Willows Unified School District K–12 Technology Scope and Sequence

Grade	Standard Identifier	Standard	Descriptive Statement	Framework Alignment: Concept	Framework Alignment: Subconcept	Framework Alignment: Practice(s)	Framework Alignment: Sub-practice(s)
K-2	K-2.CS.1	**Select and operate computing devices that perform a variety of tasks accurately and quickly based on user needs and preferences.**	People use computing devices to perform a variety of tasks accurately and quickly. Computing devices interpret and follow the given instructions literally. Students select and operate an appropriate computing device and corresponding program or app for a given task. For example, students could use computing devices to describe what plants and animals (including humans) need to survive. In this case, students could choose to use a keyboard to type explanatory sentences onto graphics. They could also choose to use a touchscreen device with a stylus to annotate an image for a slideshow, or choose to use a camera enabled device to make a video. Student choices may reflect their own needs or the needs of others. (CA NGSS: K-LS1-1; 2-LS4-1) Alternatively, students could choose to use a computing device with audio recording capabilities to recount stories or poems. Students could clarify thoughts, ideas, or feelings via their preference of either using a device with digital drawing tools, or by creating paper and pencil drawing based on their needs and preferences. (CA CCSS for ELA/Literacy SL.K.5, SL.1.5, SL.2.5)	Computing Systems	Devices	Inclusion	1.1
K-2	K-2.CS.2	**Explain the functions of common hardware and software components of computing systems.**	A computing system is composed of hardware and software. Hardware includes the physical components of a computer system. Software provides instructions for the system. These instructions are represented in a form that a computer can understand and are designed for specific purposes. Students identify and describe the function of hardware, such as desktop computers, laptop computers, tablet devices, monitors, keyboards, mice, trackpads, microphones, and printers. Students also identify and describe common software applications such as web browsers, games, and word processors. For example, students could create drawings of a computing system and label its major components with appropriate terminology. Students could then explain the function of each component. (VAPA Visual Arts 2 5.0) (CA CCSS for ELA/Literacy SL.K.5, SL.K.6, SL.1.5, SL.1.6, SL.2.5, SL.2.6) Alternatively, students could each be assigned a component of a computing system and arrange their bodies to represent the system. Students could then describe how their assigned component functions within the system. (P.E.K.1, 1.1)	Computing Systems	Hardware & Software	Communicating	7.2

K-2	K-2.CS.3	**Describe basic hardware and software problems using accurate terminology.**	Problems with computing systems have different causes. Accurate description of the problem aids users in finding solutions. Students communicate a problem with accurate terminology (e.g., when an app or program is not working as expected, a device will not turn on, the sound does not work, etc.). Students at this level do not need to understand the causes of hardware and software problems. For example, students could sort hardware and software terms on a word wall, and refer to the word wall when describing problems using "I see..." statements (e.g., "I see the pointer on the screen is missing", "I see that the computer will not turn on"). (CA CCSS for ELA/Literacy L.K.5.A, L.1.5.A, SL K.5, SL1.5, SL 2.5) (Visual Arts Kinder 5.2) Alternatively, students could use appropriate terminology during collaborative conversations as they learn to debug, troubleshoot, collaborate, and think critically with technology. (CA CCSS for ELA/Literacy SL.K.1, SL.1.1, SL.2.1)	Computing Systems	Troubleshooting	Testing, Communicating	6.2, 7.2
K-2	K-2.NI.4	**Model and describe how people connect to other people, places, information and ideas through a network.**	Information is passed between multiple points (nodes) on a network. The Internet is a network that enables people to connect with other people worldwide through many different points of connection. Students model ways that people communicate, find information, or acquire ideas through a network. Students use a network, such as the internet, to access information from multiple locations or devices. For example, students could utilize a cloud-based platform to access shared documents or note-taking applications for group research projects, and then create a model (e.g., flowchart) to illustrate how this network aids collaboration. (CA CCSS for ELA/Literacy W.K.7, W.1.7, W.2.7) Alternatively, students could design devices that use light or sound to aid communication across distances (e.g., light source to send signals, paper cup and string "telephones," and a pattern of drum beats) and then describe how networks build connections. (CA NGSS: 1-PS4-4)	Networks & the Internet	Network Communication & Organization	Abstraction	4.4
K-2	K-2.NI.5	**Explain why people use passwords.**	Passwords protect information from unwanted use by others. When creating passwords, people often use patterns of familiar numbers and text to more easily remember their passwords. However, this may make the passwords weaker. Knowledge about the importance of passwords is an essential first step in learning about cybersecurity. Students explain that strong passwords are needed to protect devices and information from unwanted use. For example, students could play a game of guessing a three character code. In one version of the game, the characters are only numbers. In the second version, characters are numbers or letters. Students describe why it would take longer to guess the correct code in the second case. Alternatively, students could engage in a collaborative discussion regarding passwords and their importance. Students may follow-up the discussion by exploring strong password components (combination of letters, numbers, and characters), creating their own passwords, and writing opinion pieces indicating reasons their passwords are strong. (CA CCSS for ELA/Literacy SL.K.1, SL.1.1, SL 2.1, W.1.1, W.2.1)	Networks & the Internet	Cybersecurity	Communciating	7.2

K-2	K-2.NI.6	**Create patterns to communicate a message.**	Connecting devices to a network or the Internet provides great benefit, but care must be taken to protect devices and information from unauthorized access. Messages can be protected by using secret languages or codes. Patterns help to ensure that the intended recipient can decode the message. Students create a pattern that can be decoded and translated into a message. For example, students could use a table to associate each text character with a number. Then, they could select a combination of text characters and use mathematical functions (e.g., simple arithmetic operations) to transform the numbers associated with the characters into a secret message. Using inverse functions, a peer could translate the secret message back into its original form. (CA CCSS for Mathematics 2.OA.A.1, 2.OA.B.2) Alternatively, students could use icons or invented symbols to represent patterns of beat, rhythm, or pitch to decode a musical phrase. (VAPA Music K.1.1, 1.1.1, 2.1.1, 2.2.2)	Networks & the Internet	Cybersecurity	Abstraction	4.4
K-2	K-2.DA.7	**Store, copy, search, retrieve, modify, and delete information using a computing device, and define the information stored as data.**	Information from the real world can be stored and processed by a computing device. When stored on a computing device, it is referred to as data. Data can include images, text documents, audio files, and video files. Students store, copy, search, retrieve, modify, and delete information using a computing device and define the information stored as data. For example, students could produce a story using a computing device, storing it locally or remotely (e.g., in the cloud). They could then make a copy of the story for peer revision and editing. When the final copy of the story is complete, students delete any unnecessary files. They search for and retrieve data from a local or remote source, depending on where it was stored. (CA CCSS for ELA/Literacy W.K.6, W.K.5, W1.6, W.1.5, W.2.6, W.2.5) Alternatively, students could record their voices singing an age-appropriate song. They could store the data on a computing device, search for peers' audio files, retrieve their own files, and delete unnecesary takes. (VAPA Music K.2.2, 1.2.2, 2.2.2)	Data & Analysis	Storage	Abstraction	4.2
K-2	K-2.DA.8	**Collect and present data in various visual formats.**	Data can be collected and presented in various visual formats. For example, students could measure temperature changes throughout a day. They could then discuss ways to display the data visually. Students could extend the activity by writing different narratives based on collected data, such as a story that begins in the morning when temperatures are low and one that begins in the afternoon when the sun is high and temperatures are higher. (CA CCSS for ELA/Literacy RL.K.9, RL.1.9, RL.2.9, W.K.3, W.1.3, W.2.3). Alternatively, students collect peers' favorite flavor of ice cream and brainstorm differing ways to display the data. In groups, students can choose to display and present the data in a format of their choice. (CA CCSS for Mathematics K.MD.3, 1.MD.4, 2.MD.10)	Data & Analysis	Collection Visualization & Transformation	Communicating, Abstraction	7.1, 4.4

K-2	K-2.DA.9	**Identify and describe patterns in data visualizations, such as charts or graphs, to make predictions.**	Data can be used to make inferences or predictions about the world. For example, students could record the number of each color of candy in a small packet. Then, they compare their individual data with classmates. Students could use the collected data to predict how many of each colored candy will be in a full size bag of like candy. (CA CCSS for Mathematics K.MD.3, 1.MD.4, 2.MD.10) Alternatively, students could sort and classify objects according to their properties and note observations. Students could then create a graph or chart of their observations and look for connections/relationships (e.g., items that are hard are usually also smooth, or items that are fluffy are usually also light in weight.) Students then look at pictures of additional objects and make predictions regarding the properties of the objects pictured. (CA NGSS: 2-PS1-1, 2-PS1-2)	Data & Analysis	Inference & Models	Abstraction	4.1
K-2	K-2.AP.10	**Model daily processes by creating and following algorithms to complete tasks.**	Algorithms are sequences of instructions that describe how to complete a specific task. Students create algorithms that reflect simple life tasks inside and outside of the classroom. For example, students could create algorithms to represent daily routines for getting ready for school, transitioning through center rotations, eating lunch, and putting away art materials. Students could then write a narrative sequence of events. (CA CCSS for ELA/Literacy W.K.3, W.1.3, W.2.3) Alternatively, students could create a game or a dance with a specific set of movements to reach an intentional goal or objective. (P.E K.2, 1.2, 2.2) Additionally, students could create a map of their neighborhood and give step-by-step directions of how they get to school. (HSS.K.4, 1.2, 2.2)	Algorithms & Programming	Algorithms	Computational Problems, Abstraction	4.4, 3.2
K-2	K-2.AP.11	**Model the way programs store data.**	Information in the real world can be represented in computer programs. Students model the digital storage of data by transforming real-world information into symbolic representations that include text, numbers, and images. For example, after identifying symbols on a map and explaining what they represent in the real world, students could create their own symbols and corresponding legend to represent items on a map of their classroom (HSS.K.4.3, 1.2.3, 2.2.2) Alternatively, students could invent symbols to represent beat and/or pitch. Students could then modify symbols within the notation and explain how the musical phrase changes. (VAPA Music K.1.1, 1.1.1, 2.1.1, 2.2.2)	Algorithms & Programming	Variables	Abstraction	4.4

K-2	K-2.AP.12	**Create programs with sequences of commands and simple loops, to express ideas or address a problem.**	People create programs by composing sequences of commands that specify the precise order in which instructions should be executed. Loops enable programs to repeat a sequence of commands multiple times. For example, students could follow simple movements in response to oral instructions. Students could then create a simple sequence of movement commands in response to a given problem (e.g., In how many ways can you travel from point A to point B?) and represent it as a computer program, using loops to repeat commands. (VAPA Dance K.1.4, 1.2.3, 1.2.5, 1.2.8, 2.2.1, 2.2.2, 2.2.3) Alternatively, on a mat with many different CVC words, students could program robots to move to words with a similar vowel sound. Students could look for multiple ways to solve the problem and simplify their solution by incorporating loops. (CA CCSS for ELA/Literacy RF.K.2.D, RF.1.2.C)	Algorithms & Programming	Control, Modularity	Creating	5.2
K-2	K-2.AP.13	**Decompose the steps needed to solve a problem into a sequence of instructions.**	Decomposition is the act of breaking down tasks into simpler tasks. For example, students could break down the steps needed to make a peanut butter and jelly sandwich, to brush their teeth, to draw a shape, to move a character across the screen, or to solve a level of a coding app. In a visual programming environment, students could break down the steps needed to draw a shape. (CA CCSS for Mathematics K.G.5, 1.G.1, 2.G.1) Alternatively, students could decompose the planning of a birthday party into tasks such as: 1) Decide when and where it should be, 2) List friends and family to invite, 3) Send the invitations, 4) Bake a cake, 5) Decorate, etc.	Algorithms & Programming	Modularity	Computational Problems	3.2
K-2	K-2.AP.14	**Develop plans that describe a program's sequence of events, goals, and expected outcomes.**	Creating a plan for what a program will do clarifies the steps that will be needed to create the program and can be used to check if a program runs as expected. Students create a planning document to illustrate their program's sequence of events, goals, and expected outcomes of what their program will do. Planning documents could include a story map, a storyboard, or a sequential graphic organizer, to illustrate their program's sequence of events, goals, and expected outcomes of what their program will do. Students at this level may complete the planning process with help from the teacher. For example, students could create a storyboard or timeline that represents a family's history, leading to their current location of residence. Students could then create a plan for a program that animates the story of family locations. (HSS 2.1.1) (CA CCSS for ELA/Literacy W.K.3, W.1.3, W.2.3)	Algorithms & Programming	Program Development	Creating, Communicating	5.1, 7.2

