CSTA Standards

Identifier:	Grade:	Standard:	Concept:	Practice(s):
1A-AP-08	K-2	**Model daily processes by creating and following algorithms (sets of step-by-step instructions) to complete tasks.** *Composition is the combination of smaller tasks into more complex tasks. Students could create and follow algorithms for making simple foods, brushing their teeth, getting ready for school, participating in cleanup time.* **Practice(s):** Developing and Using Abstractions: 4.4	Algorithms & Programming	Abstraction
1A-AP-09	K-2	**Model the way programs store and manipulate data by using numbers or other symbols to represent information.** *Information in the real world can be represented in computer programs. Students could use thumbs up/down as representations of yes/no, use arrows when writing algorithms to represent direction, or encode and decode words using numbers, pictographs, or other symbols to represent letters or words.* **Practice(s):** Developing and Using Abstractions: 4.4	Algorithms & Programming	Abstraction
1A-AP-10	K-2	**Develop programs with sequences and simple loops, to express ideas or address a problem.** *Programming is used as a tool to create products that reflect a wide range of interests. Control structures specify the order in which instructions are executed within a program. Sequences are the order of instructions in a program. For example, if dialogue is not sequenced correctly when programming a simple animated story, the story will not make sense. If the commands to program a robot are not in the correct order, the robot will not complete the task desired. Loops allow for the repetition of a sequence of code multiple times. For example, in a program to show the life cycle of a butterfly, a loop could be combined with move commands to allow continual but controlled movement of the character.* **Practice(s):** Creating Computational Artifacts: 5.2	Algorithms & Programming	Creating
1A-AP-11	K-2	**Decompose (break down) the steps needed to solve a problem into a precise sequence of instructions.** *Decomposition is the act of breaking down tasks into simpler tasks. 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.* **Practice(s):** Recognizing and Defining Computational Problems: 3.2	Algorithms & Programming	Computational Problems

1A-AP-12	K-2	**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 a program and can be used to check if a program is correct. Students could create a planning document, such as a story map, a storyboard, or a sequential graphic organizer, to illustrate what their program will do. Students at this stage may complete the planning process with help from their teachers.* **Practice(s):** Creating Computational Artifacts, Communicating About Computing: 5.1, 7.2	Algorithms & Programming	Creating, Communicating
1A-AP-13	K-2	**Give attribution when using the ideas and creations of others while developing programs.** *Using computers comes with a level of responsibility. Students should 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 or orally, if sharing work on a class blog or website. Proper attribution at this stage does not require a formal citation, such as in a bibliography or works cited document.* **Practice(s):** Communicating About Computing: 7.3	Algorithms & Programming	Communicating
1A-AP-14	K-2	**Debug (identify and fix) errors in an algorithm or program that includes sequences and simple loops.** *Algorithms or programs may not always work correctly. Students should be able to 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.* **Practice(s):** Testing and Refining Computational Artifacts: 6.2	Algorithms & Programming	Testing
1A-AP-15	K-2	**Using correct terminology, describe steps taken and choices made during the iterative process of program development.** *At this stage, students should be able to talk or write about the goals and expected outcomes of the programs they create and the choices that they made when creating programs. This could be done using coding journals, discussions with a teacher, class presentations, or blogs.* **Practice(s):** Communicating About Computing: 7.2	Algorithms & Programming	Communicating
		Select and operate appropriate software to perform a variety of tasks, and recognize that users have different needs and preferences for the technology they use.		

1A-CS-01	K-2	*People use computing devices to perform a variety of tasks accurately and quickly. Students should be able to select the appropriate app/program to use for tasks they are required to complete. For example, if students are asked to draw a picture, they should be able to open and use a drawing app/program to complete this task, or if they are asked to create a presentation, they should be able to open and use presentation software. In addition, with teacher guidance, students should compare and discuss preferences for software with the same primary functionality. Students could compare different web browsers or word processing, presentation, or drawing programs.* **Practice(s):** Fostering an Inclusive Computing Culture: 1.1	Computing Systems	Inclusion
1A-CS-02	K-2	**Use appropriate terminology in identifying and describing the function of common physical components of computing systems (hardware).** *A computing system is composed of hardware and software. Hardware consists of physical components.Students should be able to identify and describe the function of external hardware, such as desktop computers, laptop computers, tablet devices, monitors, keyboards, mice, and printers.* **Practice(s):** Communicating About Computing: 7.2	Computing Systems	Communicating
1A-CS-03	K-2	**Describe basic hardware and software problems using accurate terminology.** *Problems with computing systems have different causes. Students at this level do not need to understand those causes, but they should be able to 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.). Ideally, students would be able to use simple troubleshooting strategies, including turning a device off and on to reboot it, closing and reopening an app, turning on speakers, or plugging in headphones. These are, however, not specified in the standard, because these problems may not occur.* **Practice(s):** Testing and Refining Computational Artifacts, Communicating About Computing: 6.2, 7.2	Computing Systems	Testing, Communicating
1A-DA-05	K-2	**Store, copy, search, retrieve, modify, and delete information using a computing device and define the information stored as data.** *All information stored and processed by a computing device is referred to as data. Data can be images, text documents, audio files, software programs or apps, video files, etc. As students use software to complete tasks on a computing device, they will be manipulating data.* **Practice(s):** Developing and Using Abstractions: 4.2	Data & Analysis	Abstraction

