-making process in a well-organized flowchart, storyboard, ordered directions, notations, or other method. For example, the students might create a flowchart to illustrate which equipment to use for recess based on the weather, play preference, and a student’s energy level.

Standard 6.AP.2 Create naming conventions for **variables** that support the debugging process and incorporate these **variables** into a simple program. (*Practice 7: Communicating about Computing*)

To make the debugging process easier, students will create and name variables that store data in a meaningful and logical way. For example, when writing an algorithm, students will incorporate names based on the command function such as use the variable “turn” to describe direction, “loop” for repeating tasks.

Standard 6.AP.3 Annotate programs in order to document their use and improve readability, testing, and **debugging**. (*Practice 7: Communicating about computing*)

Students will annotate by adding descriptors, comments or notations to describe a program for future use and easier debugging. For example, students could add comments to describe the functionality of different segments of code. These annotations are like those in textbooks and instruction manuals or note-taking on a presentation slide.

Impacts of Computing (IC):

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

Standard 6.IC.1 Recognize and discuss issues of **bias** and **accessibility** in existing technologies. (*Practice 1: Fostering an inclusive computer culture. Practice 7: Communicating about computing.*)

Students will be able to recognize and discuss the usability and accessibility of various technology tools such as apps, games, and devices acknowledging designer bias. For example, students could discuss if devices in their school are ADA compliant and whether software they use has been designed for a particular user or a diverse population.

DRAFT

Grade 7

Computing Systems (CS):

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

Standard 7.CS.1 Design modifications to computing devices in order to improve the ways users interact with the devices. (*Practice 3: Recognizing and Defining Computational Problems.*)

Students will be able to identify problems with existing computing devices or technologies and design modifications to improve the ways users interact with those technologies. For example, students may design changes to an existing device in order to improve accessibility for users with visual, audio, physical, language and/or other barriers or students may redesign an existing computing device to be more functional for an everyday user.

Network and the Internet (NI):

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

Standard 7.NI.1 Model the role of protocols in transmitting data across networks and the Internet. (*Practice 7: Communication about Computing*)

Students will model how protocols such as HTTP and TCP/IP allow for the transmission of data across networks and the internet. For example, students will participate in a role play and physically act out the data transmission process following protocols (a set of rules).

Data and Analysis (DA):

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

Standard 7.DA.1 Collect data using computational tools and transform the data to make it more useful. (*Practice 2: Collaborating about Computing.*)

Students will use computational tools to collect and transform data in a real-world scenario or applications. For example, students will use a Microbit circuit board to collect temperatures, soil moisture levels, etc. and use a program/app to create a data visualization. Additionally, students may create and administer a survey in a social studies class to aggregate data on a pertinent topic and then create a chart or graph to better display the data.

Algorithms and Programming (AP):

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

Standard 7.AP.1 Design and **iteratively develop programs** that combine **control structures**. (*Practice 5: Creating Computational Artifacts; Practice 6: Testing and Refining Computational Artifacts*)

Students will design, develop, test, and refine programs using control structures such as loops or conditional logic statements. For example, students will create a choose your own adventure story/presentation, a flowchart, or code a simple interactive game or animation.

Standard 7.AP.2 Seek and **incorporate** feedback from team members and users to **refine** a solution to a programming project that meets the user's needs. (*Practice 2: Collaborating Around Computing; Practice 6: Testing and Refining Computational Artifacts.*)

Students will collaborate to seek and incorporate feedback from team members on a team project and use that feedback to refine their project to meet the needs of all users. For example, students will solicit feedback from others on a programming project to improve the quality of their work and meet the needs of all users.

Standard 7.AP.3 Systematically **test** and **refine programs** using a range of test cases. (*Practice 6: Testing and Refining Computational Artifacts.*)

Students will use a variety of problem-solving processes such as the engineering design process, decision matrix, pros and cons, or DMAIC (define, measure, analyze, improve and control) to test and refine a project or program. Students will test and refine a computer program, an engineering artifact, or solution. For example, students may test and refine a math program solving for surface area of different shapes (triangles, quadrilaterals, polygons, cubes).