K-2	K-2.AP.15	**Give attribution when using the ideas and creations of others while developing programs.**	Computing makes it easy to reuse and remix others' creations, and this comes with a level of responsibility. Students credit artifacts that were created by others, such as pictures, music, and code. Credit could be given orally if presenting their work to the class, or in writing if sharing work on a class blog or website. Proper attribution at this stage does not require formal citation, such as in a bibliography or works cited document. For example, when creating an animation of the sun, moon, and stars using a blocks-based language, students could draw their own sun and use an image of the moon and stars from a website or a teammate. When students present the model to the class, they can orally give credit to the website or peer for the contributions. (CA CCSS for ELA/Literacy SL.K.5, SL.1.5, SL.2.5) (NGSS.1-ESS1-1) (CA Model School Library Standards 2.3.b, 2.4.2.a)	Algorithms & Programming	Program Development	Communicating	7.3
K-2	K-2.AP.16	**Debug errors in an algorithm or program that includes sequences and simple loops.**	Algorithms or programs may not always work correctly. Students use various strategies, such as changing the sequence of the steps, following the algorithm in a step-by-step manner, or trial and error to fix problems in algorithms and programs. For example, when given images placed in a random order, students could give step-by-step commands to direct a robot, or a student playing a robot, to navigate to the images in the correct sequence. Examples of images include storyboard cards from a familiar story (CA CCSS for ELA/Literacy RL.K.2, RL.1.2, RL.2.2) and locations of the sun at different times of the day (CA NGSS: 1-ESS1-1). Alternatively, students could "program" the teacher or another classmate by giving precise instructions to make a peanut butter and jelly sandwich or navigate around the classroom. When the teacher or classmate doesn't respond as intended, students correct their commands. Additionally, students could receive a partially completed soundboard program that has a variety of animals programmed to play a corresponding sound when the user touches them. Students correct any sounds that don't match the animal (e.g., if the cat moos, students change the moo sound to meow).	Algorithms & Programming	Program Development	Testing	6.2
K-2	K-2.AP.17	**Describe the steps taken and choices made during the iterative process of program development.**	Program developers make choices and iterate to continually refine their product. At this stage, students explain or write about the goals and expected outcomes of the programs they create and the choices that they made when creating programs. Students could use coding journals, discussions with a teacher, class presentations, or blogs. For example, students could use a combination of images, verbal reflections, a physical model, and/or written text to show the step-by-step process taken to develop a program such as cutting and pasting coding commands into a journal, using manipulatives that represent different commands and control structures, and taking screenshots of code and adding to a digital journal. This iterative process could be documented via a speech, journal, one on one conference with teacher or peer, small group conference, or blog. (CA CCSS for ELA/Literacy SL.K.5, SL.1.5, SL.2.5) (CA NGSS: K-2-ETS1-2)	Algorithms & Programming	Program Development	Communicating	7.2

K-2	K-2.IC.18	**Compare how people lived and worked before and after the adoption of new computing technologies.**	Computing technologies have changed the way people live and work. Students describe the positive and negative impacts of these changes. For example, as a class, students could create a timeline that includes advancements in computing technologies. Each student could then choose an advancement from the timeline and make a graphic organizer noting how people's lives were different before and after its introduction into society. Student responses could include: In the past, if students wanted to read about a topic, they needed access to a library to find a book about it. Today, students can view and read information on the Internet about a topic or they can download e-books about it directly to a device. Such information may be available in more than one language and could be read to a student, allowing for great accessibility. (HSS.K.6.3) Alternatively, students could retell or dramatize stories, myths, and fairy tales from two distinct time periods before and after a particular computing technology had been introduced. For example, the setting of one story could take place before smartphones had been invented, while a second setting could take place with smartphones in use by characters in the story. Students could note the positive and negative aspects of smartphones on the daily lives of the characters in the story. (VAPA Theatre Arts K.3.1, K.3.2, 1.2.2, 2.3.2) (CA CCSS for ELA/Literacy RL.K.2, RL.K.9, RL.1., RL.1.9, RL.2.2, RL.2.9)	Impacts of Computing	Culture	Computational Problems	3.1
K-2	K-2.IC.19	**Work respectfully and responsibly with others when communicating electronically.**	Electronic communication facilitates positive interactions, such as sharing ideas with many people, but the public and anonymous nature of electronic communication also allows intimidating and inappropriate behavior in the form of cyberbullying. Responsible electronic communication includes limiting access to personably identifiable information. Students learn and use appropriate behavior when communicating electronically (often called "netiquette"). For example, students could share their work on a classroom blog or in other collaborative spaces online, taking care to avoid sharing information that is inappropriate or that could personally identify themselves to others. (CA CCSS for ELA/Literacy W.K.6, W.1.6, W.21.6) Alternatively, students could provide feedback to others on their work in a kind and respectful manner. They could learn how written words can be easily misinterpreted and may seem negative when the intention may be to express confusion, give ideas, or prompt further discussion. They could also learn to identify harmful behavior on collaborative spaces and intervening to find the proper authority to help. (CA CCSS for ELA/Literacy W.K.5, W.1.5, W.2.5) (HSS 1.1.2)	Impacts of Computing	Social Interactions	Collaborating	2.1

K-2	K-2.IC.20	**Describe approaches and rationales for keeping login information private, and for logging off of devices appropriately.**	People use computing technology in ways that can help or hurt themselves and/or others. Harmful behaviors, such as sharing passwords or other private information and leaving public devices logged in should be recognized and avoided. Students keep login information private, log off of devices appropriately, and discuss the importance of these practices. For example, while learning about individual responsibility and citizenship, students could create a "privacy folder" to store login information, and keep this folder in a secure location that is not easily seen and accessed by classmates. Students could discuss the relative benefits and impacts of choosing to store passwords in a folder online versus on paper. They could also describe how using the same login and password across many systems and apps could lead to significant security issues and requires even more vigilance in maintaining security. (HSS K.1) Alternatively, students can write an informational piece regarding the importance of keeping login information private and logging off of public devices. (CA CCSS for ELA/Literacy W.K.2, W.1.2, W.2.2)	Impacts of Computing	Safety Law & Ethics	Computational Problems	3.1
3-5	3-5.CS.1	**Describe how computing devices connect to other components to form a system.**	Computing devices often depend on other devices or components. Students describe physical and wireless connections to other components, including both input devices (e.g., keyboards, sensors, remote controls, microphones) and output devices (e.g., 3D printers, monitors, speakers). For example, students could describe the relationship among the heart, lungs, muscles, blood, and oxygen during physical activity and then compare this to how a mouse, keyboard, printer, and desktop computer connect and interact to allow for input, processing, and output. (P.E.3.4.7) Alternatively, when describing how light reflected from objects enters the eye and is then transferred to the brain to construct a visual image, students could compare this to a computing system that uses programming to construct a visual image when data is transferred and constructed/reconstructed through a keyboard, camera, or other components. (CA NGSS: 4-PS4-2)	Computing Systems	Devices	Communicating	7.2

3-5	3-5.CS.2	**Demonstrate how computer hardware and software work together as a system to accomplish tasks.**	Hardware and software are both needed to accomplish tasks with a computing device. Students create a model to illustrate ways in which hardware and software work as a system. Students could draw a model on paper or in a drawing program, program an animation to demonstrate it, or demonstrate it by acting this out in some way. At this level, a model should only include the basic elements of a computer system, such as input, output, processor, sensors, and storage. For example, students could create a diagram or flow chart to indicate how a keyboard, desktop computer, monitor, and word processing software interact with each other. The keyboard (hardware) detects a key press, which the operating system and word processing application (software) displays as a new character that has been inserted into the document and is visible through the monitor (hardware). Students could also create a model by acting out the interactions of these different hardware and software components. Alternatively, when describing that animals and people receive different types of information through their senses, process the information in their brain, and respond to the information in different ways, students could compare this to the interaction of how the information traveling through a computer from mouse to processor are similar to signals sent through the nervous system telling our brain about the world around us to prompt responses. (CA NGSS: 4-LS1-2)	Computing Systems	Hardware & Software	Abstraction	4.4
3-5	3-5.CS.3	**Determine potential solutions to solve simple hardware and software problems using common troubleshooting strategies.**	Although computing systems vary, common troubleshooting strategies can be used across many different systems. Students use troubleshooting strategies to identify problems that could include a device not responding, lacking power, lacking a network connection, an app crashing, not playing sounds, or password entry not working. Students use and develop various solutions to address these problems. Solutions may include rebooting the device, checking for power, checking network availability, opening and closing an app, making sure speakers are turned on or headphones are plugged in, and making sure that the caps lock key is not on. For example, students could prepare for and participate in a collaborative discussion in which they identify and list computing system problems and then describe common successful fixes. (CA CCSS for ELA/Literacy SL.3.1, SL.4.1, SL.5.1) Alternatively, students could write informative/explanatory texts, create a poster, or use another medium of communication to examine common troubleshooting strategies and convey these ideas and information clearly. (CA CCSS for ELA/Literacy W.3.2, W.4.2, W.5.2)	Computing Systems	Troubleshooting	Testing	6.2

3-5	3-5.NI.4	**Model how information is broken down into smaller pieces, transmitted as packets through multiple devices over networks and the Internet, and reassembled at the destination.**	Information is sent and received over physical or wireless paths. It is broken down into smaller pieces called packets, which are sent independently and reassembled at the destination. Students demonstrate their understanding of this flow of information by, for instance, drawing a model of the way packets are transmitted, programming an animation to show how packets are transmitted, or demonstrating this through an unplugged activity in which they physically act this out. For example, students could design a structure using building blocks or other materials with the intention of re-engineering it in another location, just as early Americans did after the intercontinental railroad was constructed in the 1850s (HSS.4.4.1, 4.4.2). Students could deconstruct the designed structure, place materials into specific containers (or plastic bags/brown paper bags/etc.), and develop instructions on how to recreate the structure once each container arrives at its intended destination. (CA NGSS: 3-5-ETS1) For example, students could cut up a map of the United States by state lines. Students could then place the states in envelopes and transmit the "packets" through a physical network, represented by multiple students spreading out in arms reach of at least two others. At the destination, the student who receives the packets resassembles the individual states back into a map of the United States. (HSS 5.9) Alternatively, students could perform a similar activity with a diatonic scale, cutting the scale into individual notes. Each note, in order, should be placed into a numbered envelope based on its location on the scale. These envelopes can be transmitted across the network of students and reassembled at the destination. (VAPA Music 4.1.2)	Networks & the Internet	Network Communication & Organization	Abstraction	4.4
3-5	3-5.NI.5	**Describe physical and digital security measures for protecting personal information.**	Personal information can be protected physically and digitally. Cybersecurity is the protection from unauthorized use of electronic data, or the measures taken to achieve this. Students identify what personal information is and the reasons for protecting it. Students describe physical and digital approaches for protecting personal information such as using strong passwords and biometric scanners. For example, students could engage in a collaborative discussion orally or in writing regarding topics that relate to personal cybersecurity issues. Discussion topics could be based on current events related to cybersecurity or topics that are applicable to students, such as the necessity of backing up data to guard against loss, how to create strong passwords and the importance of not sharing passwords, or why we should keep operating systems updated and use anti-virus software to protect data and systems. Students could also discuss physical measures that can be used to protect data including biometric scanners, locked doors, and physical backups. (CA CCSS for ELA/Literacy SL.3.1, SL.4.1, SL.5.1)	Networks & the Internet	Cybersecurity	Computational Problems	3.1

3-5	3-5.NI.6	**Create patterns to protect information from unauthorized access.**	Encryption is the process of converting information or data into a code, especially to prevent unauthorized access. At this level, students use patterns as a code for encryption, to protect information. Patterns should be decodable to the party for whom the message is intended, but difficult or impossible for those with unauthorized access. For example, students could create encrypted messages via flashing a flashlight in Morse code. Other students could decode this established language even if it wasn't meant for them. To model the idea of protecting data, students should create their own variations on or changes to Morse code. This ensures that when a member of that group flashes a message only other members of their group can decode it, even if other students in the room can see it. (CA NGSS: 4-PS4-3) Alternatively, students could engage in a CS Unplugged activity that models public key encryption: One student puts a paper containing a written secret in a box, locks it with a padlock, and hands the box to a second student. Student 2 puts on a second padlock and hands it back. Student 1 removes her lock and hands the box to student 2 again. Student 2 removes his lock, opens the box, and has access to the secret that student 1 sent him. Because the box always contained at least one lock while in transit, an outside party never had the opportunity to see the message and it is protected.	Networks & the Internet	Cybersecurity	Abstraction	4.4
3-5	3-5.DA.7	**Explain that the amount of space required to store data differs based on the type of data and/or level of detail.**	All saved data requires space to store it, whether locally or not (e.g., on the cloud). Music, images, video, and text require different amounts of storage. Video will often require more storage and different format than music or images alone because video combines both. The level of detail represented by that data also affects storage requirements. For instance, two pictures of the same object can require different amounts of storage based upon their resolution, and a high-resolution photo could require more storage than a low-resolution video. Students select appropriate storage for their data. For example, students could create an image using a standard drawing app. They could save the image in different formats (e.g., .png, .jpg, .pdf) and compare file sizes. They should also notice that different file sizes can result in differences in quality or resolution (e.g., some pictures could be more pixelated while some could be sharper). Alternatively, in an unplugged activity, students could represent images by coloring in squares within a large grid. They could model how a larger grid requires more storage but also represents a clearer image (i.e., higher resolution).	Data & Analysis	Storage	Abstraction	4.2