1A-DA-06	K-2	**Collect and present the same data in various visual formats.** *The collection and use of data about the world around them is a routine part of life and influences how people live. Students could collect data on the weather, such as sunny days versus rainy days, the temperature at the beginning of the school day and end of the school day, or the inches of rain over the course of a storm. Students could count the number of pieces of each color of candy in a bag of candy, such as Skittles or M&Ms. Students could create surveys of things that interest them, such as favorite foods, pets, or TV shows, and collect answers to their surveys from their peers and others. The data collected could then be organized into two or more visualizations, such as a bar graph, pie chart, or pictograph.* **Practice(s):** Communicating About Computing, Developing and Using Abstractions: 7.1, 4.4	Data & Analysis	Communicating, Abstraction
1A-DA-07	K-2	**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. Students could analyze a graph or pie chart of the colors in a bag of candy or the averages for colors in multiple bags of candy, identify the patterns for which colors are most and least represented, and then make a prediction as to which colors will have most and least in a new bag of candy. Students could analyze graphs of temperatures taken at the beginning of the school day and end of the school day, identify the patterns of when temperatures rise and fall, and predict if they think the temperature will rise or fall at a particular time of the day, based on the pattern observed.* **Practice(s):** Developing and Using Abstractions: 4.1	Data & Analysis	Abstraction
1A-IC-16	K-2	**Compare how people live and work before and after the implementation or adoption of new computing technology.** *Computing technology has positively and negatively changed the way people live and work. 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.* **Practice(s):** Communicating About Computing: 7	Impacts of Computing	Communicating

1A-IC-17	K-2	**Work respectfully and responsibly with others online.** *Online communication facilitates positive interactions, such as sharing ideas with many people, but the public and anonymous nature of online communication also allows intimidating and inappropriate behavior in the form of cyber bullying. Students could share their work on blogs or in other collaborative spaces online, taking care to avoid sharing information that is inappropriate or that could personally identify them to others. Students could provide feedback to others on their work in a kind and respectful manner and could tell an adult if others are sharing things they should not share or are treating others in an unkind or disrespectful manner on online collaborative spaces.* **Practice(s):** Collaborating Around Computing: 2.1	Impacts of Computing	Collaborating
1A-IC-18	K-2	**Keep login information private, and log off of devices appropriately.** *People use computing technology in ways that can help or hurt themselves or others. Harmful behaviors, such as sharing private information and leaving public devices logged in should be recognized and avoided.* **Practice(s):** Communicating About Computing: 7.3	Impacts of Computing	Communicating
1A-NI-04	K-2	**Explain what passwords are and why we use them, and use strong passwords to protect devices and information from unauthorized access.** *Learning to protect one's device or information from unwanted use by others is an essential first step in learning about cybersecurity. Students are not required to use multiple strong passwords. They should appropriately use and protect the passwords they are required to use.* **Practice(s):** Communicating About Computing: 7.3	Networks & the Internet	Communicating
1B-AP-08	3-5	**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 most appropriate for a specific situation. Students should be able to look at different ways to solve the same task and decide which would be the best solution. For example, students could use a map and plan multiple algorithms to get from one point to another. They could look at routes suggested by mapping software and change the route to something that would be better, based on which route is shortest or fastest or would avoid a problem. Students might compare algorithms that describe how to get ready for school. Another example might be to write different algorithms to draw a regular polygon and determine which algorithm would be the easiest to modify or repurpose to draw a different polygon.* **Practice(s):** Testing and Refining Computational Artifacts, Recognizing and Defining Computational Problems: 6.3, 3.3	Algorithms & Programming	Testing, Computational Problems

1B-AP-09	3-5	**Create programs that use variables to store and modify data.** *Variables are used to store and modify data. At this level, understanding how to use variables is sufficient. For example, students may use mathematical operations to add to the score of a game or subtract from the number of lives available in a game. The use of a variable as a countdown timer is another example.* **Practice(s):** Creating Computational Artifacts: 5.2	Algorithms & Programming	Creating
1B-AP-10	3-5	**Create programs that include sequences, 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. For example, students could write a program to explain the water cycle and when a specific component is clicked (event), the program would show information about that part of the water cycle. Conditionals allow for the execution of a portion of code in a program when a certain condition is true. For example, students could write a math game that asks multiplication fact questions and then uses a conditional to check whether or not the answer that was entered is correct. Loops allow for the repetition of a sequence of code multiple times. For example, in a program that produces an animation about a famous historical character, students could use a loop to have the character walk across the screen as they introduce themselves.* **Practice(s):** Creating Computational Artifacts: 5.2	Algorithms & Programming	Creating
1B-AP-11	3-5	**Decompose (break down) problems into smaller, manageable subproblems to facilitate the program development process.** *Decomposition is the act of breaking down tasks into simpler tasks. For example, students could create an animation by separating a story into different scenes. For each scene, they would select a background, place characters, and program actions.* **Practice(s):** Recognizing and Defining Computational Problems: 3.2	Algorithms & Programming	Computational Problems
1B-AP-12	3-5	**Modify, remix, or incorporate portions of an existing program into one's own work, 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. For example, students could modify prewritten code from a single-player game to create a two-player game with slightly different rules, remix and add another scene to an animated story, use code to make a ball bounce from another program in a new basketball game, or modify an image created by another student.* **Practice(s):** Creating Computational Artifacts: 5.3	Algorithms & Programming	Creating