Standard 7.AP.4 Select and **assign** tasks to maintain a project timeline when collaboratively developing **computational artifacts**. (*Practice 2: Collaborating Around Computing. Practice 5: Creating Computational Artifacts.*)

Students will select, assign, and manage tasks within a project timeline of milestones and due dates while collaboratively working on projects. For example, students will use tools such as storyboards, to-do lists, team roles, and other project management tools to organize their projects and share the work across team members and help them be more efficient in managing time and resources.

Impacts of Computing (IC):

Computing affects many aspects of the world in both positive and negative ways at local, national, and global levels. Individuals and communities influence computing through their behaviors and cultural and social interactions, and in turn, computing

influences new cultural practices. An informed and responsible person should understand the social implications of the digital world, including equity and access to computing.

Standard 7.IC.1 Compare tradeoffs associated with **computing technologies** that affect people's everyday activities and career options. (*Practice 1: Fostering an Inclusive Computing Culture; Practice 7: Communicating about Computing.*)

Advancements in computer technology have trade-offs. Students will 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.⁶

DRAFT

⁶ Kansas Computer Science Standards Grades P-12. (2019) Retrieved from: [https://www.ksde.org/Portals/0/CSAS/Content%20Area%20\(A-E\)/Computer%20Science/Kansas%20Computer%20Science%20Model%20Standards%20with%20Description.pdf?ver=2019-04-23-165056-093](https://www.ksde.org/Portals/0/CSAS/Content%20Area%20(A-E)/Computer%20Science/Kansas%20Computer%20Science%20Model%20Standards%20with%20Description.pdf?ver=2019-04-23-165056-093)

Grade 8

Computing Systems (CS):

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

Standard 8.CS.1 Design a project that combines hardware and software components to collect and exchange data. (*Practice 5: Creating Computational Artifacts; Practice 4: Developing and Using Abstractions*)

Students will use hardware (computer, tablet, mobile device, etc.) and appropriate software (word processing, presentation, spreadsheet, movie maker/editing, etc.) to design a project. For example, students can create a news broadcast related to the Great Depression. For example, students can collect information (interview) of how the depression affected each group of people, the economic impacts, and how the depression impacted Utah's economy.

Network and the Internet (NI):

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

Standard 8.NI.1 Explain how proper protocols transmit data across networks and the internet. (*Practice 4. Developing and Using Abstractions*)

Students will understand rules are developed to deliver data that is broken down into packets (smaller bits of data) to travel across networks and the internet. Students will explain data is delivered in a fast and secure path to avoid missing information. For example, students can create a plan of action to deliver supplies needed in a national disaster. They will need to determine the best route(s) for quick and secure delivery of supplies.

Data and Analysis (DA):

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

Standard 8.DA.3 Test and analyze the effects of changing variables in models/simulations. (*Practice 3. Recognizing and Defining Computational Problems; Practice 4. Developing and Using Abstractions; Practice 5. Creating Computational Artifacts*)

Students will demonstrate how changing variables will affect outcomes in a model/simulation. For example, students will understand the relationship between the mass and speed of objects and the relative amount of kinetic energy of the objects. Students can test and analyze a full cart vs. an empty

cart or rolling spheres with different masses down a ramp to measure the effects on stationary masses.

Algorithms and Programming (AP):

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

Standard 8.AP.1 Develop a program with **iterative protocols** that combine control structures and use **compound conditions**. (*Practice 5. Creating Computational Artifacts; Practice 6. Testing and Refining Computational Artifacts*)

Students will develop programs that use compound conditions (True/False, If/Then, etc.) and loops. The development process should include multiple phases and pseudocode. For example, students will understand the relationship of cause and effect relationships in particle motion, temperature, density, and the state of a pure substance when heat energy is added or removed. Students can create true/false and if/then statements in the development process showing the results of adding and removing heat energy and the cause and effect it has on different substance's states.

8.AP.2 Create **procedures** with or without **parameters** to **organize** code and make it easier to **reuse**. (*Practice 4. Developing and Using Abstractions; Practice 5. Creating Computational Artifacts*)

Students will organize code that can be reused with or without parameters. Students will create procedures that can identify properties of functions. Students will be able to demonstrate the properties of two functions based on x and y values.