3-5	3-5.DA.8	**Organize and present collected data visually to highlight relationships and support a claim.**	Raw data has little meaning on its own. Data is often sorted or grouped to provide additional clarity. Organizing data can make interpreting and communicating it to others easier. Data points can be clustered by a number of commonalities. The same data could be manipulated in different ways to emphasize particular aspects or parts of the data set. For example, students could create and administer electronic surveys to their classmates. Possible topics could include favorite books, family heritage, and after school activities. Students could then create digital displays of the data they have collected such as column histogram charts showing the percent of respondents in each grade who selected a particular favorite book. Finally, students could make quantitative statements supported by the data such as which books are more appealing to specific ages of students. As an extension, students could write an opinion piece stating a claim and supporting it with evidence from the data they collected. (CA CCSS for Mathematics 3.MD.3, 4.MD.4, 5.MD.2) (CA CCSS for ELA/Literacy W.3.1, W.4.1, W.5.1) Alternatively, students could represent data in tables and graphical displays to describe weather experienced in the last several years. They could select the type of graphical display based on the specific data represented (e.g., daily high/low temperatures on a scatter plot, average temperatures for a month across years in a column chart). Students could then make a claim about expected weather in future months based on the data. (CA NGSS: 3-ESS2-1)	Data & Analysis	Collection Visualization & Transformation	Communicating	7.1
3-5	3-5.DA.9	**Use data to highlight and/or propose relationships, predict outcomes, or communicate ideas.**	The accuracy of data analysis is related to how the data is represented. Inferences or predictions based on data are less likely to be accurate if the data is insufficient, incomplete, or inaccurate or if the data is incorrect in some way. Additionally, people select aspects and subsets of data to be transformed, organized, and categorized. Students should be able to refer to data when communicating an idea, in order to highlight and/or propose relationships, predict outcomes, highlight different views and/or communicate insights and ideas. For example, students can be provided a scenario in which they are city managers who have a specific amount of funds to improve a city in California. Students can collect data of a city concerning land use, vegetation, wildlife, climate, population density, services and transportation (HSS.4.1.5) to determine and present what area needs to be focused on to improve a problem. Students can compare their data and planned use of funds with peers, clearly communicating or predict outcomes based on data collected. (CA CCCS for ELA/Literacy SL.3.1, SL.4.1, SL.5.1) Alternatively, students could record the temperature at noon each day to show that temperatures are higher in certain months of the year. If temperatures are not recorded on non-school days or are recorded incorrectly, the data would be incomplete and ideas being communicated could be inaccurate. Students may also record the day of the week on which the data was collected, but this would have no relevance to whether temperatures are higher or lower. In order to have sufficient and accurate data on which to communicate the idea, students might use data provided by a governmental weather agency. (CA NGSS: 3-ESS2-1)	Data & Analysis	Inference & Models	Communicating	7.1

3-5	3-5.AP.10	**Compare and refine multiple algorithms for the same task and determine which is the most appropriate.**	Different algorithms can achieve the same result, though sometimes one algorithm might be more appropriate for a specific solution. Students examine different ways to solve the same task and decide which would be the better solution for the specific scenario. For example, students could use a map and create multiple algorithms to model the early land and sea routes to and from European settlements in California. They could then compare and refine their algorithms to reflect faster travel times, shorter distances, or avoid specific characteristics, such as mountains, deserts, ocean currents, and wind patterns. (HSS.4.2.2) Alternatively, students could identify multiple algorithms for decomposing a fraction into a sum of fractions with the same denominator and record each decomposition with an equation (e.g., 2 1/8 = 1 + 1 + 1/8 = 8/8 + 8/8 + 1/8). Students could then select the most efficient algorithm (e.g., fewest number of steps). (CA CCSS for Mathematics 4.NF.3b) Additionally, students could compare algorithms that describe how to get ready for school and modify them for supporting different goals including having time to care for a pet, being able to talk with a friend before classes start, or taking a longer route to school to accompany a younger sibling to their school first. Students could then write an opinion piece, justifying with reasons their selected algorithm is most appropriate. (CA CCSS for ELA/Literacy W.3.1, W.4.1, W.5.1)	Algorithms & Programming	Algorithms	Testing, Computational Problems	6.3, 3.3
3-5	3-5.AP.11	**Create programs that use variables to store and modify data.**	Variables are used to store and modify data. Students use variables in programs they create. At this level, students may need guidance in identifying when to create variables (i.e., performing the abstraction). For example, students could create a game to represent predators and prey in an ecosystem. They could declare a "score" variable, assign it to 0 at the start of the game, and add 1 (increment) the score each time the predator captures its prey. They could also declare a second "numberOfLives" variable, assign it to 3 at the start of the game, and subtract 1 (decrement) each time a prey is captured. They could program the game to end when "numberOfLives" equals 0. (CA NGSS: 5-LS2-1) (CA CCSS for Mathematics 5.OA.3) Alternatively, when students create programs to draw regular polygons, they could use variables to store the line size, line color, and/or side length. Students can extend learning by creatively combining a variety of polygons to create digital artwork, comparing and contrasting this to another work of art made by the use of different art tools and media, such as watercolor or tempera paints. (CA CCSS for Mathematics 3.G.1) (VAPA Visual Arts 3.1.4)	Algorithms & Programming	Variables	Creating	5.2

3-5	3-5.AP.12	**Create programs that include events, loops, and conditionals.**	Control structures specify the order (sequence) in which instructions are executed within a program and can be combined to support the creation of more complex programs. Events allow portions of a program to run based on a specific action. Conditionals allow for the execution of a portion of code in a program when a certain condition is true. Loops allow for the repetition of a sequence of code multiple times. For example, students could program an interactive map of the United States of America. They could use events to initiate a question when the user clicks on a state and conditionals to check whether the user input is correct. They could use loops to repeat the question until the user answers correctly or to control the length of a "congratulations" scenario that plays after a correct answer. (HSS.5.9) Alternatively, students could write a math fluency game that asks products of two one-digit numbers and then uses a conditional to check whether or not the answer that was entered is correct. They could use a loop to repeatedly ask another question. They could use events to allow the user to click on a green button to play again or a red button to end the game. (CA CCSS for Mathematics 3.OA.7) Additionally, students could create a program as a role-playing game based on a literary work. Loops could be used to animate a character's movement. When reaching a decision point in the story, an event could initiate the user to type a response. A conditional could change the setting or have the story play out differently based on the user input. (CA CCSS for ELA/Literacy RL.5.3)	Algorithms & Programming	Control	Creating	5.2
3-5	3-5.AP.13	**Decompose problems into smaller, manageable tasks which may themselves be decomposed.**	Decomposition is the act of breaking down tasks into simpler tasks. This manages complexity in the problem solving and program development process. For example, students could create an animation to represent a story they have written. Students write a story and then break it down into different scenes. For each scene, they would select a background, place characters, and program actions in that scene. (CA CCSS for ELA/Literacy W.3.3, W.4.3, W.5.3) Alternatively, students could create a program to allow classmates to present data collected in an experiment. For example, if students collected rain gauge data once per week for 3 months, students could break down the program tasks: 1) ask the user to input 12 weeks worth of data, 2) process the data (e.g., add the first four entries to calculate the rain amount for month 1, convert to metric system measurements), and 3) direct the creation or resizing of objects (e.g., one rectangular chart bar for each month) to represent the total number of rainfall for that month. (CA NGSS: 3-ETS-1-2) (CA CCSS for Mathematics 3.MD.2)	Algorithms & Programming	Modularity	Computational Problems	3.2

3-5	3-5.AP.14	**Create programs by incorporating smaller portions of existing programs, to develop something new or add more advanced features.**	Programs can be broken down into smaller parts, which can be incorporated into new or existing programs. Students incorporate predefined functions into their original designs. At this level, students do not need to understand all of the underlying implementation details of the abstractions that they use. For example, students could use code from a ping pong animation to make a ball bounce in a new basketball game. They could also incorporate code from a single-player basketball game to create a two-player game with slightly different rules. Alternatively, students could remix an animated story and add their own conclusion and/or additional dialogue. (CA CCSS for ELA/Literacy W.3.3.B, W.3.3.D, W.4.3.B, W.4.3.E, W.5.3.B, W.5.3.E) Additionally, when creating a game that occurs on the moon or planets, students could incorporate and modify code that simulates gravity on Earth. They could modify the strength of the gravitational force based on the mass of the planet or moon. (CA NGSS: 5-PS2-1)	Algorithms & Programming	Modularity, Program Development	Abstraction, Creating	4.2, 5.3
3-5	3-5.AP.15	**Use an iterative process to plan and develop a program by considering the perspectives and preferences of others.**	Planning is an important part of the iterative process of program development. Students gain a basic understanding of the importance and process of planning before beginning to write code for a program. They plan the development of a program by outlining key features, time and resource constraints, and user expectations. Students should document the plan as, for example, a storyboard, flowchart, pseudocode, or story map. For example, students could collaborate with a partner to plan and develop a program that graphs a function. They could iteratively modify the program based on feedback from diverse users, such as students who are color blind and may have trouble differentiating lines on a graph based on the color. (CA CCSS for Mathematics 5.G.1, 5.G.2) Alternatively, students could plan as a team to develop a program to display experimental data. They could implement the program in stages, generating basic displays first and then soliciting feedback from others on how easy it is to interpret (e.g., are labels clear and readable?, are lines thick enough?, are titles understandable?). Students could iteratively improve their display to make it more readable and to better support the communication of the finding of the experiment. (NGSS 3-5-ETS1-1, 3-5-ETS1-2, 3-5-ETS1-3)	Algorithms & Programming	Program Development	Inclusion, Creating	1.1, 5.1

3-5	3-5.AP.16	**Observe intellectual property rights and give appropriate attribution when creating, remixing, or combining programs.**	Intellectual property rights can vary by country, but copyright laws give the creator of a work a set of rights and prevents others from copying the work and using it in ways that they may not like. Students consider common licenses that place limitations or restrictions on the use of others' work, such as images and music downloaded from the Internet. When incorporating the work of others, students attribute the work. At this level, students could give attribution by including credits or links directly in their programs, code comments, or separate project pages. For example, when making a program to model the life cycle of a butterfly, students could modify and reuse an existing program that describes the life cycle of a frog. Based on their research, students could identify and use Creative Commons-licensed or public domain images and sounds of caterpillars and butterflies. Students give attribution by properly citing the source of the original piece as necessary. (CA NGSS: 3-LS-1-1) (CA CCSS for ELA/Literacy W.3.8, W.4.8, W.5.8) Alternatively, when creating a program explaining the structure of the United States goverment, students find Creative Commons-licensed or public domain images to represent the three branches of government and attribute ownership of the images appropriately. If students find and incorporate an audio file of a group playing part of the national anthem, they appropriately give attribution on the project page. (HSS.3.4.4)	Algorithms & Programming	Program Development	Creating, Communicating	5.2, 7.3
3-5	3-5.AP.17	**Test and debug a program or algorithm to ensure it accomplishes the intended task.**	Programs do not always run properly. Students need to understand how to test and make necessary corrections to their programs to ensure they run properly. Students successfully identify and fix errors in (debug) their programs and programs created by others. Debugging strategies at this level may include testing to determine the first place the solution is in error and fixing accordingly, leaving "breadcrumbs" in a program, and soliciting assistance from peers and online resources. For example, when students are developing a program to control the movement of a robot in a confined space, students test various inputs that control movement of the robot to make sure it behaves as intended (e.g., if an input would cause the robot to move past a wall of the confined space, it should not move at all). (CA NGSS: 3-5-ETS1-3) Additionally, students could test and debug an algorithm by tracing the inputs and outputs on a whiteboard. When noticing "bugs" (errors), students could identify what was supposed to happen and step through the algorithm to locate and then correct the error.	Algorithms & Programming	Program Development	Testing	6.2

3-5	3-5.AP.18	**Perform different roles when collaborating with peers during the design, implementation, and review stages of program development.**	Collaborative computing is the process of creating computational artifacts by working in pairs or on teams. It involves asking for the contributions and feedback of others. Effective collaboration can often lead to better outcomes than working independently. With teacher guidance, students take turns in different roles during program development, such as driver, navigator, notetaker, facilitator, and debugger, as they design and implement their program. For example, while taking on different roles during program development, students could create and maintain a journal about their experiences working collaboratively. (CA CCSS for ELA/Literacy W.3.10, W.4.10, W.5.10) (CA NGSS: 3-5-ETS1-2)	Algorithms & Programming	Program Development	Collaborating	2.2
3-5	3-5.AP.19	**Describe choices made during program development using code comments, presentations, and demonstrations.**	People communicate about their code to help others understand and use their programs. Explaining one's design choices gives others a better understanding of one's work. Students may explain their step-by-step process of creating a program in a presentation or demonstration of their personal code journals. They describe how comments within code organize thought and process during the develpment of the program. For example, students could describe the decision to have the score in a game flash when it can be rounded to 100 by writing a comment in the code. (CA CCSS for Mathematics 3.NBT.1) Alternatively, students could present their overall program development experience and justify choices made by using storyboards, annotated images, videos, and/or journal entries. (CA CCSS for ELA/Literacy SL.3.4, SL.4.4, SL.5.4, SL.3.5, SL.4.5, SL.5.5) (CA NGSS: 3-5-ETS1-1, 3.5-ETS1-2, 3.5-ETS1-3)	Algorithms & Programming	Program Development	Communicating	7.2
3-5	3-5.IC.20	**Discuss computing technologies that have changed the world, and express how those technologies influence, and are influenced by, cultural practices.**	New computing technologies are created and existing technologies are modified for many reasons, including to increase their benefits, decrease their risks, and meet societal needs. Students, with guidance from their teacher, discuss topics that relate to the history of computing technologies and changes in the world due to these technologies. Topics could be based on current news content, such as robotics, wireless Internet, mobile computing devices, GPS systems, wearable computing, and how social media has influenced social and political changes. For example, students could conduct research in computing technologies that impact daily life such as self-driving cars. They engage in a collaborative discussion describing impacts of these advancements (e.g., self-driving cars could reduce crashes and decrease traffic, but there is a cost barrier to purchasing them). (CA CCSS for ELA/Literacy W.3.7, W.4.7, W.5.7, SL.3.1, SL.4.1, SL.5.1) Alternatively, students could discuss how technological advancements affected the entertainment industry and then compare and contrast the impacts on audiences. For instance, people with access to high-speed Internet may be able to choose to utilize streaming media (which may cost less than traditional media options), but those in rural areas may not have the same access and be able to reap those benefits. (VAPA Theatre Arts 4.3.2, 4.4.2)	Impacts of Computing	Culture	Computational Problems	3.1