1B-AP-13	3-5	**Use an iterative process to plan the development of a program by including others' perspectives and considering user preferences.** *Planning is an important part of the iterative process of program development. Students outline key features, time and resource constraints, and user expectations. Students should document the plan as, for example, a storyboard, flowchart, pseudocode, or story map.* **Practice(s):** Fostering an Inclusive Computing Culture, Creating Computational Artifacts: 1.1, 5.1	Algorithms & Programming	Inclusion, Creating
1B-AP-14	3-5	**Observe intellectual property rights and give appropriate attribution when creating or remixing programs.** *Intellectual property rights can vary by country but copyright laws give the creator of a work a set of rights that prevents others from copying the work and using it in ways that they may not like. Students should identify instances of remixing, when ideas are borrowed and iterated upon, and credit the original creator. Students should also consider common licenses that place limitations or restrictions on the use of computational artifacts, such as images and music downloaded from the Internet. At this stage, attribution should be written in the format required by the teacher and should always be included on any programs shared online.* **Practice(s):** Creating Computational Artifacts, Communicating About Computing: 5.2, 7.3	Algorithms & Programming	Creating, Communicating
1B-AP-15	3-5	**Test and debug (identify and fix errors) a program or algorithm to ensure it runs as intended.** *As students develop programs they should continuously test those programs to see that they do what was expected and fix (debug), any errors. Students should also be able to successfully debug simple errors in programs created by others.* **Practice(s):** Testing and Refining Computational Artifacts: 6.1, 6.2	Algorithms & Programming	Testing
1B-AP-16	3-5	**Take on varying roles, with teacher guidance, when collaborating with peers during the design, implementation, and review stages of program development.** *Collaborative computing is the process of performing a computational task by working in pairs or on teams. Because it involves asking for the contributions and feedback of others, effective collaboration can lead to better outcomes than working independently. Students should take turns in different roles during program development, such as note taker, facilitator, program tester, or "driver" of the computer.* **Practice(s):** Collaborating Around Computing: 2.2	Algorithms & Programming	Collaborating
		Describe choices made during program development using code comments, presentations, and demonstrations.		

1B-AP-17	3-5	*People communicate about their code to help others understand and use their programs. Another purpose of communicating one's design choices is to show an understanding of one's work. These explanations could manifest themselves as in-line code comments for collaborators and assessors, or as part of a summative presentation, such as a code walk-through or coding journal.* **Practice(s):** Communicating About Computing: 7.2	Algorithms & Programming	Communicating
1B-CS-01	3-5	**Describe how internal and external parts of computing devices function to form a system.** *Computing devices often depend on other devices or components. For example, a robot depends on a physically attached light sensor to detect changes in brightness, whereas the light sensor depends on the robot for power. Keyboard input or a mouse click could cause an action to happen or information to be displayed on a screen; this could only happen because the computer has a processor to evaluate what is happening externally and produce corresponding responses. Students should describe how devices and components interact using correct terminology.* **Practice(s):** Communicating About Computing: 7.2	Computing Systems	Communicating
1B-CS-02	3-5	**Model how computer hardware and software work together as a system to accomplish tasks.** *In order for a person to accomplish tasks with a computer, both hardware and software are needed. At this stage, a model should only include the basic elements of a computer system, such as input, output, processor, sensors, and storage. 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.* **Practice(s):** Developing and Using Abstractions: 4.4	Computing Systems	Abstraction
1B-CS-03	3-5	**Determine potential solutions to solve simple hardware and software problems using common troubleshooting strategies.** *Although computing systems may vary, common troubleshooting strategies can be used on all of them. Students should be able to identify solutions to problems such as the device not responding, no power, no network, app crashing, no sound, or password entry not working. Should errors occur at school, the goal would be that students would use various strategies, such as rebooting the device, checking for power, checking network availability, closing and reopening an app, making sure speakers are turned on or headphones are plugged in, and making sure that the caps lock key is not on, to solve these problems, when possible.* **Practice(s):** Testing and Refining Computational Artifacts: 6.2	Computing Systems	Testing
		Organize and present collected data visually to highlight relationships and support a claim.		

