Computer Science Essential Concepts
And
Subconcepts

Concept: Algorithms and Programming

Kindergarten - Highschool

Computer Science Essential Concepts and Subconcepts

The Arizona Computer Science Standards for grades kindergarten through twelve are organized into five Essential Concepts:

- **Computing Systems:** This involves the interaction that people have with a wide variety of computing devices that collect, store, analyze, and act upon information in ways that can affect human capabilities both positively and negatively. The physical components (hardware) and instructions (software) that make up a computing system communicate and process information in digital form. An understanding of hardware and software is useful when troubleshooting a computing system that does not work as intended. Computing Systems has three subconcepts, they are: Devices, Hardware and Software, and Troubleshooting.

- **Networks and the Internet (with Cybersecurity):** This involves the networks that connect computing systems. Computing devices do not operate in isolation. Networks connect computing devices to share information and resources and are an increasingly integral part of computing. Networks and communication systems provide greater connectivity in the computing world by providing fast, secure communication and facilitating innovation. Networking and the Internet must also consider Cybersecurity. Cybersecurity, also known as information technology security, involves the protection of computers, networks, programs, and data from unauthorized or unintentional access, manipulation, or destruction. Many organizations, such as government, military, corporations, financial institutions, hospitals, and others collect, process, and store significant amounts of data on computing devices. That data is transmitted across multiple networks to other computing devices. The confidential nature of government, financial, and other types of data requires continual monitoring and protection for the sake of continued operation of vital systems and national security. This concept has two subconcepts within it, they are: Cybersecurity, and Network Communication and Organization.

- **Data and Analysis:** This involves the data that exist and the computing systems that exist to process that data. The amount of digital data generated in the world is rapidly expanding, so the need to process data effectively is increasingly important. Data is collected and stored so that it can be analyzed to better understand the world and make more accurate predictions. This concept has three subconcepts, they are: Collection, Visualization and Transformation, Storage, and Inference and Models

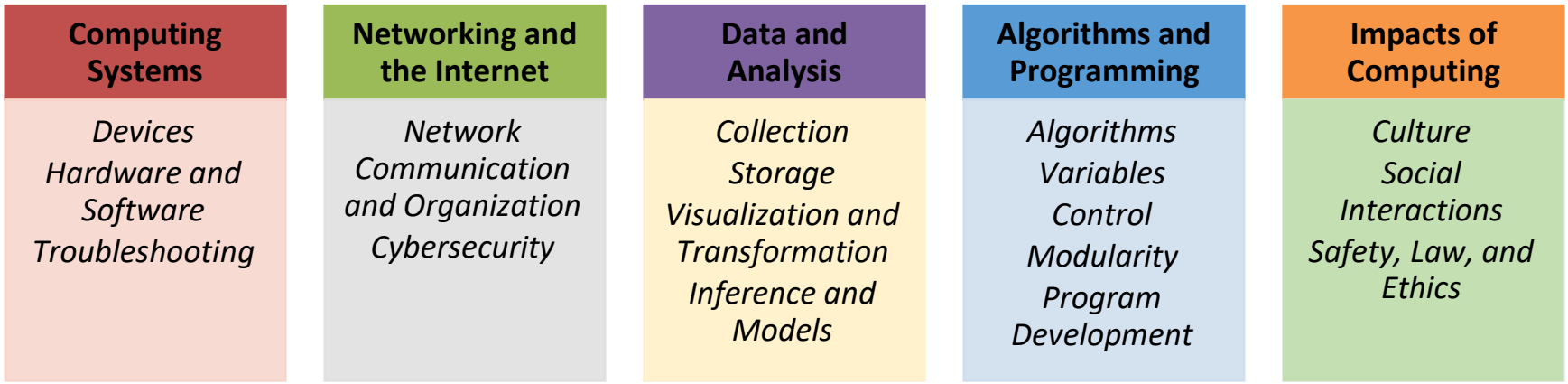
- **Algorithms and Programming:** Involves the use of algorithms. An algorithm is a sequence of steps designed to accomplish a specific task. Algorithms are translated into programs, or code, to provide instructions for computing devices. Algorithms and programming control all

computing systems, empowering people to communicate with the world in new ways and solve compelling problems. The development process to create meaningful and efficient programs involves choosing which information to use and how to process and store it, breaking apart large problems into smaller ones, recombining existing solutions, and analyzing different solutions. This concept has 5 subconcepts, they are: Algorithms, Variables, Control, Modularity, and Program Development