3-5	3-5.IC.21	**Propose ways to improve the accessibility and usability of technology products for the diverse needs and wants of users.**	The development and modification of computing technology is driven by people's needs and wants and can affect groups differently. Students anticipate the needs and wants of diverse end users and propose ways to improve access and usability of technology, with consideration of potential perspectives of users with different backgrounds, ability levels, points of view, and disabilities. For example, students could research a wide variety of disabilities that would limit the use of traditional computational tools for the creation of multimedia artifacts, including digital images, songs, and videos. Students could then brainstorm and propose new software that would allow students that are limited by the disabilities to create similar artifacts in new ways (e.g., graphical display of music for the deaf, the sonification of images for visually impaired students, voice input for those that are unable to use traditional input like the mouse and the keyboard). (CA CCSS for ELA/Literacy W.3.7, W.4.7, W.5.7) Alternatively, as they anticipate unique user needs, students may consider using both speech and text to convey information in a game. They may also wish to vary the types of programs they create, knowing that not everyone shares their own tastes. (CA NGSS: 3-5-ETS1-1, 3-5-ETS1-2, 3-5-ETS1-3)	Impacts of Computing	Culture	Inclusion	1.2
3-5	3-5.IC.22	**Seek and explain the impact of diverse perspectives for the purpose of improving computational artifacts.**	Computing technologies enable global collaboration and sharing of ideas. Students solicit feedback from a diverse group of users and creators and explain how this input improves their computational artifacts. For example, students could seek feedback from classmates via user surveys, in order to create an idea and then make a claim as to how to improve the overall structure and function of their computational artifact. Using the feedback students could write an opinion piece supporting their claim. (CA CCSS for ELA/Literacy W.3.1, W.4.1, W.5.1) Alternatively, with guidance from their teacher, students could use video conferencing tools, shared documents, or other online collaborative spaces, such as blogs, wikis, forums, or website comments, to gather and synthesize feedback from individuals and groups about programming projects. (CA CCSS for ELA/Literacy SL.3.1, SL.4.1, SL.5.1)	Impacts of Computing	Social Interactions	Inclusion	1.1

3-5	3-5.IC.23	**Describe reasons creators might limit the use of their work.**	Ethical complications arise from the opportunities provided by computing. With the ease of sending and receiving copies of media on the Internet, in formats such as video, photos, and music, students consider the opportunities for unauthorized use, such as online piracy and disregard of copyrights. The license of a downloaded image or audio file may restrict modification, require attribution, or prohibit use entirely. For example, students could take part in a collaborative discussion regarding reasons why musicians who sell their songs in digital format choose to license their work so that they can earn money for their creative efforts. If others share the songs without paying for them, the musicians do not benefit financially and may struggle to produce music in the future. (CA CCSS for ELA/Literacy SL.3.1, SL.4.1, SL.5.1) Alternatively, students could review the rights and reproduction guidelines for digital artifacts on a publicly accessible media source. They could then state an opinion with reasons they believe these guidelines are in place. (CA CCSS for ELA/Literacy W.3.1, W.4.1, W.5.1)	Impacts of Computing	Safety Law & Ethics	Communicating	7.3
6-8	6-8.CS.1	**Design modifications to computing devices in order to improve the ways users interact with the devices.**	Computing devices can extend the abilities of humans, but design considerations are critical to make these devices useful. Students suggest modifications to the design of computing devices and describe how these modifications would improve usabilty. For example, students could create a design for the screen layout of a smartphone that is more usable by people with vision impairments or hand tremors. They might also design how to use the device as a scanner to convert text to speech. Alternatively, students could design modifications for a student ID card reader to increase usability by planning for scanner height, need of scanner device to be connected physically to the computer, robustness of scanner housing, and choice of use of RFID or line of sight scanners. (CA NGSS: MS-ETS1-1)	Computing Systems	Devices	Inclusion, Computational Problems	1.2, 3.3
6-8	6-8.CS.2	**Design a project that combines hardware and software components to collect and exchange data.**	Collecting and exchanging data involves input, output, storage, and processing. When possible, students select the components for their project designs by considering tradeoffs between factors such as functionality, cost, size, speed, accessibility, and aesthetics. Students do not need to implement their project design in order to meet this standard. For example, students could design a mobile tour app that displays information relevant to specific locations when the device is nearby or when the user selects a virtual stop on the tour. They select appropriate components, such as GPS or cellular-based geolocation tools, textual input, and speech recognition, to use in their project design. Alternatively, students could design a project that uses a sensor to collect the salinity, moisture, and temperature of soil. They may select a sensor that connects wirelessly through a Bluetooth connection because it supports greater mobility, or they could instead select a physical USB connection that does not require a separate power source. (CA NGSS: MS-ETS1-1, MS-ETS1-2)	Computing Systems	Hardware & Software	Creating	5.1

6-8	6-8.CS.3	**Systematically apply troubleshooting strategies to identify and resolve hardware and software problems in computing systems.**	When problems occur within computing systems, it is important to take a structured, step-by-step approach to effectively solve the problem and ensure that potential solutions are not overlooked. Examples of troubleshooting strategies include following a troubleshooting flow diagram, making changes to software to see if hardware will work, checking connections and settings, and swapping in working components. Since a computing device may interact with interconnected devices within a system, problems may not be due to the specific computing device itself but to devices connected to it. For example, students could work through a checklist of solutions for connectivity problems in a lab of computers connected wirelessly or through physical cables. They could also search for technical information online and engage in technical reading to create troubleshooting documents that they then apply. (CA CCSS for ELA/Literacy RST.6-8.10) Alternatively, students could explore and utilize operating system tools to reset a computer's default language to English. Additionally, students could swap out an externally-controlled sensor giving fluctuating readings with a new sensor to check whether there is a hardware problem.	Computing Systems	Troubleshooting	Testing	6.2
6-8	6-8.NI.4	**Model the role of protocols in transmitting data across networks and the Internet.**	Protocols are rules that define how messages between computers are sent. They determine how quickly and securely information is transmitted across networks, as well as how to handle errors in transmission. Students model how data is sent using protocols to choose the fastest path and to deal with missing information. Knowledge of the details of how specific protocols work is not expected. The priority at this grade level is understanding the purpose of protocols and how they enable efficient and errorless communication. For example, students could devise a plan for sending data representing a textual message and devise a plan for resending lost information. Alternatively, students could devise a plan for sending data to represent a picture, and devise a plan for interpreting the image when pieces of the data are missing. Additionally, students could model the speed of sending messages by Bluetooth, Wi-Fi, or cellular networks and describe ways errors in data transmission can be detected and dealt with.	Networks & the Internet	Network Communication & Organization	Abstraction	4.4

6-8	6-8.NI.5	**Explain potential security threats and security measures to mitigate threats.**	Cybersecurity is an important field of study and it is valuable for students to understand the need for protecting sensitive data. Students identify multiple methods for protecting data and articulate the value and appropriateness for each method. Students are not expected to implement or explain the implementation of such technologies. For example, students could explain the importance of keeping passwords hidden, setting secure router administrator passwords, erasing a storage device before it is reused, and using firewalls to restrict access to private networks. Alternatively, students could explain the importance of two-factor authentication and HTTPS connections to ensure secure data transmission.	Networks & the Internet	Cybersecurity	Computational Problems	3.1, 3.3
6-8	6-8.NI.6	**Apply multiple methods of information protection to model the secure transmission of information.**	Digital information is protected using a variety of cryptographic techniques. Cryptography is essential to many models of cybersecurity. At its core, cryptography has a mathematical foundation. Cryptographic encryption can be as simple as letter substitution or as complicated as modern methods used to secure networks and the Internet. Students encode and decode messages using encryption methods, and explore different levels of complexity used to hide or secure information. For example, students could identify methods of secret communication used during the Revolutionary War (e.g., ciphers, secret codes, invisible ink, hidden letters) and then secure their own methods such as substitution ciphers or steganography (i.e., hiding messages inside a picture or other data) to compose a message from either the Continental Army or British Army. (HSS.8.1) Alternatively, students could explore functions and inverse functions for encryption and decryption and consider functions that are complex enough to keep data secure from their peers. (CA CCSS for Mathematics 8.F.1)	Networks & the Internet	Cybersecurity	Abstraction	4.4
6-8	6-8.DA.7	**Represent data in multiple ways.**	Computers store data as sequences of 0s and 1s (bits). Software translates to and from this low-level representation to higher levels that are understandable by people. Furthermore, higher level data can be represented in multiple ways, such as the digital display of a color and its corresponding numeric RGB value, or a bar graph, a pie chart, and table representation of the same data in a spreadsheet. For example, students could use a color picker to explore the correspondence between the digital display or name of a color (high-level representations) and its RGB value or hex code (low-level representation). Alternatively, students could translate a word (high-level representation) into Morse code or its corresponding sequence of ASCII codes (low-level representation).	Data & Analysis	Storage	Abstraction	4.4

6-8	6-8.DA.8	**Collect data using computational tools and transform the data to make it more useful.**	Data collection has become easier and more ubiquitous. The cleaning of data is an important transformation for ensuring consistent format, reducing noise and errors (e.g., removing irrelevant responses in a survey), and/or making it easier for computers to process. Students build on their ability to organize and present data visually to support a claim, understanding when and how to transform data so information can be more easily extracted. Students also transform data to highlight or expose relationships. For example, students could use computational tools to collect data from their peers regarding the percentage of time technology is used for school work and entertainment, and then create digital displays of their data and findings. Students could then transform the data to highlight relationships representing males and females as percentages of a whole instead of as individual counts. (CA CCSS for Mathematics 6.SP.4, 7.SP.3, 8.SP.1, 8.SP.4) Alternatively, students could collect data from online forms and surveys, from a sensor, or by scraping a web page, and then transform the data to expose relationships. They could highlight the distribution of data (e.g., words on a web page, readings from a sensor) by giving quantitative measures of center and variability. (CA CCSS for Mathematics 6.SP.5.c, 7.SP.4)	Data & Analysis	Collection Visualization & Transformation	Communicating	7.1
6-8	6-8.DA.9	**Test and analyze the effects of changing variables while using computational models.**	Variables within a computational model may be changed, in order to alter a computer simulation or to more accurately represent how various data is related. Students interact with a given model, make changes to identified model variables, and observe and reflect upon the results. For example, students could test a program that makes a robot move on a track by making changes to variables (e.g., height and angle of track, size and mass of the robot) and discussing how these changes affect how far the robot travels. (CA NGSS: MS-PS2-2) Alternatively, students could test a game simulation and change variables (e.g., skill of simulated players, nature of opening moves) and analyze how these changes affect who wins the game. (CA NGSS: MS-ETS1-3) Additionally, students could modify a model for predicting the likely color of the next pick from a bag of colored candy and analyze the effects of changing variables representing the common color ratios in a typical bag of candy. (CA CCSS for Mathematics 7.SP.7, 8.SP.4)	Data & Analysis	Inference & Models	Abstraction, Testing	4.4, 6.1

6-8	6-8.AP.10	**Use flowcharts and/or pseudocode to design and illustrate algorithms that solve complex problems.**	Complex problems are problems that would be difficult for students to solve without breaking them down into multiple steps. Flowcharts and pseudocode are used to design and illustrate the breakdown of steps in an algorithm. Students design and illustrate algorithms using pseudocode and/or flowcharts that organize and sequence the breakdown of steps for solving complex problems. For example, students might use a flowchart to illustrate an algorithm that produces a recommendation for purchasing sneakers based on inputs such as size, colors, brand, comfort, and cost. Alternatively, students could write pseudocode to express an algorithm for suggesting their outfit for the day, based on inputs such as the weather, color preferences, and day of the week.	Algorithms & Programming	Algorithms	Abstraction	4.4, 4.1
6-8	6-8.AP.11	**Create clearly named variables that store data, and perform operations on their contents.**	A variable is a container for data, and the name used for accessing the variable is called the identifier. Students declare, initialize, and update variables for storing different types of program data (e.g., text, integers) using names and naming conventions (e.g. camel case) that clearly convey the purpose of the variable, facilitate debugging, and improve readability. For example, students could program a quiz game with a score variable (e.g. quizScore) that is initially set to zero and increases by increments of one each time the user answers a quiz question correctly and decreases by increments of one each time a user answers a quiz question incorrectly, resulting in a score that is either a positive or negative integer. (CA CCSS for Mathematics 6.NS.5) Alternatively, students could write a program that prompts the user for their name, stores the user's response in a variable (e.g. userName), and uses this variable to greet the user by name.	Algorithms & Programming	Variables	Creating	5.1, 5.2