1B-DA-06	3-5	*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, a data set of sports teams could be sorted by wins, points scored, or points allowed, and a data set of weather information could be sorted by high temperatures, low temperatures, or precipitation.* **Practice(s):** Communicating About Computing: 7.1	Data & Analysis	Communicating
1B-DA-07	3-5	**Use data to highlight or propose cause-and-effect relationships, predict outcomes, or communicate an idea.** *The accuracy of data analysis is related to how realistically data is represented. Inferences or predictions based on data are less likely to be accurate if the data is not sufficient or if the data is incorrect in some way. Students should be able to refer to data when communicating an idea. For example, in order to explore the relationship between speed, time, and distance, students could operate a robot at uniform speed, and at increasing time intervals to predict how far the robot travels at that speed. In order to make an accurate prediction, one or two attempts of differing times would not be enough. The robot may also collect temperature data from a sensor, but that data would not be relevant for the task. Students must also make accurate measurements of the distance the robot travels in order to develop a valid prediction. Students could record the temperature at noon each day as a basis 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 or at different times of the day, the data would be incomplete and the 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 want to use data provided by a governmental weather agency.* **Practice(s):** Communicating About Computing: 7.1	Data & Analysis	Communicating

1B-IC-18	3-5	**Discuss computing technologies that have changed the world, and express how those technologies influence, and are influenced by, cultural practices.** *New computing technology is 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, should discuss topics that relate to the history of technology and the changes in the world due to technology. Topics could be based on current news content, such as robotics, wireless Internet, mobile computing devices, GPS systems, wearable computing, or how social media has influenced social and political changes.* **Practice(s):** Recognizing and Defining Computational Problems: 3.1	Impacts of Computing	Computational Problems
1B-IC-19	3-5	**Brainstorm 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 are driven by people's needs and wants and can affect groups differently. Anticipating the needs and wants of diverse end users requires students to purposefully consider potential perspectives of users with different backgrounds, ability levels, points of view, and disabilities. For example, students may consider using both speech and text when they wish 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.* **Practice(s):** Fostering an Inclusive Computing Culture: 1.2	Impacts of Computing	Inclusion
1B-IC-20	3-5	**Seek diverse perspectives for the purpose of improving computational artifacts.** *Computing provides the possibility for collaboration and sharing of ideas and allows the benefit of diverse perspectives. For example, students could seek feedback from other groups in their class or students at another grade level. Or, with guidance from their teacher, they could use video conferencing tools or other online collaborative spaces, such as blogs, wikis, forums, or website comments, to gather feedback from individuals and groups about programming projects.* **Practice(s):** Fostering an Inclusive Computing Culture: 1.1	Impacts of Computing	Inclusion

1B-IC-21	3-5	**Use public domain or creative commons media, and refrain from copying or using material created by others without permission.** *Ethical complications arise from the opportunities provided by computing. The ease of sending and receiving copies of media on the Internet, such as video, photos, and music, creates the opportunity for unauthorized use, such as online piracy, and disregard of copyrights. Students should consider the licenses on computational artifacts that they wish to use. For example, the license on a downloaded image or audio file may have restrictions that prohibit modification, require attribution, or prohibit use entirely.* **Practice(s):** Communicating About Computing: 7.3	Impacts of Computing	Communicating
1B-NI-04	3-5	**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 should 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 which has them act it out in some way.* **Practice(s):** Developing and Using Abstractions: 4.4	Networks & the Internet	Abstraction
1B-NI-05	3-5	**Discuss real-world cybersecurity problems and how personal information can be protected.** *Just as we protect our personal property offline, we also need to protect our devices and the information stored on them. Information can be protected using various security measures. These measures can be physical and/or digital. Students could discuss or use a journaling or blogging activity to explain, orally or in writing, about 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 install and keep anti-virus software updated to protect data and systems.* **Practice(s):** Recognizing and Defining Computational Problems: 3.1	Networks & the Internet	Computational Problems

2-AP-10	6-8	**Use flowcharts and/or pseudocode to address complex problems as algorithms.** *Complex problems are problems that would be difficult for students to solve computationally. Students should use pseudocode and/or flowcharts to organize and sequence an algorithm that addresses a complex problem, even though they may not actually program the solutions. For example, students might express an algorithm that produces a recommendation for purchasing sneakers based on inputs such as size, colors, brand, comfort, and cost. Testing the algorithm with a wide range of inputs and users allows students to refine their recommendation algorithm and to identify other inputs they may have initially excluded.* **Practice(s):** Developing and Using Abstractions: 4.4, 4.1	Algorithms & Programming	Abstraction
2-AP-11	6-8	**Create clearly named variables that represent different data types and perform operations on their values.** *A variable is like a container with a name, in which the contents may change, but the name (identifier) does not. When planning and developing programs, students should decide when and how to declare and name new variables. Students should use naming conventions to improve program readability. Examples of operations include adding points to the score, combining user input with words to make a sentence, changing the size of a picture, or adding a name to a list of people.* **Practice(s):** Creating Computational Artifacts: 5.1, 5.2	Algorithms & Programming	Creating
2-AP-12	6-8	**Design and iteratively develop programs that combine control structures, including nested loops and compound conditionals.** *Control structures can be combined in many ways. Nested loops are loops placed within loops. Compound conditionals combine two or more conditions in a logical relationship (e.g., using AND, OR, and NOT), and nesting conditionals within one another allows the result of one conditional to lead to another. For example, when programming an interactive story, students could use a compound conditional within a loop to unlock a door only if a character has a key AND is touching the door.* **Practice(s):** Creating Computational Artifacts: 5.1, 5.2	Algorithms & Programming	Creating