- **Impacts of Computing:** This involves the effect that computing has on daily life. Computing affects many aspects of the world in both positive and negative ways at local, national, and global levels. Individuals and communities influence computing through their behaviors and cultural and social interactions, and in turn, computing influences new cultural practices. An informed and responsible person should understand the social implications of the digital world, including equity and access to computing. This concept has 3 subconcepts, they are: Culture, Social Interactions, and Safety, Law, and Ethics

Concepts are categories that represent major content areas in the field of computer science. They represent specific areas of disciplinary importance rather than abstract, general ideas. Each essential concept is supported by various subconcepts that represent specific ideas within each concept. Figure 1 provides a visual representation of the Essential Concepts and the supporting subconcepts.

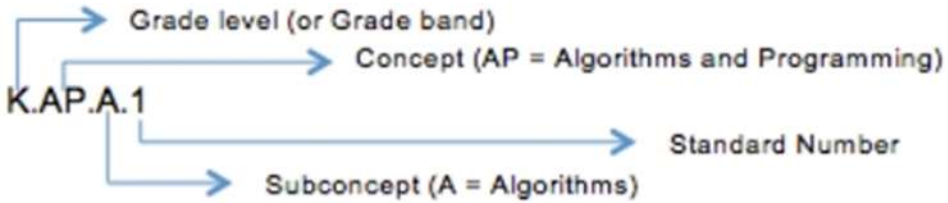
Figure 1: Computer science essential concepts and subconcepts



The pages following break the concepts and subconcepts down by Concept, from Kindergarten through High School. Each Concept is labeled and separated from the next. This will allow teachers to more easily track progression within the standards.

Each standard will list the grade level, the concept, the subconcept, and the standard number. Figure 2 provides an example of the coding for, and how to read, a standard:

Figure 2: Standard Coding Scheme for Standards



Concept: Algorithms and Programming (AP)

Subconcept: Algorithms (A)	
K.AP.A.1	**With teacher assistance, model daily processes by following algorithms (sets of step-by-step instructions) to complete tasks.** *Routines, such as morning meeting, clean-up time, and dismissal, are examples of algorithms that are common in many early elementary classrooms. Just as people use algorithms to complete daily routines, they can program computers to use algorithms to complete different tasks. Algorithms are commonly implemented using a precise language that computers can interpret. For example, students begin to recognize daily step-by-step processes, such as brushing teeth or following a morning procedure, as "algorithms" that lead to an end result.* *Practice(s):* Developing and Using Abstractions: 4.4
Subconcept: Variables (V)	
K.AP.V.1	**With teacher assistance, model the way programs store and manipulate data by using numbers or other symbols to represent information.** *Information in the real world can be represented in computer programs. Students could use thumbs up/down as representations of yes/no, use arrows when writing algorithms to represent direction, or encode and decode words using numbers, pictographs, or other symbols to represent letters or words.* Practice(s): Developing and Using Abstractions: 4.4
Subconcept: Control (C)	
K.AP.C.1	**With teacher assistance, identify 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. Computers follow instructions literally. Sequences are the order of instructions in a program. For example, sequences of instructions include steps for drawing a shape or moving a character across the screen. 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. For example, kindergarten students should be able to recognize loops and sequences in songs, rhymes, and games, such as the song B-I-N-G-O.* Practice(s): Creating Computational Artifacts: 5.1
Subconcept: Modularity (M)	
K.AP.M.1	**With teacher assistance, solve a problem by breaking it down into smaller parts.** *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.* Practice(s): Recognizing and Defining Computational Problems: 3.1