6-8	6-8.AP.12	**Design and iteratively develop programs that combine control structures and use compound conditions.**	Control structures can be combined in many ways. Nested loops are loops placed within loops, and nested conditionals allow the result of one conditional to lead to another. Compound conditions combine two or more conditions in a logical relationship (e.g., using AND, OR, and NOT). Students appropriately use control structures to perform repetitive and selection tasks. For example, when programming an interactive story, students could use a compound conditional within a loop to unlock a door only if a character has a key AND is touching the door. (CA CCSS for ELA/Literacy W.6.3, W.7.3, W.8.3) Alternatively, students could use compound conditionals when writing a program to test whether two points lie along the line defined by a particular linear function. (CA CCSS for Mathematics 8.EE.7) Additionally, students could use nested loops to program a character to do the "chicken dance" by opening and closing the beak, flapping the wings, shaking the hips, and clapping four times each; this dance "chorus" is then repeated several times in its entirety.	Algorithms & Programming	Control	Creating	5.1, 5.2
6-8	6-8.AP.13	**Decompose problems and subproblems into parts to facilitate the design, implementation, and review of programs.**	Decomposition facilitates program development by allowing students to focus on one piece at a time (e.g., getting input from the user, processing the data, and displaying the result to the user). Decomposition also enables different students to work on different parts at the same time. Students break down (decompose) problems into subproblems, which can be further broken down to smaller parts. Students could create an arcade game, with a title screen, a game screen, and a win/lose screen with an option to play the game again. To do this, students need to identify subproblems that accompany each screen (e.g., selecting an avatar goes in the title screen, events for controlling character action and scoring goes in the game screen, and displaying final and high score and asking whether to play again goes in the win/lose screen). Alternatively, students could decompose the problem of calculating and displaying class grades. Subproblems might include: accept input for students grades on various assignments, check for invalid grade entries, calculate per assignment averages, calculate per student averages, and display histograms of student scores for each assignment. (CA CCSS for Mathematics 6.RP.3c, 6.SP.4, 6.SP.5)	Algorithms & Programming	Modularity	Computational Problems	3.2

6-8	6-8.AP.14	**Create procedures with parameters to organize code and make it easier to reuse.**	Procedures support modularity in developing programs. Parameters can provide greater flexibility, reusability, and efficient use of resources. Students create procedures and/or functions that are used multiple times within a program to repeat groups of instructions. They generalize the procedures and/or functions by defining parameters that generate different outputs for a wide range of inputs. For example, students could create a procedure to draw a circle which involves many instructions, but all of them can be invoked with one instruction, such as "drawCircle." By adding a radius parameter, students can easily draw circles of different sizes. (CA CCSS for Mathematics 7.G.4) Alternatively, calculating the area of a regular polygon requires multiple steps. Students could write a function that accepts the number and length of the sides as parameters and then calculates the area of the polygon. This function can then be re-used inside any program to calculate the area of a regular polygon. (CA CCSS for Mathematics 6.G.1)	Algorithms & Programming	Modularity	Abstraction	4.1, 4.3
6-8	6-8.AP.15	**Seek and incorporate feedback from team members and users to refine a solution that meets user needs.**	Development teams that employ user-centered design processes create solutions (e.g., programs and devices) that can have a large societal impact (e.g., an app that allows people with speech difficulties to allow a smartphone to clarify their speech). Students begin to seek diverse perspectives throughout the design process to improve their computational artifacts. Considerations of the end-user may include usability, accessibility, age-appropriate content, respectful language, user perspective, pronoun use, or color contrast. For example, if students are designing an app to teach their classmates about recycling, they could first interview or survey their classmates to learn what their classmates already know about recycling and why they do or do not recycle. After building a prototype of the app, the students could then test the app with a sample of their classmates to see if they learned anything from the app and if they had difficulty using the app (e.g., trouble reading or understanding text). After gathering interview data, students could refine the app to meet classmate needs. (CA NGSS: MS-ETS1-4)	Algorithms & Programming	Program Development	Collaborating, Inclusion	2.3, 1.1

6-8	6-8.AP.16	**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 use portions of code, algorithms, digital media, and/or data created by others in their own programs and websites. They give attribution to the original creators to acknowledge their contributions. For example, when creating a side-scrolling game, students may incorporate portions of code that create a realistic jump movement from another person's game, and they may also import Creative Commons-licensed images to use in the background. Alternatively, when creating a website to demonstrate their knowledge of historical figures from the Civil War, students may use a professionally-designed template and public domain images of historical figures. (HSS.8.10.5) Additionally, students could import libraries and connect to web application program interfaces (APIs) to make their own programming processes more efficient and reduce the number of bugs (e.g., to check whether the user input is a valid date, to input the current temperature from another city).	Algorithms & Programming	Program Development	Abstraction, Creating, Communicating	4.2, 5.2, 7.3
6-8	6-8.AP.17	**Systematically test and refine programs using a range of test cases.**	Use cases and test cases are created to evaluate whether programs function as intended. At this level, students develop use cases and test cases with teacher guidance. Testing should become a deliberate process that is more iterative, systematic, and proactive than at lower levels. For example, students test programs by considering potential errors, such as what will happen if a user enters invalid input (e.g., negative numbers and 0 instead of positive numbers). Alternatively, in an interactive program, students could test that the character cannot move off of the screen in any direction, cannot move through walls, and can interact with other characters. They then adjust character behavior as needed.	Algorithms & Programming	Program Development	Testing	6.1
6-8	6-8.AP.18	**Distribute tasks and maintain a project timeline when collaboratively developing computational artifacts.**	Collaboration is a common and crucial practice in programming development. Often, many individuals and groups work on the interdependent parts of a project together. Students assume pre-defined roles within their teams and manage the project workflow using structured timelines. With teacher guidance, they begin to create collective goals, expectations, and equitable workloads. For example, students could decompose the design stage of a game into planning the storyboard, flowchart, and different parts of the game mechanics. They can then distribute tasks and roles among members of the team and assign deadlines. Alternatively, students could work as a team to develop a storyboard for an animation representing a written narrative, and then program the scenes individually. (CA CCSS for ELA/Literacy W.6.3, W.7.3, W.8.3)	Algorithms & Programming	Program Development	Collaborating, Creating	2.2, 5.1

6-8	6-8.AP.19	**Document programs in order to make them easier to use, read, test, and debug.**	Documentation allows creators, end users, and other developers to more easily use and understand a program. Students provide documentation for end users that explains their artifacts and how they function (e.g., project overview, user instructions). They also include comments within code to describe portions of their programs and make it easier for themselves and other developers to use, read, test, and debug. For example, students could add comments to describe functionality of different segments of code (e.g., input scores between 0 and 100, check for invalid input, calculate and display the average of the scores). They could also communicate the process used by writing design documents, creating flowcharts, or making presentations. (CA CCSS for ELA/Literacy SL.6.5, SL.7.5, SL.8.5)	Algorithms & Programming	Program Development	Communicating	7.2
6-8	6-8.IC.20	**Compare 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 consider current events related to broad ideas, including privacy, communication, and automation. For example, students could compare and contrast the impacts of computing technologies with the impacts of other systems developed throughout history such as the Pony Express and US Postal System. (HSS.7.8.4) Alternatively, students could identify tradeoffs for both personal and professional uses of a variety of computing technologies. For instance, driverless cars can increase convenience and reduce accidents, but they may be susceptible to hacking. The emerging industry will reduce the number of taxi and shared-ride drivers, but may create more software engineering and cybersecurity jobs.	Impacts of Computing	Culture	Communicating	7.2
6-8	6-8.IC.21	**Discuss issues of bias and accessibility in the design of existing technologies.**	Computing technologies should support users of many backgrounds and abilities. In order to maximize accessiblity, these differences need to be addressed by examining diverse populations. With the teacher's guidance, students test and discuss the usability of various technology tools, such as apps, games, and devices. For example, students could discuss the impacts of facial recognition software that works better for lighter skin tones and recognize that the software was likely developed with a homogeneous testing group. Students could then discuss how accessibility could be improved by sampling a more diverse population. (CA CCSS for ELA/Literacy SL.6.1, SL.7.1, SL.8.1)	Impacts of Computing	Culture	Inclusion	1.2
6-8	6-8.IC.22	**Collaborate with many contributors when creating a computational artifact.**	Users have diverse sets of experiences, needs, and wants. These need to be understood and integrated into the design of computational artifacts. Students use applications that enable crowdsourcing to gather services, ideas, or content from a large group of people. At this level, crowdsourcing can be done at the local level (e.g., classroom, school, or neighborhood) and/or global level (e.g., age-appropriate online communities). For example, a group of students could use electronic surveys to solicit input from their neighborhood regarding an important social or political issue. They could collaborate with a community artist to combine animations and create a digital community collage informing the public about various points of view regarding the topic. (VAPA Visual Art 8.5.2, 8.5.4)	Impacts of Computing	Social Interactions	Collaborating, Creating	2.4, 5.2

6-8	6-8.IC.23	**Compare tradeoffs associated with licenses for computational artifacts to balance the protection of the creators' rights and the ability for others to use and modify the artifacts.**	Using and building on the works of others allows people to create meaningful works and fosters innovation. Copyright is an important law that helps protect the rights of creators so they receive credit and get paid for their work. Creative Commons is a kind of copyright that makes it easier for people to copy, share, and build on creative work, as long as they give credit for it. There are different kinds of Creative Commons licenses that allow people to do things such as change, remix, or make money from their work. As creators, students can pick and choose how they want their work to be used, and then create a Creative Commons license that they include in their work. For example, students could create interactive animations to educate others on bullying or protecting the environment. They then select an appropriate license to reflect how they want their program to be used by others (e.g., allow others to use their work and alter it, as long as they do not make a profit from it). Students use established methods to both protect their artifacts and attribute use of protected artifacts.	Impacts of Computing	Safety Law & Ethics	Communicating	7.3
6-8	6-8.IC.24	**Compare tradeoffs between allowing information to be public and keeping information private and secure.**	While it is valuable to establish, maintain, and strengthen connections between people online, security attacks often start with intentionally or unintentionally providing personal information online. Students identify situations where the value of keeping information public outweighs privacy concerns, and vice versa. They also recognize practices such as phishing and social engineering and explain best practices to defend against them. For example, students could discuss the benefits of artists and designers displaying their work online to reach a broader audience. Students could also compare the tradeoffs of making a shared file accessible to anyone versus restricting it to specific accounts. (CA CCSS for ELA/Literacy SL.6.1, SL.7.1, SL.8.1) Alternatively, students could discuss the benefits and dangers of the increased accessibility of information available on the internet, and then compare this to the advantages and disadvantages of the introduction of the printing press in society. (HSS.7.8.4)	Impacts of Computing	Safety Law & Ethics	Communicating	7.2
9-12	9-12.CS.1	**Describe ways in which abstractions hide the underlying implementation details of computing systems to simplify user experiences.**	An abstraction is a representation of an idea or phenomenon that hides details irrelevant to the question at hand. Computing systems, both stand alone and embedded in products, are often integrated with other systems to simplify user experiences. For example, students could identify geolocation hardware embedded in a smartphone and describe how this simplifies the users experience since the user does not have to enter her own location on the phone. Alternatively, students might select an embedded device such as a car stereo, identify the types of data (e.g., radio station presets, volume level) and procedures (e.g., increase volume, store/recall saved station, mute) it includes, and explain how the implementation details are hidden from the user.	Computing Systems	Devices	Abstraction	4.1

9-12	9-12.CS.2	**Compare levels of abstraction and interactions between application software, system software, and hardware.**	At its most basic level, a computer is composed of physical hardware on which software runs. Multiple layers of software are built upon various layers of hardware. Layers manage interactions and complexity in the computing system. System software manages a computing device's resources so that software can interact with hardware. Application software communicates with the user and the system software to accomplish its purpose. Students compare and describe how application software, system software, and hardware interact. For example, students could compare how various levels of hardware and software interact when a picture is to be taken on a smartphone. Systems software provides low-level commands to operate the camera hardware, but the application software interacts with system software at a higher level by requesting a common image file format (e.g., .png) that the system software provides.	Computing Systems	Hardware & Software	Abstraction	4.1
9-12	9-12.CS.3	**Develop guidelines that convey systematic troubleshooting strategies that others can use to identify and fix errors.**	Troubleshooting complex problems involves the use of multiple sources when researching, evaluating, and implementing potential solutions. Troubleshooting also relies on experience, such as when people recognize that a problem is similar to one they have seen before and adapt solutions that have worked in the past. For example, students could create a list of troubleshooting strategies to debug network connectivity problems such as checking hardware and software status and settings, rebooting devices, and checking security settings. Alternatively, students could create troubleshooting guidelines for help desk employees based on commonly observed problems (e.g., problems connecting a new device to the computer, problems printing from a computer to a network printer).	Computing Systems	Troubleshooting	Testing	6.2
9-12	9-12.NI.4	**Describe issues that impact network functionality.**	Many different organizations, including educational, governmental, private businesses, and private households rely on networks to function adequately in order to engage in online commerce and activity. Quality of Service (QoS) refers to the capability of a network to provide better service to selected network traffic over various technologies from the perspective of the consumer. Students define and discuss performance measures that impact network functionality, such as latency, bandwidth, throughput, jitter, and error rate. For example, students could use online network simulators to explore how performance measures impact network functionality and describe impacts when various changes in the network occur. Alternatively, students could describe how pauses in television interviews conducted over satellite telephones are impacted by networking factors such as latency and jitter.	Networks & the Internet	Network Communication & Organization	Abstraction	4.1