2-AP-13	6-8	**Decompose problems and subproblems into parts to facilitate the design, implementation, and review of programs.** *Students should break down problems into subproblems, which can be further broken down to smaller parts. Decomposition facilitates aspects of 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. For example, animations can be decomposed into multiple scenes, which can be developed independently.* **Practice(s):** Recognizing and Defining Computational Problems: 3.2	Algorithms & Programming	Computational Problems
2-AP-14	6-8	**Create procedures with parameters to organize code and make it easier to reuse.** *Students should create procedures and/or functions that are used multiple times within a program to repeat groups of instructions. These procedures can be generalized by defining parameters that create different outputs for a wide range of inputs. For example, a procedure to draw a circle involves many instructions, but all of them can be invoked with one instruction, such as "drawCircle." By adding a radius parameter, the user can easily draw circles of different sizes.* **Practice(s):** Developing and Using Abstractions: 4.1, 4.3	Algorithms & Programming	Abstraction
2-AP-15	6-8	**Seek and incorporate feedback from team members and users to refine a solution that meets user needs.** *Development teams that employ user-centered design create solutions (e.g., programs and devices) that can have a large societal impact, such as an app that allows people with speech difficulties to translate hard-to-understand pronunciation into understandable language. Students should begin to seek diverse perspectives throughout the design process to improve their computational artifacts. Considerations of the end-user may include usability, accessibility, age-appropriate content, respectful language, user perspective, pronoun use, color contrast, and ease of use.* **Practice(s):** Collaborating Around Computing, Fostering an Inclusive Computing Culture: 2.3, 1.1	Algorithms & Programming	Collaborating, Inclusion
2-AP-16	6-8	**Incorporate existing code, media, and libraries into original programs, and give attribution.** *Building on the work of others enables students to produce more interesting and powerful creations. Students should use portions of code, algorithms, and/or digital media in their own programs and websites. At this level, they may also import libraries and connect to web application program interfaces (APIs). For example, when creating a side-scrolling game, students may incorporate portions of code that create a realistic jump movement from another person's game, and they may also import Creative Commons-licensed images to use in the background. Students should give attribution to the original creators to acknowledge their contributions.* **Practice(s):** Developing and Using Abstractions, Creating Computational Artifacts, Communicating About Computing: 4.2, 5.2, 7.3	Algorithms & Programming	Abstraction, Creating, Communicating

2-AP-17	6-8	**Systematically test and refine programs using a range of test cases.** *Use cases and test cases are created and analyzed to better meet the needs of users and to evaluate whether programs function as intended. At this level, testing should become a deliberate process that is more iterative, systematic, and proactive than at lower levels. Students should begin to test programs by considering potential errors, such as what will happen if a user enters invalid input (e.g., negative numbers and 0 instead of positive numbers).* **Practice(s):** Testing and Refining Computational Artifacts: 6.1	Algorithms & Programming	Testing
2-AP-18	6-8	**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 should assume pre-defined roles within their teams and manage the project workflow using structured timelines. With teacher guidance, they will begin to create collective goals, expectations, and equitable workloads. For example, students may divide the design stage of a game into planning the storyboard, flowchart, and different parts of the game mechanics. They can then distribute tasks and roles among members of the team and assign deadlines.* **Practice(s):** Collaborating Around Computing: 2.2	Algorithms & Programming	Collaborating
2-AP-19	6-8	**Document programs in order to make them easier to follow, test, and debug.** *Documentation allows creators and others to more easily use and understand a program. Students should provide documentation for end users that explains their artifacts and how they function. For example, students could provide a project overview and clear user instructions. They should also incorporate comments in their product and communicate their process using design documents, flowcharts, and presentations.* **Practice(s):** Communicating About Computing: 7.2	Algorithms & Programming	Communicating
2-CS-01	6-8	**Recommend improvements to the design of computing devices, based on an analysis of how users interact with the devices.** *The study of human–computer interaction (HCI) can improve the design of devices, including both hardware and software. Students should make recommendations for existing devices (e.g., a laptop, phone, or tablet) or design their own components or interface (e.g., create their own controllers). Teachers can guide students to consider usability through several lenses, including accessibility, ergonomics, and learnability. For example, assistive devices provide capabilities such as scanning written information and converting it to speech.* **Practice(s):** Recognizing and Defining Computational Problems: 3.3	Computing Systems	Computational Problems