Subconcept: Program Development (PD)	
K.AP.PD.1	**With teacher assistance, develop plans that describe a program's sequence of events, goals, and expected outcomes.** *Programming is used as a tool to create products that reflect a wide range of interests, such as video games, interactive art projects, and digital stories. Students could create a planning document, such as a story map, a storyboard, or a sequential graphic organizer, to illustrate what an end product will do. Students at this stage may complete the planning process with help from their teachers. For example, kindergarten students could illustrate the beginning, middle, and end of a favorite story.* Practice(s): Creating Computational Artifacts, Communicating About Computing: 5.1, 7.2
K.AP.PD.2	**With teacher assistance, identify attribution (credit) 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
K.AP.PD.3	**With teacher assistance, 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. For example, kindergarten students should be able to identify incorrect order in a series of events and place them in the correct order, such as getting ready for school or making a peanut butter sandwich.* Practice(s): Testing and Refining Computational Artifacts: 6.2
K.AP.PD.4	**With teacher assistance, using correct terminology, describe steps taken and choices made during program development.** *At this stage, students should be able to talk or write about the goals and expected outcomes of the instructions they develop they create and the choices that they made when developing their instructions. This could be done using coding journals, discussions with a teacher, or teacher created classroom blogs. For example, kindergarten students could describe their thinking about a story map or set of instructions they develop.* *Practice(s)*: Communicating About Computing: 7.2
Subconcept: Algorithms (A)	
1.AP.A.1	**Model daily processes by following algorithms (sets of step-by-step instructions) to complete tasks.** *Routines, such as morning meeting, clean-up time, and dismissal, are examples of algorithms that are common in many early elementary classrooms. For example, students begin to understand and model daily step-by-step processes, such as brushing teeth, implementing a morning procedure, or following a simple recipe as "algorithms" that lead to an end result.* *Practice(s):* Developing and Using Abstractions: 4.4

Subconcept: Variables (V)	
1.AP.V.1	**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: 4.3
Subconcept: Control (C)	
1.AP.C.1	**Identify 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. Computers follow instructions literally. Sequences are the order of instructions in a program. For example, sequences of instructions include steps for drawing a shape or moving a character across the screen. 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. For example, first grade students independently identify loops and sequences in songs, rhymes, and games, such as the song B-I-N-G-O or the game Red Light/Green Light.* *Practice(s):* Creating Computational Artifacts: 5.1
Subconcept: Modularity (M)	
1.AP.M.1	**Solve a problem by breaking it down into smaller parts.** *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.* *Practice(s):* Recognizing and Defining Computational Problems: 3.1
Subconcept: Program Development (PD)	
1.AP.PD.1	**With teacher assistance identify plans that describe a program's sequence of events, goals, and expected outcomes.** *Programming is used as a tool to create products that reflect a wide range of interests, such as video games, interactive art projects, and digital stories. Students could create a planning document, such as a story map, a storyboard, or a sequential graphic organizer, to illustrate what their end product will do. Students at this stage may complete the planning process with help from their teachers. For example, students create a comic strip with at least 3 panels showing the sequence of a story.* *Practice(s):* Creating Computational Artifacts, Communicating About Computing: 5.3, 7.1