9-12	9-12.NI.5	**Describe the design characteristics of the Internet.**	The Internet connects devices and networks all over the world. Large-scale coordination occurs among many different machines across multiple paths every time a web page is opened or an image is viewed online. Through the domain name system (DNS), devices on the Internet can look up Internet Protocol (IP) addresses, allowing end-to-end communication between devices. The design decisions that direct the coordination among systems composing the Internet also allow for scalability and reliability. Students factor historical, cultural, and economic decisions in their explanations of the Internet. For example, students could explain how hierarchy in the DNS supports scalability and reliability. Alternatively, students could describe how the redundancy of routing between two nodes on the Internet increases reliability and scales as the Internet grows.	Networks & the Internet	Network Communication & Organization	Communicating	7.2
9-12	9-12.NI.6	**Compare and contrast security measures to address various security threats.**	Network security depends on a combination of hardware, software, and practices that control access to data and systems. The needs of users and the sensitivity of data determine the level of security implemented. Potential security problems, such as denial-of-service attacks, ransomware, viruses, worms, spyware, and phishing, present threats to sensitive data. Students compare and contrast different types of security measures based on factors such as efficiency, feasibility, ethical impacts, usability, and security. At this level, students are not expected to develop or implement the security measures that they discuss. For example, students could review case studies or current events in which governments or organizations experienced data leaks or data loss as a result of these types of attacks. Students could provide an analysis of actual security measures taken comparing to other security measure which may have led to different outcomes. Alternatively, students might discuss computer security policies in place at the local level that present a tradeoff between usability and security, such as a web filter that prevents access to many educational sites but keeps the campus network safe.	Networks & the Internet	Cybersecurity	Communication	7.2
9-12	9-12.NI.7	**Compare and contrast cryptographic techniques to model the secure transmission of information.**	Cryptography is a technique for transforming information on a computer in such a way that it becomes unreadable by anyone except authorized parties. Cryptography is useful for supporting secure communication of data across networks. Examples of cryptographic methods include hashing, symmetric encryption/decryption (private key), and assymmetric encryption/decryption (public key/private key). Students use software to encode and decode messages using cryptographic methods. Students compare the costs and benefits of using various cryptographic methods. At this level, students are not expected to perform the mathematical calculations associated with encryption and decryption. For example, students could compare and contrast multiple examples of symmetric cryptographic techiques. Alternatively, students could compare and contrast symmetric and asymmetric cryptographic techniques in which they apply for a given scenario.	Networks & the Internet	Cybersecurity	Computational Problems, Abstraction	3.3, 4.4

9-12	9-12.DA.8	**Translate between different representations of data abstractions of real-world phenomena, such as characters, numbers, and images.**	Computers represent complex real-world concepts such as characters, numbers, and images through various abstractions. Students translate between these different levels of data representations. For example, students could convert an HTML (Hyper Text Markup Language) tag for red font into RGB (Red Green Blue), HEX (Hexadecimal Color Code), HSL (Hue Saturation Lightness), RGBA(Red Green Blue Alpha), or HSLA (Hue Saturation Lightness and Alpha) representations. Alternatively, students could convert the standard representation of a character such as ! into ASCII or Unicode.	Data & Analysis	Storage	Abstraction	4.1
9-12	9-12.DA.9	**Describe tradeoffs associated with how data elements are organized and stored.**	People make choices about how data elements are organized and where data is stored. These choices affect cost, speed, reliability, accessibility, privacy, and integrity. Students describe implications for a given data organziation or storage choice in light of a specific problem. For example, students might consider the cost, speed, reliability, accessibility, privacy, and integrity tradeoffs between storing photo data on a mobile device versus in the cloud. Alternatively, students might compare the tradeoffs between file size and image quality of various image file formats and how choice of format may be infuenced by the device on which it is to be accessed (e.g., smartphone, computer).	Data & Analysis	Storage	Computational Problems	3.3
9-12	9-12.DA.10	**Create data visualizations to help others better understand real-world phenomena.**	People transform, generalize, simplify, and present large data sets in different ways to influence how other people interpret and understand the underlying information. Students select relevant data from large or complex data sets in support of a claim or to communicate the information in a more sophisticated manner. Students use software tools or programming to perform a range of mathematical operations to transform and analyze data and create powerful data visualizations (that reveal patterns in the data). For example, students could create data visualizations to reveal patterns in voting data by state, gender, political affiliation, or socioeconomic status. Alternatively, students could use U.S. government data on criticially endangered animals to visualize population change over time.	Data & Analysis	Collection Visualization & Transformation	Communicating	5.2
9-12	9-12.DA.11	**Refine computational models to better represent the relationships among different elements of data collected from a phenomenon or**	Computational models are used to make predictions about processes or phenomena based on selected data and features. They allow people to investigate the relationships among different variables to understand a system. Predictions are tested to validate models. Students evaluate these models against real-world observations. For example, students could use a population model that allows them to speculate about interactions among different species, evaluate the model based on data gathered from nature, and then refine the model to reflect more complex and realistic interactions.	Data & Analysis	Inference & Models	Abstraction, Testing	4.4, 6.3

9-12	9-12.AP.12	**Design algorithms to solve computational problems using a combination of original and existing algorithms.**	Knowledge of common algorithms improves how people develop software, secure data, and store information. Some algorithms may be easier to implement in a particular programming language, work faster, require less memory to store data, and be applicable in a wider variety of situations than other algorithms. Algorithms used to search and sort data are common in a variety of software applications. For example, students could design an algorithm to calculate and display various sports statistics and use common sorting or mathematical algorithms (e.g., average) in the design of the overall algorithm. Alternatively, students could design an algorithm to implement a game and use existing randomization algorithms to place pieces randomly in starting positions or to control the "roll" of a dice or selection of a "card" from a deck.	Algorithms & Programming	Algorithms	Creating, Abstraction	5.1, 4.2
9-12	9-12.AP.13	**Create more generalized computational solutions using collections instead of repeatedly using simple variables.**	Computers can automate repetitive tasks with algorithms that use collections to simplify and generalize computational problems. Students identify common features in multiple segments of code and substitute a single segment that uses collections (i.e., arrays, sets, lists) to account for the differences. For example, students could take a program that inputs students' scores into multiple variables and modify it to read these scores into a single array of scores. Alternatively, instead of writing one procedure to find averages of student scores and another to find averages of student absences, students could write a single general average procedure to support both tasks.	Algorithms & Programming	Variables	Abstraction	4.1
9-12	9-12.AP.14	**Justify the selection of specific control structures by identifying tradeoffs associated with implementation, readability, and performance.**	The selection of control structures in a given programming language impacts readability and performance. Readability refers to how clear the program is to other programmers and can be improved through documentation. Control structures at this level may include, for example, conditional statements, loops, event handlers, and recursion. Students justify control structure selection and tradeoffs in the process of creating their own computational artifacts. The discussion of performance is limited to a theoretical understanding of execution time and storage requirements; a quantitative analysis is not expected. For example, students could compare the readability and program performance of iterative and recursive implementations of procedures that calculate the Fibonacci sequence. Alternatively, students could compare the readability and performance tradeoffs of multiple if statements versus a nested if statement.	Algorithms & Programming	Control	Creating	5.2

9-12	9-12.AP.15	**Iteratively design and develop computational artifacts for practical intent, personal expression, or to address a societal issue by using events to initiate instructions.**	In this context, relevant computational artifacts can include programs, mobile apps, or web apps. Events can be user-initiated, such as a button press, or system-initiated, such as a timer firing. For example, students might create a tool for drawing on a canvas by first implementing a button to set the color of the pen. Alternatively, students might create a game where many events control instructions executed (e.g., when a score climbs above a threshold, a congratulatory sound is played; when a user clicks on an object, the object is loaded into a basket; when a user clicks on an arrow key, the player object is moved around the screen).	Algorithms & Programming	Control	Creating	5.1, 5.2, 5.3
9-12	9-12.AP.16	**Decompose problems into smaller subproblems through systematic analysis, using constructs such as procedures, modules, and/or classes.**	Decomposition enables solutions to complex problems to be designed and implemented as more manageable subproblems. Students decompose a given problem into subproblems that can be solved using existing functionalities, or new functionalities that they design and implement. For example, students could design a program for supporting soccer coaches in analyzing their teams' statistics. They decompose the problem in terms of managing input, analysis, and output. They decompose the data organization by designing what data will be stored per player, per game, and per team. Team players may be stored as a collection. Data per team player may include: number of shots, misses, saves, assists, penalty kicks, blocks, and corner kicks. Students design methods for supporting various statistical analyses and display options. Students design output formats for individual players or coaches.	Algorithms & Programming	Control	Abstraction	3.2
9-12	9-12.AP.17	**Create computational artifacts using modular design.**	Computational artifacts are created by combining and modifying existing computational artifacts and/or by developing new artifacts. To reduce complexity, large programs can be designed as systems of interacting modules, each with a specific role, coordinating for a common overall purpose. Students should create computational artifacts with interacting procedures, modules, and/or libraries. For example, students could incorporate a physics library into an animation of bouncing balls. Alternatively, students could integrate open-source JavaScript libraries to expand the functionality of a web application. Additionally, students could create their own game to teach Spanish vocabulary words using their own modular design (e.g., including methods to: control scoring, manage wordlists, manage access to different game levels, take input from the user, etc.).	Algorithms & Programming	Modularity	Abstraction, Creating	4.3, 5.2

9-12	9-12.AP.18	**Systematically design programs for broad audiences by incorporating feedback from users.**	Programmers use a systematic design and review process to meet the needs of a broad audience. The process includes planning to meet user needs, developing software for broad audiences, testing users from a cross-section of the audience, and refining designs based on feedback. For example, students could create a user satisfaction survey and brainstorm distribution methods to collect feedback about a mobile application. After collecting feedback from a diverse audience, students could incorporate feedback into their product design. Alternatively, while developing an e-textiles project with human touch sensors, students could collect data from peers and identify design changes needed to improve usability by users of different needs.	Algorithms & Programming	Program Development	Inclusion, Creating	1.1, 5.1
9-12	9-12.AP.19	**Explain the limitations of licenses that restrict use of computational artifacts when using resources such as libraries.**	Software licenses include copyright, freeware, and open-source licensing schemes. Licenses are used to protect the intellectual property of the author while also defining accessibility of the code. Students consider licensing implications for their own work, especially when incorporating libraries and other resources. For example, students might consider two software libraries that address a similar need, justifying their choice of one over the other. The choice could be based upon least restrictive licensing or further protections for their own intellectual property.	Algorithms & Programming	Program Development	Communicating	7.3
9-12	9-12.AP.20	**Iteratively evaluate and refine a computational artifact to enhance its performance, reliability, usability, and accessibility.**	Evaluation and refinement of computational artifacts involves measuring, testing, debugging, and responding to the changing needs and expectations of users. Aspects that can be evaluated include correctness, performance, reliability, usability, and accessibility. For example, after witnessing common errors with user input in a computational artifact, students could refine the artifact to validate user input and provide an error message if invalid data is provided. Alternatively, students could observe a robot in a variety of lighting conditions to determine whether the code controlling a light sensor should be modified to make it less sensitive. Additionally, students could also incorporate feedback from a variety of end users to help guide the size and placement of menus and buttons in a user interface.	Algorithms & Programming	Program Development	Testing	6.3

9-12	9-12.AP.21	**Design and develop computational artifacts working in team roles using collaborative tools.**	Collaborative tools can be as complex as a source code version control system or as simple as a collaborative word processor. Team roles in pair programming are driver and navigator but students can take on more specialized roles in larger teams. Teachers or students should choose resources that aid collaborative program development as programs grow more complex. For example, students might work as a team to develop a mobile application that addresses a problem relevant to the school or community, using appropriate tools to support actions such as: establish and manage the project timeline; design, share, and revise graphical user interface elements; implement program components, track planned, in-progress, and completed components, and design and implement user testing.	Algorithms & Programming	Program Development	Collaborating	2.4
9-12	9-12.AP.22	**Document decisions made during the design process using text, graphics, presentations, and/or demonstrations in the development of complex programs.**	Complex programs are often iteratively designed as systems of interacting modules, each with a specific role, coordinating for a common overall purpose. Comments are included in code both to document the purpose of modules as well as the implementation details within a module. Together these support documentation of the design process. Students use resources such as libraries and tools to edit and manage parts of the program and corresponding documentation. For example, during development of a computational artifact students could comment their code (with date, modification, and rationale), sketch a flowchart to summarize control flow in a code journal, and share ideas and updates on a white board. Students may document their logic by explaining the development process and presenting to the class. The presentation could include photos of their white board, a video or screencast explaining the development process, or recorded audio description.	Algorithms & Programming	Program Development	Communicating	7.2
9-12	9-12.IC.23	**Evaluate the ways computing impacts personal, ethical, social, economic, and cultural practices.**	Computing may improve, harm, or maintain practices. An understanding of how equity deficits, such as minimal exposure to computing, access to education, and training opportunities, are related to larger, systemic problems in society enables students to create more meaningful artifacts. Students illustrate the positive, negative, and/or neutral impacts of computing. For example, students could evaluate the accessibility of a product for a broad group of end users, such as people who lack access to broadband or who have various disabilities. Students could identify potential bias during the design process and evaluate approaches to maximize accessibility in product design. Alternatively, students could evaluate the impact of social media on cultural, economic, and social practices around the world.	Impacts of Computing	Culture	Computational Problems, Inclusion	3.1, 1.2

