Standard Identifier	Standard	Clarification	Best Aligned Resource	Resource #2	Resource #3
2-CS-01	**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.	Semester A, Unit 3, Lesson 5: *Designing Devices*	Semester A, Unit 3, Lesson 4: *Evaluating Devices*	
2-CS-02	**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.	Semester A, Unit 3, Project: *Design a Probe*	Semester A, Unit 3, Lesson 5: *Designing Devices*	
2-CS-03	**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.	Semester A, Unit 4 *Troubleshooting,* Project: Be a Troubleshooter	Semester A, Unit 4 Troubleshooting, Lesson 1: *Basic Troubleshooting*	Semester A, Unit 4 Troubleshooting, Lesson 3:What Is Wrong with Your Printer?
2-NI-04	**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.	Semester A, Unit 5 *Networks and the Internet,* Lesson 2: *What Is a Protocol?*		
2-NI-05	**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.	Semester A, Unit 6 *Cybersecurity* , Lesson 3: *How Do You Protect Your Devices?*	Semester A, Unit 6 *Cybersecurity* , Lesson 1: *Protect Your Personal Information*	
2-NI-06	**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.	Semester A, Unit 6 *Cybersecurity* , Lesson 3: *How Do You Protect Your Devices?*		
2-DA-07	**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).	Semester A, Unit 2 *Hardware & Software* , Lesson 1: Learning How Computers Talk		
2-DA-08	**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.	Semester B, Unit 1 *Data & Analysis* , Project: *Your Data Presentation*	Semester B, Unit 1 *Data & Analysis* , Lesson 3: Organizing Your Data	
2-DA-09	**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.	Semester B, Unit 4 *Program Design* , Lesson 3: Models and Simulations		
2-AP-10	**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.	Semester B, Unit 2 Algorithms and Programming, Lesson 1: Planning Your Program	Semester B, Unit 2 Algorithms and Programming, Project: Block Programming	
2-AP-11	**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.	Semester B, Unit 2 Algorithms and Programming, Lesson 3: Variables and Decisions	Semester B, Unit 2 Algorithms and Programming, Project: Block Programming	
2-AP-12	**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.	Semester B, Unit 2 Algorithms and Programming, Project: Block Programming	Semester B, Unit 2 Algorithms and Programming, Lesson 4: Nested Decisions	Semester B, Unit 2 Algorithms and Programming, Lesson 5: Loops that Solve Problems
2-AP-13	**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.	Semester B, Unit 2 *Algorithms and Programming* , Lesson 1: Planning Your Program		
2-AP-14	**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.	Semester B, Unit 4 *Program Design* , Lesson 2: *Functions with Parameters*	Semester B, Unit 4 *Program Design* , Lesson 1: *Functions*	
2-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 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.	Semester B, Unit 4 *Program Design* , Unit Project: Design for a Client	Semester B, Unit 4 *Program Design* , Lesson 2: *Designing for Accessibility*	
2-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 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.	Semester B, Unit 5 Cultural Impact of Computing, Lesson 4: Giving Credit Where Due		
2-AP-17	**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).	Semester B, Unit 3 COMPUTATIONAL THINKING AND PROBLEM SOLVING, Unit Project		

2-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 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.	Semester B, Unit 3 COMPUTATIONAL THINKING AND PROBLEM SOLVING, Unit Project	
2-AP-19	**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.	Semester B, Unit 2 Algorithms and Programming, Project: Block Programming	Semester B, Unit 4 *Program Design* , Lesson 5: Documentation
2-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 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.	Semester B, Unit 5 Cultural Impact of Computing, Unit Project	
2-IC-21	**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.	Semester B, Unit 5 Cultural Impact of Computing, Lesson 2: Bias and Accessibility	
2-IC-22	**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.	Semester B, Unit 5 Cultural Impact of Computing, Lesson 5: Global Communication, Power, and Access	
2-IC-23	**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.	Semester A, Unit 6 *Cybersecurity* , Lesson 2: Outsmarting Hackers	