1.AP.PD.2	**With teacher assistance, give attribution (credit) 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
1.AP.PD.3	**With teacher assistance, 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. For example, first graders should be able to identify and fix incorrect order in a series of events, placing them in the correct order, such as washing dishes at home. When the steps repeat, a loop is created.* *Practice(s):* Testing and Refining Computational Artifacts: 6.3
1.AP.PD.4	**Using correct terminology, describe steps taken and choices made during program development.** *At this stage, students should be able to talk or write about the goals and expected outcomes of the instructions they develop and the choices that they made when developing their instructions. This could be done using coding journals, discussions with a teacher, class presentations, or classroom blogs. For example, first grade students do a class presentation sharing the process, choices they made, and outcomes for a fictitious product they developed.* *Practice(s):* Communicating About Computing: 7.2
Subconcept: Algorithms (A)	
2.AP.A.1	**Model daily processes by creating and following algorithms (sets of step-by-step instructions to complete tasks.** *Routines, such as morning meeting, clean-up time, and dismissal, are examples of algorithms that are common in many early elementary classrooms. Just as people use algorithms to complete daily routines, they can program computers to use algorithms to complete different tasks. Algorithms are commonly implemented using a precise language that computers can interpret. For example, students begin to understand and model daily step-by-step processes, such as brushing teeth, implementing a morning procedure, or following a simple recipe as "algorithms" that lead to an end result.* *Practice(s):* Developing and Using Abstractions: 4.3
Subconcept: Variables (V)	
2.AP.V.1	**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: 4.3

Subconcept: Control (C)	
2.AP.C.1	**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. Computers follow instructions literally. Sequences are the order of instructions in a program. For example, sequences of instructions include steps for drawing a shape or moving a character across the screen. 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. For example: students independently identify loops and sequences in songs, rhymes, and games, such as the song Head, Shoulders, Knees and Toes* *Practice(s):* Creating Computational Artifacts: 5.2
Subconcept: Modularity (M)	
2.AP.M.1	**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
Subconcept: Program Development (PD)	
2.AP.PD.1	**Develop plans that describe a program's sequence of events, goals, and expected outcomes.** *Programming is used as a tool to create products that reflect a wide range of interests, such as video games, interactive art projects, and digital stories. Students could create a planning document, such as a story map, a storyboard, or a sequential graphic organizer, to illustrate what an end product will do. Students at this stage may complete the planning process with help from their teachers. For example, students create a graphic organizer modeling the life cycle of a plant.* *Practice(s):* Creating Computational Artifacts, Communicating About Computing: 5.2, 7.2
2.AP.PD.2	**Give attribution (credit) 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. Proper attribution at this stage does not require a formal citation, such as in a bibliography or works cited document. For example, students can give attribution in written form, at a minimum, by listing a website where they got the information, picture, or music.* *Practice(s):* Communicating About Computing: 7.3

2.AP.PD.3	**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. For example: Students could use arrows on a grid to map out a path to a specific coordinate, such as a treasure map. If the steps were repeated, it would create a loop.* *Practice(s):* Testing and Refining Computational Artifacts: 6.2
2.AP.PD.4	**Using correct terminology, describe steps taken and choices made during the iterative process of program (procedure) 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, or discussions with a teacher. Students work together to explain the steps in their procedure.* *Practice(s):* Communicating About Computing: 7.2
Subconcept: Algorithms (A)	
3.AP.A.1	**Recognize and compare multiple algorithms for the same task and determine which are effective.** *Different algorithms can achieve the same result, though sometimes one algorithm might be most appropriate for a specific task.. Students look at different ways to solve the same task and decide which would be the best solution. For example, students might compare algorithms that describe how to get ready for school or how to tie their shoes. Students could use a map and plan multiple algorithms to get from one point to another. 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):* Developing and Using Abstractions, 4.4
Subconcept: Variables (V)	
3.AP.V.1	**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. Students may use mathematical operations to add to the score of a game or subtract from the number of lives in a game. Programs can imply either digital or paper-based designs.* *Practice(s):* Creating Computational Artifacts: 5.2