2-CS-02	6-8	**Design projects that combine hardware and software components to collect and exchange data.** *Collecting and exchanging data involves input, output, storage, and processing. When possible, students should select the hardware and software components for their project designs by considering factors such as functionality, cost, size, speed, accessibility, and aesthetics. For example, components for a mobile app could include accelerometer, GPS, and speech recognition. The choice of a device that connects wirelessly through a Bluetooth connection versus a physical USB connection involves a tradeoff between mobility and the need for an additional power source for the wireless device.* **Practice(s):** Creating Computational Artifacts: 5.1	Computing Systems	Creating
2-CS-03	6-8	**Systematically identify and fix problems with computing devices and their components.** *Since a computing device may interact with interconnected devices within a system, problems may not be due to the specific computing device itself but to devices connected to it. Just as pilots use checklists to troubleshoot problems with aircraft systems, students should use a similar, structured process to troubleshoot problems with computing systems and ensure that potential solutions are not overlooked. Examples of troubleshooting strategies include following a troubleshooting flow diagram, making changes to software to see if hardware will work, checking connections and settings, and swapping in working components.* **Practice(s):** Testing and Refining Computational Artifacts: 6.2	Computing Systems	Testing
2-DA-07	6-8	**Represent data using multiple encoding schemes.** *Data representations occur at multiple levels of abstraction, from the physical storage of bits to the arrangement of information into organized formats (e.g., tables). Students should represent the same data in multiple ways. For example, students could represent the same color using binary, RGB values, hex codes (low-level representations), as well as forms understandable by people, including words, symbols, and digital displays of the color (high-level representations).* **Practice(s):** Developing and Using Abstractions: 4	Data & Analysis	Abstraction
2-DA-08	6-8	**Collect data using computational tools and transform the data to make it more useful and reliable.** *As students continue to build on their ability to organize and present data visually to support a claim, they will need to understand when and how to transform data for this purpose. Students should transform data to remove errors, highlight or expose relationships, and/or make it easier for computers to process. The cleaning of data is an important transformation for ensuring consistent format and reducing noise and errors (e.g., removing irrelevant responses in a survey). An example of a transformation that highlights a relationship is representing males and females as percentages of a whole instead of as individual counts.* **Practice(s):** Testing and Refining Computational Artifacts: 6.3	Data & Analysis	Testing

2-DA-09	6-8	**Refine computational models based on the data they have generated.** *A model may be a programmed simulation of events or a representation of how various data is related. In order to refine a model, students need to consider which data points are relevant, how data points relate to each other, and if the data is accurate. For example, students may make a prediction about how far a ball will travel based on a table of data related to the height and angle of a track. The students could then test and refine their model by comparing predicted versus actual results and considering whether other factors are relevant (e.g., size and mass of the ball). Additionally, students could refine game mechanics based on test outcomes in order to make the game more balanced or fair.* **Practice(s):** Creating Computational Artifacts, Developing and Using Abstractions: 5.3, 4.4	Data & Analysis	Creating, Abstraction
2-IC-20	6-8	**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 should consider current events related to broad ideas, including privacy, communication, and automation. For example, driverless cars can increase convenience and reduce accidents, but they are also susceptible to hacking. The emerging industry will reduce the number of taxi and shared-ride drivers, but will create more software engineering and cybersecurity jobs.* **Practice(s):** Communicating About Computing: 7.2	Impacts of Computing	Communicating
2-IC-21	6-8	**Discuss issues of bias and accessibility in the design of existing technologies.** *Students should test and discuss the usability of various technology tools (e.g., apps, games, and devices) with the teacher's guidance. For example, facial recognition software that works better for lighter skin tones was likely developed with a homogeneous testing group and could be improved by sampling a more diverse population. When discussing accessibility, students may notice that allowing a user to change font sizes and colors will not only make an interface usable for people with low vision but also benefits users in various situations, such as in bright daylight or a dark room.* **Practice(s):** Fostering an Inclusive Computing Culture: 1.2	Impacts of Computing	Inclusion

2-IC-22	6-8	**Collaborate with many contributors through strategies such as crowdsourcing or surveys when creating a computational artifact.** *Crowdsourcing is gathering services, ideas, or content from a large group of people, especially from the online community. It can be done at the local level (e.g., classroom or school) or global level (e.g., age-appropriate online communities, like Scratch and Minecraft). For example, a group of students could combine animations to create a digital community mosaic. They could also solicit feedback from many people though use of online communities and electronic surveys.* **Practice(s):** Collaborating Around Computing, Creating Computational Artifacts: 2.4, 5.2	Impacts of Computing	Collaborating, Creating
2-IC-23	6-8	**Describe tradeoffs between allowing information to be public and keeping information private and secure.** *Sharing information online can help establish, maintain, and strengthen connections between people. For example, it allows artists and designers to display their talents and reach a broad audience. However, security attacks often start with personal information that is publicly available online. Social engineering is based on tricking people into revealing sensitive information and can be thwarted by being wary of attacks, such as phishing and spoofing.* **Practice(s):** Communicating About Computing: 7.2	Impacts of Computing	Communicating
2-NI-04	6-8	**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 and the Internet, as well as how to handle errors in transmission. Students should model how data is sent using protocols to choose the fastest path, to deal with missing information, and to deliver sensitive data securely. For example, students could devise a plan for resending lost information or for interpreting a picture that has missing pieces. The priority at this grade level is understanding the purpose of protocols and how they enable secure and errorless communication. Knowledge of the details of how specific protocols work is not expected.* **Practice(s):** Developing and Using Abstractions: 4.4	Networks & the Internet	Abstraction
2-NI-05	6-8	**Explain how physical and digital security measures protect electronic information.** *Information that is stored online is vulnerable to unwanted access. Examples of physical security measures to protect data include keeping passwords hidden, locking doors, making backup copies on external storage devices, and erasing a storage device before it is reused. Examples of digital security measures include secure router admin passwords, firewalls that limit access to private networks, and the use of a protocol such as HTTPS to ensure secure data transmission.* **Practice(s):** Communicating About Computing: 7.2	Networks & the Internet	Communicating