9-12	9-12.IC.24	**Identify impacts of bias and equity deficit on design and implementation of computational artifacts and apply appropriate processes for evaluating issues of bias.**	Biases could include incorrect assumptions developers have made about their users, including minimal exposure to computing, access to education, and training opportunities. Students identify and use strategies to test and refine computational artifacts with the goal of reducing bias and equity deficits and increasing universal access. For example, students could use a spreadsheet to chart various forms of equity deficits, and identify solutions in existing software. Students could use and refine the spreadsheet solutions to create a strategy for methodically testing software specifically for bias and equity.	Impacts of Computing	Culture	Inclusion	1.2
9-12	9-12.IC.25	**Demonstrate ways a given algorithm applies to problems across disciplines.**	Students identify how a given algorithm can be applied to real-world problems in different disciplines. For example, students could demonstrate how a randomization algorithm can be used to select participants for a clinical medical trial or to select a flash card to display on a vocabulary quiz. Alternatively, students could demonstrate how searching and sorting algorithms are needed to organize records in manufacturing settings, or to support doctors queries of patient records, or to help governments manage support services they provide to their citizens.	Impacts of Computing	Culture	Computational Problems	3.1
9-12	9-12.IC.26	**Study, discuss, and think critically about the potential impacts and implications of emerging technologies on larger social, economic, and political structures, with evidence from credible sources.**	For example, after studying the rise of artifical intelligence, students create a cause and effect chart to represent positive and negative impacts of this technology on society.	Impacts of Computing	Culture	Communciating	7.2

9-12	9-12.IC.27	**Use collaboration tools and methods to increase connectivity with people of different cultures and careers.**	Increased digital connectivity and communication between people across a variety of cultures and in differing professions has changed the collaborative nature of personal and professional interaction. Students identify, explain, and use appropriate collaborative tools. For example, students could compare ways that various technological collaboration tools could help a team become more cohesive and then choose one of these tools to manage their teamwork. Alternatively, students could use different collaborative tools and methods to solicit input from not only team members and classmates but also others, such as participants in online forums or local communities.	Impacts of Computing	Social Interactions	Collaborating	2.4
9-12	9-12.IC.28	**Explain the beneficial and harmful effects that intellectual property laws can have on innovation.**	Laws and ethics govern aspects of computing such as privacy, data, property, information, and identity. Students explain the beneficial and harmful effects of intellectual property laws as they relate to potential innovations and governance. For example, students could explain how patents protect inventions but may limit innovation. Alternatively, students could explain how intellectual property laws requiring that artists be paid for use of their media might limit the choice of songs developers can use in their computational artifacts.	Impacts of Computing	Safety Law & Ethics	Communicating	7.3
9-12	9-12.IC.29	**Explain the privacy concerns related to the collection and generation of data through automated processes.**	Data can be collected and aggregated across millions of people, even when they are not actively engaging with or physically near the data collection devices. Students recognize automated and non-evident collection of information and the privacy concerns they raise for individuals. For example, students could explain the impact on an individual when a social media site's security settings allows for mining of account information even when the user is not online. Alternatively, students could discuss the impact on individuals of using surveillance video in a store to track customers. Additionally, students could discuss how road traffic can be monitored to change signals in real time to improve road efficiency without drivers being aware and discuss policies for retaining data that identifies drivers' cars and their behaviors.	Impacts of Computing	Safety Law & Ethics	Communicating	7.2

9-12	9-12.IC.30	**Evaluate the social and economic implications of privacy in the context of safety, law, or ethics.**	Laws govern many aspects of computing, such as privacy, data, property, information, and identity. International differences in laws and ethics have implications for computing. Students make and justify claims about potential and/or actual privacy implications of policies, laws, or ethics and consider the associated tradeoffs, focusing on society and the economy. For example, students could explore the case of companies tracking online shopping behaviors in order to decide which products to target to consumers. Students could evaluate the ethical and legal dilemmas of collecting such data without consumer knowledge in order to profit companies. Alternatively, students could evaluate the implications of net neutrality laws on society's access to information and on the impacts to businesses of varying sizes.	Impacts of Computing	Safety Law & Ethics	Communicating	7.2
9-12 Specialty	9-12S.CS.1	**Illustrate ways computing systems implement logic through hardware components.**	Computing systems use processors (e.g., a central processing unit or CPU) to execute program instructions. Processors are composed of components that implement the logical or computational operations required by the instructions. AND, OR, and NOT are examples of logic gates. Adders are examples of higher-leveled circuits built using low-level logic gates. Students illustrate how modern computing devices are made up of smaller and simpler components which implement the logic underlying the functionality of a computer processor. At this level, knowledge of how logic gates are constructed is not expected. For example, students could construct truth tables, draw logic circuit diagrams, or use an online logic circuit simulator. Students could explore the interaction of the CPU, RAM, and I/O by labeling a diagram of the von Neumann architecture. Alternatively, students could design higher-level circuits using low-level logic gates (e.g., adders).	Computing Systems	Devices	Communicating, Abstractions	7.2, 4.4
9-12 Specialty	9-12S.CS.2	**Categorize and describe the different functions of operating system software.**	Operating systems (OS) software is the code that manages the computer's basic functions. Students describe at a high level the different functions of different components of operating system software. Examples of functions could include memory management, data storage/retrieval, processes management, and access control. For example, students could use monitoring tools including within an OS to inspect the services and functions running on a system and create an artifact to describe the activity that they observed (e.g., when a browser is running with many tabs open, memory usage is increased). They could also inspect and describe changes in the activity monitor that occur as different applications are executing (e.g., processor utilization increases when a new application is launched).	Computing Systems	Hardware & Software	Communicating	7.2

9-12 Specialty	9-12S.NI.3	**Examine the scalability and reliability of networks, by describing the relationship between routers, switches, servers, topology, and addressing.**	Choice of network topology is determined, in part, by how many devices can be supported and the character of communication needs between devices. Each device is assigned an address that uniquely identifies it on the network. Routers function by comparing addresses to determine how information on the network should reach its desgination. Switches compare addresses to determine which computers will receive information. Students explore and explain how network performance degrades when various factors affect the network. For example, students could use online network simulators to describe how network performance changes when the number of devices increases. Alternatively, students could visualize and describe changes to the distribution of network traffic when a router on the network fails.	Networks & the Internet	Network Communication & Organization	Abstractions	4.4
9-12 Specialty	9-12S.NI.4	**Explain how the characteristics of the Internet influence the systems developed on it.**	The design of the Internet includes hierarchy and redundancy to help it scale reliably. An end-to-end architecture means that key functions are placed at endpoints in the network (i.e., an Internet user's computer and the server hosting a website) rather than in the middle of the network. Open standards for transmitting information across the Internet help fuel its growth. This design philosophy impacts systems and technologies that integrate with the Internet. Students explain how Internet-based systems depend on these characteristics. For example, students could explain how having common, standard protocols enable products and services from different developers to communicate. Alternatively, students could describe how the end-to-end architecture and redundancy in routing enables Internet users to access information and services even if part of the network is down; the information can still be routed from one end to another through a different path.	Networks & the Internet	Network Communication & Organization	Communicating	7.2
9-12 Specialty	9-12S.NI.5	**Develop solutions to security threats.**	Designing and implementing cybersecurity measures requires knowledge of software, hardware, and human components and understanding tradeoffs. Students design solutions to security threats and compare tradeoffs of easier access and use against the costs of losing information and disrupting services. For example, students could refine a technology that allows users to use blank or weak passwords. Alternatively, students could implement a firewall or proxy protection between an organization's private local area network (LAN) and the public Internet. Additionally, students could find and close exploitable threats on an infected computer in order to protect information.	Networks & the Internet	Cybersecurity	Creating	5.3

9-12 Specialty	9-12S.NI.6	**Analyze cryptographic techniques to model the secure transmission of information.**	Cryptography is essential to many models of cybersecurity. Open standards help to ensure cryptographic security. Certificate Authorities (CAs) issue digital certificates that validate the ownership of encrypted keys used in secured communications across the Internet. Students encode and decode messages using encryption and decryption methods, and they should understand the different levels of complexity to hide or secure information. For example, students could analyze the relative designs of private key vs. public key encryption techniques and apply the best choice for a particular scenario. Alternatively, students could analyze the design of the Diffie-Helman algorithm to RSA (Rivest–Shamir–Adleman) and apply the best choice for a particular scenario. They could provide a cost-benefit analysis of runtime and ease of cracking for various encryption techniques which are commonly used to secure transmission of data over the Internet.	Networks & the Internet	Cybersecurity	Computational Problems, Abstractions	3.3, 4.2
9-12 Specialty	9-12S.DA.7	**Select and use data collection tools and techniques to generate data sets.**	Data collection and organization is essential for obtaining new information insights and revealing new knowledge in our modern world. As computers are able to process larger sets of data, gathering data in an efficient and reliable matter remains important. The choice of data collection tools and quality of the data collected influences how new information, insights, and knowledge will support claims and be communicated. Students devise a reliable method to gather information, use software to extract digital data from data sets, and clean and organize the data in ways that support summaries of information obtained from the data. At this level, students may, but are not required to, create their own data collection tools. For example, students could create a computational artifact that records information from a sonic distance sensor to monitor the motion of a prototype vehicle. Alternatively, students could develop a reliable and practical way to automatically digitally record the number of animals entering a portion of a field to graze. Additionally, students could also find a web site containing data (e.g., race results for a major marathon), scrape the data from the web site using data collection tools, and format the data so it can be analyzed.	Data & Analysis	Collection Visualization & Transformation	Communicating	7.1
9-12 Specialty	9-12S.DA.8	**Use data analysis tools and techniques to identify patterns in data representing complex systems.**	Data analysis tools can be useful for identifying patterns in large amounts of data in many different fields. Computers can help with the processing of extremely large sets of data making very complex systems manageable. Students use computational tools to analyze, summarize, and visualize a large set of data. For example, students could analyze a data set containing marathon times and determine how age, gender, weather, and course features correlate with running times. Alternatively, students could analyze a data set of social media interactions to identify the most influential users and visualize the intersections between different social groups.	Data & Analysis	Collection Visualization & Transformation	Communicating, Abstraction	7.1, 4.1

9-12 Specialty	9-12S.DA.9	**Evaluate the ability of models and simulations to test and support the refinement of hypotheses.**	A model could be implemented as a diagram or a program that represents key properties of a physical or other system. A simulation is based on a model, and enables observation of the system as key properties change. Students explore, explain, and evaluate existing models and simulations, in order to support the refinement of hypotheses about how the systems work. At this level, the ability to accurately and completely model and simulate complex systems is not expected. For example, a computer model of ants following a path created by other ants who found food explains the trail-like travel patterns of the insect. Students could evaluate if the output of the model fits well with their hypothesis that ants navigate the world through the use of pheromones. They could explain how the computer model supports this hypothesis and how it might leave out certain aspects of ant behavior and whether these are important to understanding ant travel behavior. Alternatively, students could hypothesize how different ground characteristics (e.g., soil type, thickness of sediment above bedrock) relate to the severity of shaking at the surface during an earthquake. They could add or modify input about ground characteristics into an earthquake simulator, observe the changed simulation output, and then evaluate their hypotheses.	Data & Analysis	Inference & Models	Abstraction	4.4
9-12 Specialty	9-12S.AP.10	**Describe how artificial intelligence drives many software and physical systems.**	Artificial intelligence is a sub-discipline of computer science that enables computers to solve problems previously handled by biological systems. There are many applications of artificial intelligence, including computer vision and speech recognition. Students research and explain how artificial intelligence has been employed in a given system. Students are not expected to implement an artificially intelligent system in order to meet this standard. For example, students could observe an artificially intelligent system and notice where its behavior is not human-like, such as when a character in a videogame makes a mistake that a human is unlikely to make, or when a computer easily beats even the best human players at a given game. Alternatively, students could interact with a search engine asking various questions, and after reading articles on the topic, they could explain how the computer is able to respond to queries.	Algorithms & Programming	Algorithms	Communicating, Computational Problems	7.2, 3.1
9-12 Specialty	9-12S.AP.11	**Implement an algorithm that uses artificial intelligence to overcome a simple challenge.**	Artificial intelligence algorithms allow a computer to perceive and move in the world, use knowledge, and engage in problem solving. Students create a computational artifact that is able to carry out a simple task commonly performed by living organisms. Students do not need to realistically simulate human behavior or solve a complex problem in order to meet this standard. For example, students could implement an algorithm for playing tic-tac-toe that would select an appropriate location for the next move. Alternatively, students could implement an algorithm that allows a solar-powered robot to move to a sunny location when its batteries are low.	Algorithms & Programming	Algorithms	Creating, Computational Problems	5.3, 3.1