Subconcept: Control (C)	
3.AP.C.1	**Create programs that include sequences, events, loops, and/or 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. If dialogue is not sequenced correctly when programming a simple animated story, the story will not make sense. Events allow portions of a program to run based on a specific action. 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. Loops allow for the repetition of a sequence of code multiple times. 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. Conditionals allow for the execution of a portion of code in a program when a certain condition is true. 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.* Practice(s): Creating Computational Artifacts: 5.2
Subconcept: Modularity (M)	
3.AP.M.1	**Decompose problems into smaller, manageable subproblems to facilitate the program development process.** *Decomposition is the act of breaking down a task into multiple simpler tasks. Decomposition also enables different people to work on different parts at the same time. 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
Subconcept: Program Development (PD)	
3.AP.PD.1	**With teacher guidance, 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. With teacher guidance, 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
3.AP.PD.2	**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 the creator may not like. Students should identify if ideas were borrowed or adjusted, 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

3.AP.PD.3	**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. Identifying a mistake in a math problem, for example; Sally solved the following problem as 11, there were five groups with six apples in each. How many apples were there? Was she correct? Fix her mistake if she was incorrect.* Practice(s): Testing and Refining Computational Artifacts: 6.1, 6.2
3.AP.PD.4	With teacher guidance, students take on varying roles, **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
3.AP.PD.5	**Describe choices made during program (procedure) development using code comments, presentations, and/or demonstrations.** *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. Students work together to explain the steps in their procedure.* Practice(s): Communicating About Computing: 7.2
Subconcept: Algorithms (A)	
4.AP.A.1	**Compare and refine multiple algorithms for the same task and determine which is the most effective.** 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 might compare algorithms that describe how to get ready for school or how to tie their shoes. 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. 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

Subconcept: Variables (V)	
4.AP.V.1	**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, without a fuller understanding of the technical aspects of variables (such as identifiers and memory locations. Students may use mathematical operations to add to the score of a game or subtract from the number of lives in a game. Programs can imply either digital or paper-based designs.* *Practice(s):* Creating Computational Artifacts: 5.2
Subconcept: Control (C)	
4.AP.C.1	**Create programs that include sequences, events, loops, and/or 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. If dialogue is not sequenced correctly when programming a simple animated story, the story will not make sense. Events allow portions of a program to run based on a specific action. 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. Loops allow for the repetition of a sequence of code multiple times. 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. Conditionals allow for the execution of a portion of code in a program when a certain condition is true. 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.* Practice(s): Creating Computational Artifacts: 5.2
Subconcept: Modularity (M)	
4.AP.M.1	**Decompose problems into smaller, manageable subproblems to facilitate the program development process.** *Decomposition is the act of breaking down a task into multiple simpler tasks. Decomposition also enables different people to work on different parts at the same time. 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
4.AP.M.2	**Modify, remix, or incorporate portions of an existing program into one's own work to 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

Subconcept: Program Development (PD)	
4.AP.PD.1	**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
4.AP.PD.2	**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 the creator 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
4.AP.PD.3	**Test and debug (identify and fix errors) a program/app 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
4.AP.PD.4	**With teacher guidance, students take on varying roles 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
4.AP.PD.5	**Describe choices made during program development using code comments, presentations, and/or demonstrations.** *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

Subconcept: Algorithms (A)	
5.AP.A.1	**Compare, test, and refine multiple algorithms for the same task and determine which is the most effective.** 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. Students test their algorithms to verify their effectiveness. *Practice(s):* Testing and Refining Computational Artifacts, Recognizing and Defining Computational Problems: 6.1, 6.3
Subconcept: Variables (V)	
5.AP.V.1	**Recognizing that the data type determines the values that can be stored and the operations that can be performed on the data.** *Variables are the vehicle through which computer programs store different types of data. At this level, understanding how to use variables is sufficient, without a fuller understanding of the technical aspects of variables (such as identifiers and memory locations). Data types vary by programming language, but many have types for numbers and text. Examples of operations associated with those types include multiplying numbers and combining text. Some visual, block-based languages do not have explicitly declared types but still have certain operations that apply only to particular types of data in a program. Programs can imply either digital or paper-based designs. Students* create programs that use variables to store and modify data. *Practice(s):* Creating Computational Artifacts: 5.2
Subconcept: Control (C)	
5.AP.C.1	**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. For example, if dialogue is not sequenced correctly when programming a simple animated story, the story will not make sense. 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. 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. 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.* Practice(s): Creating Computational Artifacts: 5.1