2-NI-06	6-8	**Apply multiple methods of encryption to model the secure transmission of information.** *Encryption can be as simple as letter substitution or as complicated as modern methods used to secure networks and the Internet. Students should encode and decode messages using a variety of encryption methods, and they should understand the different levels of complexity used to hide or secure information. For example, students could secure messages using methods such as Caesar cyphers or steganography (i.e., hiding messages inside a picture or other data). They can also model more complicated methods, such as public key encryption, through unplugged activities.* **Practice(s):** Developing and Using Abstractions: 4.4	Networks & the Internet	Abstraction

KIPR Curriculum

Module
Module 2- Creating Algorithms (Unplugged) Module 3- Unplugged programming
Module 2- Creating Algorithms (Unplugged) Module 3- Unplugged programming Module 8- Writing your First Program Module 9- Moving Your Robot Module 11- Using a Servo Module 12- Using Multiple Servos
Module 8- Writing your First Program Module 9- Moving Your Robot Module 11- Using a Servo Module 12- Using Multiple Servos
Module 2- Creating Algorithms (Unplugged) Module 3- Unplugged programming Module 8- Writing your First Program Module 9- Moving Your Robot Module 11- Using a Servo Module 12- Using Multiple Servos

Module 2- Creating Algorithms (Unplugged) Module 3- Unplugged programming Module 8- Writing your First Program Module 9- Moving Your Robot Module 11- Using a Servo Module 12- Using Multiple Servos
Module 5- Navigating the Digital World Teamwork and Project Management Strategies
Module 8- Writing your First Program Module 9- Moving Your Robot Module 11- Using a Servo Module 12- Using Multiple Servos
Module 8- Writing your First Program Module 9- Moving Your Robot Module 11- Using a Servo Module 12- Using Multiple Servos
Module 5- Navigating the Digital World

Module 4- Computer Communication Module 6- Introduction to Robots
Module 6- Introduction to Robots Module 7- Introduction to Programming Languages Module 8- Writing Your First Program
Module 8- Writing Your First Program

Activity M4 Activity M41 Activity M68 Activity M83 Activity M84 Activity M85 Activity M93 Activity M94 Activity M95 Activity M96
Activity M4 Activity M41 Activity M68 Activity M83 Activity M84 Activity M85 Activity M93 Activity M94 Activity M95 Activity M96
Module 5- Navigating the Digital World

Module 5- Navigating the Digital World Teamwork and Project Management Strategies
Module 5- Navigating the Digital World Module 8- Writing Your First Program
Module 5- Navigating the Digital World
Module 2- Creating Algorithms (Unplugged) Module 9- Moving Your Robot Module 10- Introduction to Engineering Module 11- Using a Servo Module 12- Using Multiple Servos Module 13- Introduction to Variable Module 14- Digital Sensors Module 15- Analog Sensors Module 16- Motor Position Counter Writing Functions for Loops Creating functions using Void

Module 11- Using a Servo Module 12- Using Multiple Servos Module 13- Introduction to Variable Module 14- Digital Sensors Module 15- Analog Sensors Module 16- Motor Position Counter Writing Functions for Loops Creating functions using Void
Module 9- Moving Your Robot Module 10- Introduction to Engineering Module 11- Using a Servo Module 12- Using Multiple Servos Module 13- Introduction to Variable Module 14- Digital Sensors Module 15- Analog Sensors Module 16- Motor Position Counter Writing Functions for Loops Creating functions using Void
Module 2- Creating Algorithms (Unplugged) Module 9- Moving Your Robot Module 10- Introduction to Engineering Module 11- Using a Servo Module 12- Using Multiple Servos Module 13- Introduction to Variable Module 14- Digital Sensors Module 15- Analog Sensors
Module 2- Creating Algorithms (Unplugged) Module 9- Moving Your Robot Module 10- Introduction to Engineering Module 11- Using a Servo Module 12- Using Multiple Servos Module 13- Introduction to Variable Module 14- Digital Sensors Module 15- Analog Sensors Module 16- Motor Position Counter Writing Functions

Module 2- Creating Algorithms (Unplugged) Module 9- Moving Your Robot Module 10- Introduction to Engineering Module 11- Using a Servo Module 12- Using Multiple Servos Module 13- Introduction to Variable Module 14- Digital Sensors Module 15- Analog Sensors Module 16- Motor Position Counter
Module 5- Navigating the Digital World Module 8- Writing Your First Program Teamwork and Project Management Strategies
Module 8- Writing Your First Program Module 9- Moving Your Robot Module 10- Introduction to Engineering Module 11- Using a Servo Module 12- Using Multiple Servos Module 13- Introduction to Variable Module 14- Digital Sensors Module 15- Analog Sensors
Teamwork and Project Management Strategies Module 8- Writing Your First Program Module 9- Moving Your Robot Module 10- Introduction to Engineering Module 11- Using a Servo Module 12- Using Multiple Servos Module 13- Introduction to Variable Module 14- Digital Sensors Module 15- Analog Sensors Module 16- Motor Position Counter Writing Functions for Loops
Module 8- Writing Your First Program Module 9- Moving Your Robot Module 10- Introduction to Engineering Module 11- Using a Servo