8.AP.3 Create a new program **incorporating** existing code, media, and **libraries**; and give proper **attribution**. (*Practice 2. Collaborating Around Computing; Practice 4. Developing and Using Abstractions; Practice 5. Creating Computational Artifacts; Practice 7. Communicating about computing*)

Students will write original programs that incorporate someone else's code and/or media and give proper attribution to the source. Students can manipulate an existing file from a block code program (i.e. Scratch) to demonstrate the conflicts during the American expansion as American Indians were forced from their lands and the tensions over slavery.

Grade 9 / Grade 10

Computing Systems (CS):

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

Standard 9/10.CS.1 Describe ways in which the specific implementation details of a computing system are hidden by **abstractions** in order to manage complexity. (*Practice 4. Developing and Using Abstractions; Practice 7. Communicating About Computing*)

Students will describe how layers of generality simplify the users experience by hiding many of the complex details. For example, the summation symbol Σ indicates that you are adding all terms instead of writing each term individually with plus signs in between. Students could also explain the challenges Alan Turing faced and the process he used in breaking Enigma.

Standard 9/10.CS.2. Identify the different levels of **abstraction** in a computer system. (*Practice 4. Developing and Using Abstractions; Practice 7: Communicating About Computing*)

Students will identify different layers of computing abstraction which could include applications, operating systems, and hardware. For example, an educational app (ex. CANVAS) utilizes the phone's hardware and communicates to the "app" on the phone to send assignments to teachers or comment on a discussion board. Another example of this is describing the functions of the different systems of the body and how they work together to make the body function.

Standard 9/10.CS.3 Develop guidelines that communicate systematic troubleshooting strategies that others can use to **identify** and **fix** errors. (*Practice 6. Testing and Refining Computational Artifacts.*)

Students will develop strategies for troubleshooting and fixing problems and/or errors in a system. 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.⁷ For example, students will design a solution to a space exploration challenge by breaking it down into smaller, more manageable problems that can be solved through the structure and function of a device. Define the problem, identify criteria and constraints, develop possible solutions using models, analyze data to make improvements from iteratively testing solutions, and optimize a solution. Examples of problems could include, cosmic radiation exposure, transportation on other planets or moons, or supplying energy to space travelers.

Network and the Internet (NI):

Computing devices typically do not operate in isolation. Networks connect computing devices to share information and resources and are an increasingly integral part of computing. Networks and communication systems provide greater connectivity in the

⁷ CSTA standards <https://www.csteachers.org/page/standards>

computing world by providing fast, secure communication and facilitating innovation.

Standard 9/10.NI.1 Describe essential elements for connecting to a **network** and identify issues that impact network functionality. (*Practice 7: Communicating About Computing.*)

Students will describe which hardware, software, and information are needed to connect to the internet. Students will also identify issues that might slow down a network connection (overloaded cell phone towers, sporting events, and natural disasters). Teachers may utilize an online network simulator to demonstrate network functionality.

Standard 9/10.NI.2 Describe the design structure of the internet and identify standard **protocols**. (*Practice 4: Developing and Using Abstractions*)

Students will describe how the internet is designed to have multiple paths to any two things that are connected, in case one path is compromised. They will also describe how standard rules allow everything to connect to one network. For example, students can discuss how they can drive home using a different path if the road is closed on one possible path. They can also discuss how the traffic rules help them travel safely.

Data and Analysis (DA):

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

Standard 9/10.DA.1 Demonstrate different representations of data (numbers, characters, and images). (*Practice 4: Developing and Using Abstractions*)

Students will be able to represent data or information in different forms. For example, students will decipher a message in binary code using an alphanumeric key. Students will understand that images or logos could be used to portray information as well.

Standard 9/10.DA.2 Describe disadvantages or benefits associated with how **data elements** are organized and stored. (*Practice 3. Recognizing and Defining Computational Problems; Practice 7: Communicating About Computing*)

Students will describe the properties for a given data set or proper storage choice considering a specific problem [file types, compression (**Lossy** vs. **Lossless**), speed, file size, accessibility]. For example, students will determine the best option for storing photos or music, whether it be on mobile vs computer vs cloud and describe the benefits or costs associated with each method.

Standard 9/10.DA.3 Create data visualizations to help others better understand real-world phenomena or factual data information. (*Practice 5. Creating Computational Artifacts; Practice 7: Communicating About Computing; Practice 4. Developing and Using Abstractions*)

Students will create data visualizations using factual data to better interpret the information. For example, students could develop a chart marking the stock market trends and pertinent historic events (either societal or technological events) to see what types of events affect the stock market in a negative or positive manner.