9-12 Specialty	9-12S.AP.12	**Implement searching and sorting algorithms to solve computational problems.**	One of the core uses of computers is to store, organize, and retrieve information when working with large amounts of data. Students create computational artifacts that use searching and/or sorting algorithms to retrieve, organize, or store information. Students do not need to select their algorithm based on efficiency. For example, students could write a script to sequence their classmates in order from youngest to oldest. Alternatively, students could write a program to find certain words within a text and report their location.	Algorithms & Programming	Algorithms	Abstraction, Creating	4.2, 5.2
9-12 Specialty	9-12S.AP.13	**Evaluate algorithms in terms of their efficiency.**	Algorithms that perform the same task can be implemented in different ways, which take different amounts of time to run on a given input set. Algorithms are commonly evaluated using asymptotic analysis (i.e., "Big O") which involves exploration of behavior when the input set grows very large. Students classify algorithms by the most common time classes (e.g., log n, linear, n log n, and quadratic or higher). For example, students could read a given algorithm, identify the control constructs, and in conjunction with input size, identify the efficiency class of the algorithm.	Algorithms & Programming	Algorithms	Abstraction	3.3
9-12 Specialty	9-12S.AP.14	**Compare and contrast fundamental data structures and their uses.**	Data structures are designed to provide different ways of storing and manipulating data sets to optimize various aspects of storage or runtime performance. Choice of data structures is made based on expected data characteristics and expected program functions. Students = compare and contrast how basic functions (e.g.., insertion, deletion, and modification) would differ for common data structures including lists, arrays, stacks, and queues. For example, students could draw a diagram of how different data structures change when items are added, deleted, or modified. They could explain tradeoffs in storage and efficiency issues. Alternatively, when presented with a description of a program and the functions it would be most likely to be running, students could list pros and cons for a specific data structure use in that scenario.	Algorithms & Programming	Variables	Abstraction	4.2
9-12 Specialty	9-12S.AP.15	**Demonstrate the flow of execution of a recursive algorithm.**	Recursion is a powerful problem solving approach where the problem solution is built on solutions of smaller instances of the same problem. A base case, which returns a result without referencing itself, must be defined, otherwise infinite recursion will occur. Students represent a sequence of calls to a recursive algorithm and show how the process resolves to a solution. For example, students could draw a diagram to illustrate flow of execution by keeping track of parameter and returned values for each recursive call. Alternatively, students could create a video showing the passing of arguments as the recursive algorithm runs.	Algorithms & Programming	Control	Computational Problems, Communicating	3.2, 7.2

9-12 Specialty	9-12S.AP.16	**Analyze a large-scale computational problem and identify generalizable patterns or problem components that can be applied to a solution.**	As students encounter complex, real-world problems that span multiple disciplines or social systems, they need to be able to decompose problems and apply already developed code as part of their solutions. Students decompose complex problems into manageable subproblems that could potentially be solved with programs or procedures that can be reused or already exist. For example, in analyzing an Internet radio app, students could identify that users need to create an account and enter a password. They could identify a common application programming interface (API) for checking and displaying password strength. Additionally, students could recognize that the songs would need to be sorted by the time last played in order to display the most recently played songs and identify a common API for sorting dates from most to least recent. Alternatively, in analyzing the problem of tracking medical treatment in a hospital, students could recognize that patient records need to be stored in a database and identify a database solution to support quick access and modification of patient records. Additionally, they could recognize that records in the database need to be stored securely and could identify an encryption API to support the desired level of privacy.	Algorithms & Programming	Modularity	Computational Problems, Abstraction	3.2, 4.2
9-12 Specialty	9-12S.AP.17	**Construct solutions to problems using student-created components, such as procedures, modules, and/or objects.**	Programmers often address complex tasks through design and decomposition using procedures and/or modules. In object-oriented programming languages, classes can support this decomposition. Students create a computational artifact that solves a problem through use of procedures, modules, and/or objects. This problem should be of sufficient complexity to benefit from decomposition and/or use of objects. For example, students could write a flashcard program in which each card is able to show both the question and answer and record user history. Alternatively, students could create a simulation of an ecosystem in which sprites carry out behaviors, such as consuming resources.	Algorithms & Programming	Modularity	Abstraction, Creating	4.3, 5.2
9-12 Specialty	9-12S.AP.18	**Demonstrate code reuse by creating programming solutions using libraries and APIs.**	Code reuse is critical both for managing complexity in modern programs, but also in increasing programming efficiency and reliability by having programmers reuse code that has been highly vetted and tested. Software libraries allow developers to integrate common and often complex functionality without having to reimplement that functionality from scratch. Students identify, evaluate, and select appropriate application programming interfaces (APIs) from software libraries to use with a given language and operating system. They appropriately use resources such as technical documentation, online forums, and developer communities to learn about libraries and troubleshoot problems with APIs that they have chosen. For example, students could import charting and graphing modules to display data sets, adopt an online service that provides cloud storage and retrieval for a database used in a multiplayer game, or import location services into an app that identifies points of interest on a map. Libraries of APIs can be student-created or publicly available (e.g., common graphics libraries or map/navigation APIs).	Algorithms & Programming	Modularity	Abstractions, Creating, Troubleshooting	4.2, 5.3, 6.2

9-12 Specialty	9-12S.AP.19	**Plan and develop programs for broad audiences using a specific software life cycle process.**	Software development processes are used to help manage the design, development, and product/project management of a software solution. Various types of processes have been developed over time to meet changing needs in the software landscape. The systems development life cycle (SDLC), also referred to as the application development life cycle, is a term used in systems engineering, information systems, and software engineering to describe a process for planning, creating, testing, and deploying an information system. Other examples of common processes could include agile, spiral, or waterfall. Students develop a program following a specific software life cycle process, with proper scaffolding from the teacher. For example, students could work in teams on a common project using the agile development process, which is based on breaking product development work into small increments. Alternatively, students could be guided in implementing sprints to focus work on daily standup meetings or scrums to support efficient communication.	Algorithms & Programming	Program Development	Collaborating, Creating	2.2, 2.3, 5.2
9-12 Specialty	9-12S.AP.20	**Develop programs for multiple computing platforms.**	Humans use computers in various forms in their lives and work. Depending on the situation, software solutions are more appropriate or valuable when available on different computational platforms or devices. Students develop programs for more than one computing platform (e.g. desktop, web, or mobile). For example, students could develop a mobile app for a location-aware software product and a different program that is installed on a computer. Alternatively, students could create a browser-based product and make it accessible across multiple platforms or computers (e.g., email).	Algorithms & Programming	Program Development	Creating	5.2
9-12 Specialty	9-12S.AP.21	**Identify and fix security issues that might compromise computer programs.**	Some common forms of security issues arise from specific programming languages, platforms, or program implementation choices. Students read a given a piece of code that contains a common security vulnerability, explain the code's intended function or purpose, provide and explain examples of how a specific input could exploit that vulnerability (e.g., the program accessing data or performing in unintended ways), and implement a change in the code to mitigate this vulnerability. For example, students could review code that takes a date as input, recognize that the code doesn't check for appropriate last days of the month, and modify the code to do that. Alternatively, students could review code that supports entry of patient data (e.g., height and weight) and doesn't prompt users to double check unreasonable values (e.g., height at 6 feet and weight at 20 pounds).	Algorithms & Programming	Program Development	Troubleshooting	6.2

9-12 Specialty	9-12S.AP.22	**Develop and use a series of test cases to verify that a program performs according to its design specifications.**	Testing software is a critically important process. The ability of students to identify a set of important test cases communicates their understanding of the design specifications and potential issues due to implementation choices. Students select and apply their own test cases to cover both general behavior and the edge cases which show behavior at boundary conditions. For example, for a program that is supposed to accept test scores in the range of [0,100], students could develop appropriate tests (e.g, a negative value, 0, 100, and a value above 100). Alternatively, students developing an app to allow users to create and store calendar appointments could develop and use a series of test cases for various scenarios including checking for correct dates, flagging for user confirmation when a calendar event is very long, checking for correct email address format for invitees, and checking for appropriate screen display as users go through the process of adding, editing, and deleting events.	Algorithms & Programming	Program Development	Testing	6.1
9-12 Specialty	9-12S.AP.23	**Modify an existing program to add additional functionality and discuss intended and unintended implications.**	Modularity and code reuse is key in modern software. However, when code is modified, the programmer should consider relevant situations in which this code might be used in other places. Students create and document modifications to existing programs that enhance functionality, and then identify, document, and correct unintended consequences. For example, students could take an existing a procedure that calculates the average of a set of numbers and returns an integer (which lacks precision) and modify it to return a floating point number instead. The student would explain how the change might impact multiple scenarios.	Algorithms & Programming	Program Development	Creating, Abstraction	5.3, 4.2
9-12 Specialty	9-12S.AP.24	**Evaluate key qualities of a program through a process such as a code review.**	Code reviews are a common software industry practice and valuable for developing technical communication skills. Key qualities of code include correctness, usability, readability, efficiency, and scalability. Students walk through code they created and explain how it works. Additionally, they follow along when someone else is explaining their code and ask appropriate questions. For example, students could present their code to a group or visually inspect code in pairs. Alternatively, in response to another student's presentation, students could provide feedback including comments on correctness of the code, comments on how code interacts with code that calls it, and design and documentation features.	Algorithms & Programming	Program Development	Testing	6.3

9-12 Specialty	9-12S.AP.25	**Use version control systems, integrated development environments (IDEs), and collaborative tools and practices (e.g., code documentation) while developing software within a group.**	Software development is a process that benefits from the use of tools that manage complexity, iterative development, and collaboration. Large or complex software projects often require contributions from multiple developers. Version control systems and other collaborative tools and practices help coordinate the process and products contributed by individuals on a development team. An integrated development environment (IDE) is a program within which a developer implements, compiles or interprets, tests, debugs, and deploys a software project. Students use common software development and documentation support tools in the context of a group software development project. At this level, facility with the full functionality available in the collaborative tools is not expected. For example, students could use common version control systems to modify and improve code or revert to a previous code version. Alternatively, students could use appropriate IDEs to support more efficient code design and development. Additionally, students could use various collaboration, communication, and code documentation tools designed to support groups engaging in complex and interrelated work.	Algorithms & Programming	Program Development	Collaborating, Creating	2.4, 5.2
9-12 Specialty	9-12S.AP.26	**Compare multiple programming languages, and discuss how their features make them suitable for solving different types of problems.**	Particular problems may be more effectively solved using some programming languages than other programming languages. Students provide a rationale for why a specific programming language is better suited for a solving a particular class of problem. For example, students could explain how a language with a large library base can make developing a web application easier. Alternatively, students could explain how languages that support particular programming paradigms (e.g., object-oriented or functional) can make implementation more aligned with design choices. Additionally, students could discuss how languages that implement garbage collection are good for simplicity of memory management, but may result in poor performance characteristics	Algorithms & Programming	Program Development	Communicating	7.2

9-12 Specialty	9-12S.IC.27	**Evaluate computational artifacts with regard to improving their beneficial effects and reducing harmful effects on society.**	People design computational artifacts to help make the lives of humans better. Students evaluate an artifact and comment on aspects of it which positively or negatively impact users and give ideas for reducing the possible negative impacts. For example, students could discuss how algorithms that screen job candidates' resumes can cut costs for companies (a beneficial effect) but introduce or amplify bias in the hiring process (a harmful effect). Alternatively, students could discuss how turn-by-turn navigation tools can help drivers avoid traffic and find alternate routes (a beneficial effect), but sometimes channel large amounts of traffic down small neighborhood streets (a harmful effect). Additionally, students could discuss how social media algorithms can help direct users' attention to interesting content (a beneficial effect), while simultaneously limiting users' exposure to information that contradicts pre-existing beliefs (a harmful effect).	Impacts of Computing	Culture	Testing, Inclusion	6.1, 1.2
9-12 Specialty	9-12S.IC.28	**Evaluate how computational innovations that have revolutionized aspects of our culture might evolve.**	It is important to be able to evaluate current technologies and innovations and their potential for future impact on society. Students describe how a given computational innovation might change in the future and impacts these evolutions could have on society, economy, or culture. For example, students could consider ways in which computers may support education (or healthcare) in the future, or how developments in virtual reality might impact arts and entertainment. Alternatively, students could consider how autonomous vehicles will affect individuals' car ownership and car use habits as well as industries that employ human drivers (e.g., trucking, taxi service).	Impacts of Computing	Culture	Communicating	7.2
9-12 Specialty	9-12S.IC.29	**Evaluate the impact of equity, access, and influence on the distribution of computing resources in a global society.**	Computers, computation, and technology can help improve the lives of humans and support positive developments in society, economy, and/or culture. However, access to such resources is not the same for everyone in the world. Students define and evaluate ways in which different technologies, applications, or computational tools might benefit all people in society or might only benefit those with the greatest access or resources. For example, students could describe ways in which groups of people benefit, do not benefit, or could benefit better by access to high-speed Internet connectivity. Alternatively, students could describe educational impacts of children not having access to a computer in their home.	Impacts of Computing	Culture	Inclusion	1.2

9-12 Specialty	9-12S.IC.30	**Debate laws and regulations that impact the development and use of software.**	Laws and regulations influence what software gets developed and how society benefits or does not. For example, students could debate the pros and cons of changes to regulations around net neutrality: Many believe that mandating that Internet service providers (ISPs) maintain net neutrality facilitates competition between Internet-based content providers and supports consumer choice, but others believe such regulations represent government overreach. Alternatively, students could debate the impacts of different copyright rules in various countries and impacts on economy, society, and culture: Long-lasting copyrights in the United States enable creators to profit from their works but also prevent works from entering the public domain where they can be freely used and adapted to create new works.	Impacts of Computing	Safety Law & Ethics	Communicating	7.2