Module 12- Using Multiple Servos Module 13- Introduction to Variable Module 14- Digital Sensors Module 15- Analog Sensors Module 16- Motor Position Counter Writing Functions for Loops
Module 4- Computer Communication Module 6- Introduction to Robots Module 14- Digital Sensors Module 15- Analog Sensors
Module 4- Computer Communication Module 6- Introduction to Robots Module 14- Digital Sensors Module 15- Analog Sensors
Module 4- Computer Communication Module 6- Introduction to Robots Module 8- Writing Your First Program Module 9- Moving Your Robot Module 10- Introduction to Engineering Module 11- Using a Servo Module 12- Using Multiple Servos Module 13- Introduction to Variable Module 14- Digital Sensors Module 15- Analog Sensors Module 16- Motor Position Counter Writing Functions for Loops Creating functions using Void
Activity M4 Activity M41 Activity M68

Activity M68 Activity M83 Activity M84 Activity M85 Activity M93 Activity M94 Activity M95 Activity M96
Activity M4 Activity M41 Activity M68 Activity M83 Activity M84 Activity M85 Activity M93 Activity M94 Activity M95 Activity M96

Module 5- Navigating the Digital World
Module 5- Navigating the Digital World
Teamwork and Project Management Strategies Module 9- Moving Your Robot Module 11- Using a Servo Module 12- Using Multiple Servos Module 13- Introduction to Variable Module 14- Digital Sensors Module 15- Analog Sensors Module 16- Motor Position Counter Writing Functions for Loops

Module 5- Navigating the Digital World
Module 4- Computer Communication
Module 5- Navigating the Digital World

Module 8- Writing Your First Program Module 9- Moving Your Robot Module 10- Introduction to Engineering Module 11- Using a Servo Module 12- Using Multiple Servos Module 13- Introduction to Variable Module 14- Digital Sensors Module 15- Analog Sensors Module 16- Motor Position Counter Writing Functions for Loops Creating functions using Void
Module 13- Introduction to Variable Module 14- Digital Sensors Module 15- Analog Sensors Module 16- Motor Position Counter Writing Functions for Loops Creating functions using Void
Module 13- Introduction to Variable Module 14- Digital Sensors Module 15- Analog Sensors Module 16- Motor Position Counter Writing Functions for Loops Creating functions using Void Using the Camera Advanced Camera Code

Module 8- Writing Your First Program Module 9- Moving Your Robot Module 10- Introduction to Engineering Module 11- Using a Servo Module 12- Using Multiple Servos Module 13- Introduction to Variable Module 14- Digital Sensors Module 15- Analog Sensors Module 16- Motor Position Counter Writing Functions for Loops Creating functions using Void
Writing Functions What is a Library?
Teamwork and Project Management Strategies Module 5- Navigating the Digital World GitHub
Module 5- Navigating the Digital World Writing Functions What is a Library?

Module 9- Moving Your Robot Module 10- Introduction to Engineering Module 11- Using a Servo Module 12- Using Multiple Servos Module 13- Introduction to Variable Module 14- Digital Sensors Module 15- Analog Sensors Module 16- Motor Position Counter Writing Functions for Loops Creating functions using Void
Teamwork and Project Management Strategies Module 5- Navigating a Digital World Github
Module 8- Writing Your First Program Module 9- Moving Your Robot Module 10- Introduction to Engineering Module 11- Using a Servo Module 12- Using Multiple Servos Module 13- Introduction to Variable Module 14- Digital Sensors Module 15- Analog Sensors Module 16- Motor Position Counter Writing Functions
Module 5- Navigating the Digital World

Module 14- Digital Sensors Module 15- Analog Sensors Module 16- Motor Position Counter
Module 8- Writing Your First Program Module 9- Moving Your Robot Module 10- Introduction to Engineering Module 11- Using a Servo Module 12- Using Multiple Servos Module 13- Introduction to Variable Module 14- Digital Sensors Module 15- Analog Sensors Module 16- Motor Position Counter
Activity M95 Using the Camera Advanced Camera Code
Module 14- Digital Sensors Module 15- Analog Sensors Module 16- Motor Position Counter Using the Camera Advanced Camera Code

Module 11- Using a Servo Module 12- Using Multiple Servos Module 13- Introduction to Variable Module 14- Digital Sensors Module 15- Analog Sensors Module 16- Motor Position Counter
Module 5- Navigating the Digital World
Module 5- Navigating the Digital World

Teamwork and Project Management Strategies Github
Github Module 5- Navigating the Digital World
Module 4- Computer Communication
Module 4- Computer Communication

Module 4- Computer Communication