Algorithms and Programming (AP):

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

Standard 9/10.AP.1 **Design algorithms** to solve computational problems using a combination of original and existing algorithms (*Practice 3. Recognizing and Defining Computational Problems; Practice 4: Developing and Using Abstractions*)

Students will create algorithms that combine existing algorithms with their original program to complete a certain task. For example, students could use the formula for energy of motion to construct a device that converts one form of energy into another form of energy to solve a complex real-life problem.

Standard 9/10.AP.2 **Create** more generalized computational solutions using collections of items (like an **array** or **list**) instead of separating using individual items. (*Practice 4: Developing and Using Abstractions*)

Students will create groups of items using sorting methods by grouping like items together to refer to all at once. For example, students could chart the number of immigrants by nationality that entered the United States during the beginning of the industrial age.

Standard 9/10.AP.3 **Decompose** problems into multiple smaller problems through systematic analysis, using constructs (such as **procedures, modules, functions, methods, and/or classes**). (*Practice 3. Recognizing and Defining Computational Problems*)

Students will break down a big or complex problem and split it into smaller, easier-to-manage components. For example, students will find roots of polynomials by factoring them into smaller components and then solving for each factor.

Standard 9/10.AP.4 **Create computational artifacts** using **modular design**. (*Practice 5: Creating Computational Artifacts*)

Students will create a computational artifact to solve a complex problem by breaking down the problems into smaller, easier-to-manage components. For example, students can solve a complex math problem using the order of operations.

Standard 9/10.AP.5 Identify and collaboratively **suggest** changes to an application's design using feedback from a variety of users. (*Practice 7: Communicating About Computing*)

Students will identify that when they are designing a program or product for a client, they must listen to the clients' needs and wants as well as be willing to accept feedback from peers. For example, students will create or redesign a company logo for a certain company, conduct focus group research on their design, and make proper design corrections based on the feedback.

Standard 9/10.AP.6 Explain the limitations of licenses that restrict **computational artifacts** when using resources created by others. (*Practice 7: Communicating About Computing*)

Students will demonstrate knowledge of different copyright licenses for software use and when to give proper reference. For example, students can research different types of patents and copyright laws that were established during the industrial age and compare them to the intellectual property laws of modern-day patents and licenses.

Standard 9/10.AP.7 Iteratively evaluate and refine a computational artifact to enhance its performance, reliability, usability, and accessibility. (*Practice 6: Testing and Refining Computational Artifacts*)

Students will evaluate how computational artifacts can be developed, tested, and edited repeatedly to improve performance, ease of use, reliability, and/or accessibility. For example, students will use the scientific method to design an air powered rocket to land hit a target from a specific distance. This could also be a great opportunity to introduce **Moore's Law**.

Standard 9/10.AP.8 Design and develop computational artifacts using collaborative tools. (*Practice 2: Collaborating Around Computing; Practice 7: Communicating About Computing*)

Students will use collaborative tools to design and develop computational artifacts as a team. For example, students can collaborate on a presentation using cloud-based applications (Office 365, Google suite, etc.) to complete the design and development process of a project.

Standard 9/10.AP.9 Create documentation (**pseudocode**) that communicates the design of the solution to a complex problem using text, graphics, and/or demonstrations. (*Practice 7: Communicating About Computing*)

Students will design solutions to problems and document these solutions—using pseudocode, flowcharts, and other means--so that they can be implemented by either the student or someone else. During and after implementation, comments and additional documentation can facilitate future maintenance of that process. For example, students will create an outline for an essay before starting on the rough draft.

Impacts of Computing (IC):

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

Standard 9/10.IC.1 Evaluate how computing has impacted and/or impacts personal, ethical, social, economic, and cultural practices. (*Practice 3. Recognizing and Defining Computational Problems; Practice 7: Communicating About Computing*)

Students will determine how computing has positively and/or negatively impacted the world around us. For example, students can research the impact computing has had on society, and as a class, be put into affirmative and negative teams to debate the effects of computing.

Standard 9/10.IC.2 Understand that **bias** is always introduced when creating

computational artifacts, **identify** ways that this unintended **bias** may impact equity, and then **evaluate** methods for alleviating that impact. (*Practice 1: Fostering an Inclusive Computing Culture*)

Students will understand that bias may impact their work and devise solutions for overcoming that bias. When creating computational artifacts, such as software applications, the programmer's experience, culture, values, and knowledge influences the design and outcome. This may inadvertently discriminate against specific groups of users. For example, students can team up to describe how ethnicity affects facial recognition and speech to text functionality in technology, and how to resolve those issues.

Standard 9/10.IC.3 **Identify** solutions to problems in other content areas using established **algorithms**. (*Practice 1: Fostering an Inclusive Computing Culture; Practice 2: Collaborating Around Computing*)

Students will develop solutions to problems that can relate to other subject areas. They will create and analyze a step-by-step process and apply it to a problem relevant to cross-curricular subjects. For example, students can examine the steps involved in solving a quadratic equation.

DRAFT

Grade 11 / Grade 12

Network and the Internet (NI):

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

Standard 11/12.NI.1 Identify types of **security threats**, and then compare and contrast measures that can be used to address, resolve, and/or prevent identified threats.

(Practice 3. Recognizing and Defining Computational Problems; Practice 7: Communicating About Computing)

Students will identify and evaluate different types of security threats and determine potential solutions with justification. For example, students will role play or act out different security threats, in a group, while also showing how to combat that security threat.

Standard 11/12.NI.2 Compare and contrast **cryptographic** techniques to model the secure transmission of **information (data)**. *(Practice 3. Recognizing and Defining Computational Problems; Practice 5. Creating Computational Artifacts; Practice 7: Communicating About Computing)*

Students will demonstrate an understanding of how information is transformed/manipulated via cryptography by creating an encryption algorithm. For example, students will understand how Alan Turing was able to break the Enigma code in World War II. Students will then create their own cypher and share among their peers.

Data and Analysis (DA):

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

Standard 11/12.DA.1 Refine or create computational artifacts to better represent the relationships among different elements of data collected from factual sources or other processes. *(Practice 3. Recognizing and Defining Computational Problems; Practice 4: Developing and Using Abstractions; Practice 5: Creating Computational Artifacts; Practice 6: Testing and Refining Computational Artifacts; Practice 7: Communicating About Computing)*

Students create and refine a **computational model** of data to explain the relationships between the different components of the model. For example, students will write a persuasive essay comparing the Allies and Axis of Power of World War II.

Algorithms and Programming (AP):

An algorithm is a sequence of steps designed to accomplish a specific task. Algorithms are translated into programs, or code, to provide instructions for computing devices. Algorithms and programming control all computing systems, empowering people to communicate with the world in new ways and solve compelling problems. The development process to create meaningful and efficient programs involves choosing

which information to use and how to process and store it, breaking apart large problems into smaller ones, recombining existing solutions, and analyzing different solutions.

Standard 11/12.AP.1 Iteratively design and develop computational artifacts for practical, personal, or societal expression that implements an algorithm based on the result of an evaluation or user input. (*Practice 2: Collaborating Around Computing Practice 3: Recognizing and Defining Computational Problems; Practice 5: Creating Computational Artifacts; Practice 6: Testing and Refining Computational Artifacts*)

Students design and create a computational artifact that develops and implements algorithms (steps) based on the results of an evaluation of a result or user input. For example, students can brainstorm ideas for creating solutions to energy problems with prioritized criteria and trade-offs while considering cost, safety, reliability, as well as possible social, cultural, and environmental impacts.

Standard 11/12.AP.2 Systematically design and create programs for broad audiences by incorporating feedback from users. (*Practice 1: Fostering an Inclusive Computing Culture; Practice 2: Collaborating Around Computing; Practice 3: Recognizing and Defining Computational Problems; Practice 4: Developing and Using Abstractions; Practice 5: Creating Computational Artifacts; Practice 6: Testing and Refining Computational Artifacts; Practice 7: Communicating About Computing*)

Students will review and evaluate feedback from users and then redesign a program (process) to reflect identified needs from user data. For example, students will create a marketing advertisement for a certain product, conduct focus group research on their advertising design, and make proper design corrections based on the feedback.

Standard 11/12.AP.3 Design and develop computational artifacts working in team roles using collaborative tools. (*Practice 2: Collaborating Around Computing; Practice 4: Developing and Using Abstractions; Practice 5: Creating Computational Artifacts; Practice 6: Testing and Refining Computational Artifacts; Practice 7: Communicating About Computing*)

Students will collaborate to design and develop multiple artifacts in teams. For example, students will work together to develop a video game in their subject matter expert roles, which may include, writer, programmer, artist, audio, etc.

Standard 11/12.AP.4 Produce documentation to support the decisions made during the design and creation process using text, graphics, presentations, and/or demonstrations in the development of complex programs. (*Practice 3: Recognizing and Defining Computational Problems; Practice 4: Developing and Using Abstractions; Practice 5: Creating Computational Artifacts; Practice 6: Testing and Refining Computational Artifacts; Practice 7: Communicating About Computing*)

Students will produce documented decisions made during the design and creation process using text, graphics, presentations, and demonstrations in the development of complex programs. For example, students will create instructions to use building blocks (like LEGO's) to instruct others to recreate their design.

Impacts of Computing (IC):

Computing affects many aspects of the world in both positive and negative ways at local, national, and global levels. Individuals and communities influence computing

through their behaviors and cultural and social interactions, and in turn, computing influences new cultural practices. An informed and responsible person should understand the social implications of the digital world, including equity and access to computing.

Standard 11/12.IC.1 Evaluate and discuss the ways computing impacts personal, ethical, social, economic, and cultural practices. (*Practice 1: Fostering an Inclusive Computing Culture; Practice 2: Collaborating Around Computing; Practice 3. Recognizing and Defining Computational Problems; Practice 7: Communicating About Computing*)

Students will evaluate and discuss the ways computing impacts personal, ethical, social, economic, and cultural practices. For example, students will research a current event that is relevant to computer science, take a side (either pro or con), and debate their findings in class.

Standard 11/12.IC.2 Identify impacts of bias and equity deficits on design and implementation of **computational artifacts**, while evaluating appropriate processes for identifying issues of bias. (*Practice 1: Fostering an Inclusive Computing Culture; Practice 2: Collaborating Around Computing; Practice 3. Recognizing and Defining Computational Problems; Practice 5: Creating Computational Artifacts; Practice 6: Testing and Refining Computational Artifacts; Practice 7: Communicating About Computing*)

Students will understand that bias may impact their work and devise solutions for overcoming that bias. When creating computational artifacts, such as software applications, the programmer's experience, culture, values, and knowledge influences the design and outcome. This may inadvertently discriminate against specific groups of users. For example, students can describe how a self-driving car can decide what action to take when every possible action leads to an accident--the programmer must account for these possibilities and the values and culture of the programmer will inform this decision.

Standard 11/12.IC.3 Demonstrate **computational thinking** using algorithms to problem solving across multiple disciplines. (*Practice 3. Recognizing and Defining Computational Problems; Practice 4: Developing and Using Abstractions; Practice 6: Testing and Refining Computational Artifacts; Practice 7: Communicating About Computing*)

Students will demonstrate ways to problem-solve across disciplines. For example, students can use computational thinking and patterns to predict certain genetic traits in chromosomes that will be passed on from parents to offspring.

GLOSSARY

All glossary definitions are attributed to the K-12 Computer Science Framework (2016) and retrieved from k12cs.org.

Term	Definitions
Abstraction	<p>(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.</p> <p>(product): A new representation of a thing, a system, or a problem that helpfully reframes a problem by hiding details irrelevant to the question at hand. [MDESE, 2016]</p>
Algorithm	A step-by-step process to complete a task.
Artifact	Anything created by a human. See computational artifact for the definition used in computer science.
Component	An element of a larger group. Usually, a component provides a particular service or group of related services. [Tech Terms, TechTarget]
Computational Artifacts	Anything created by a human using a computational thinking process and a computing device. A computational artifact can be, but is not limited to, a program, image, audio, video, presentation, or web page file. [College Board, 2016]
Computing	Any goal-oriented activity requiring, benefiting from, or creating algorithmic processes. [MDESE, 2016]
Computing Devices	A physical device that uses hardware and software to receive, process, and output information. Computers, mobile phones, and computer chips inside appliances are all examples of computing devices.
Computing System	A collection of one or more computers or computing devices, together with their hardware and software, integrated for the purpose of accomplishing shared tasks. Although a computing system can be limited to a single computer or computing device, it more commonly refers to a collection of multiple connected computers, computing devices, and hardware.
Conditionals	A feature of a programming language that performs different computations or actions depending on whether a programmer- specified Boolean condition evaluates to true or false. [MDESE, 2016] (A conditional could refer to a conditional statement, conditional expression, or conditional construct.)

Cybersecurity	The protection against access to, or alteration of, computing resources using technology, processes, and training. [TechTarget]
Data	Information that is collected and used for reference or analysis. Data can be digital or nondigital and can be in many forms, including numbers, text, show of hands, images, sounds, or video. [CAS, 2013; Tech Terms]
Debug	The process of finding and correcting errors (bugs) in programs. [MDESE, 2016]
Decompose; Decomposition	Decompose: To break down into components. Decomposition: Breaking down a problem or system into components. [MDESE, 2016]
Deconstruct	Reduce (something) to its constituent parts in order to reinterpret it.
Development	An inherently iterative process through which a desired goal, target, or result is achieved.
Device	A unit of physical hardware that provides one or more computing functions within a computing system. It can provide input to the computer, accept output, or both. [Techopedia]
Event	Any identifiable occurrence that has significance for system hardware or software. User-generated events include keystrokes and mouse clicks; system-generated events include program loading and errors. [TechTarget]
Hardware	The physical components that make up a computing system, computer, or computing device. [MDESE, 2016]
Intellectual Property Rights	Intellectual property rights are the rights given to persons over the creations of their minds. They usually give the creator an exclusive right over the use of his/her creation for a certain period of time
Iterative	Involving the repeating of a process with the aim of approaching a desired goal, target, or result. [MDESE, 2016]
Loop	A programming structure that repeats a sequence of instructions as long as a specific condition is true. [Tech Terms]
Modify	Make partial or minor changes to (something), typically to improve it or to make it less extreme.

Moore's Law	The principle that the speed and capability of computers can be expected to double every two years, as a result of increases in the number of transistors a microchip can contain.
Network	A group of computing devices (personal computers, phones, servers, switches, routers, etc.) connected by cables or wireless media for the exchange of information and resources.
Packet	The unit of data sent over a network. [Tech Terms]
Program; Programming	program (n): A set of instructions that the computer executes to achieve a particular objective. [MDESE, 2016] program (v): To produce a program by programming. programming: The craft of analyzing problems and designing, writing, testing, and maintaining programs to solve them. [MDESE, 2016]
Remixing	To create a new version of (a recording) by recombining and re-editing the elements of the existing recording and often adding material such as new vocals or instrumental tracks.
Security	See the definition for cybersecurity.
Security Threats	A threat is something that may or may not happen but has the potential to cause serious damage. Threats can lead to attacks on computer systems, networks and more.
Sequence	One of the three basic logic structures in computer programming. The other two logic structures are selection and loop. In a sequence structure, an action, or event, leads to the next ordered action in a predetermined order.
Software	Programs that run on a computing system, computer, or other computing device.
Storage/Store	(place) A place, usually a device, into which data can be entered, in which the data can be held, and from which the data can be retrieved at a later time. [FOLDOC] (process) A process through which digital data is saved within a data storage device by means of computing technology. Storage is a mechanism that enables a computer to retain data, either temporarily or permanently. [Techopedia]

System	A collection of elements or components that work together for a common purpose. [TechTarget] See also the definition for computing system.
Test Case	A set of conditions or variables under which a tester will determine whether the system being tested satisfies requirements or works correctly. [STF]
Troubleshooting	A systematic approach to problem-solving that is often used to find and resolve a problem, error, or fault within software or a computing system. [Techopedia, TechTarget]
Unauthorized Access	Unauthorized access is when someone gains access to a website, program, server, service, or other system using someone else's account or other methods. For example, if someone kept guessing a password or username for an account that was not theirs until they gained access it is considered unauthorized access.
Unplugged	Computer Science without a computer.
Variables	A symbolic name that is used to keep track of a value that can change while a program is running. Variables are not just used for numbers; they can also hold text, including whole sentences (strings) or logical values (true or false). A variable has a data type and is associated with a data storage location; its value is normally changed during the course of program execution. [CAS, 2013; Techopedia] Note: This definition differs from that used in math.