Subconcept: Modularity (M)	
5.AP.M.1	**Decompose problems into manageable subproblems to facilitate the program development process.** *Decomposition is the act of breaking down a task into multiple, simpler tasks. Decomposition also enables different people to work on different parts at the same time. 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
5.AP.M.2	**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
Subconcept: Program Development (PD)	
5.AP.PD.1	**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
5.AP.PD.2	**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 the creator 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
5.AP.PD.3	**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

5.AP.PD.4	**Take on varying roles 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
5.AP.PD.5	**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. 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
Subconcept: Algorithms (A)	
6.AP.A.1	**Identify planning strategies such as flowcharts or pseudocode, to simulate algorithms that solve problems.** *Students should be able to select planning strategies to organize and sequence an algorithm that addresses a problem, even though they may not actually program the solutions. For example, students might express an algorithm that produces a recommendation for purchasing sneakers based on inputs such as size, colors, brand, comfort, and cost.* *Practice(s): Developing and Using Abstractions: 4.4, 4.1*
Subconcept: Variables (V)	
6.AP.V.1	**Identify variables that represent different data types and perform operations on their values.** *A variable is like a container with a name, in which the contents may change, but the name (identifier) does not. When planning and developing programs, students should decide when and how to declare and name new variables. Students should use naming conventions to improve program readability. For example, possible operations include adding points to the score, combining user input with words to make a sentence, changing the size of a picture, or adding a name to a list of people.* *Practice(s): Creating Computational Artifacts: 5.1, 5.2*
Subconcept: Control (C)	
6.AP.C.1	**Design programs that combine control structures, including nested loops and compound conditionals.** *Control structures can be combined in many ways. Nested loops are loops placed within loops. Compound conditionals combine two or more conditions in a logical relationship (e.g., using AND, OR, and NOT), and nesting conditionals within one another allows the result of one conditional to lead to another. For example, when programming an interactive story, students could use a compound conditional within a loop to unlock a door only if a character has a key AND is touching the door.* *Practice(s): Creating Computational Artifacts: 5.1, 5.2*

Subconcept: Modularity (M)	
6.AP.M.1	**Decompose problems into parts to facilitate the design, implementation, and review of programs.** *In order to understand how programs are designed and used, problems should be broken down into smaller pieces that are easier to work with.* *Practice(s): Recognizing and Defining Computational Problems: 3.2*
6.AP.M.2	**Use procedures to organize code and make it easier to reuse.** *Students should compare procedures and/or functions that are used multiple times within a program to repeat groups of instructions. These procedures can be generalized by defining parameters that create different outputs for a wide range of inputs. For example, a procedure to draw a circle involves many instructions, but all of them can be invoked with one instruction, such as "drawCircle." By adding a radius parameter, the user can easily draw circles of different sizes.* *Practice(s): Developing and Using Abstractions: 4.1, 4.3*
Subconcept: Program Development (PD)	
6.AP.PD.1	**Seek and incorporate feedback from team members and users to refine a solution that meets user needs.** *Development teams that employ user-centered design create solutions (e.g., programs and devices) that can have a large societal impact, such as an app that allows people with speech difficulties to translate hard-to-understand pronunciation into understandable language. Students should seek diverse perspectives throughout the design process to improve their computational artifacts. For example, considerations of the end-user may include usability, accessibility, age-appropriate content, respectful language, user perspective, pronoun use, color contrast, and ease of use.* *Practice(s): Collaborating Around Computing, Fostering an Inclusive Computing Culture: 2.3, 1.1*
6.AP.PD.2	**Incorporate existing code into programs and give attribution.** *Building on the work of others enables students to produce more interesting and powerful creations. Students should use portions of code in their own programs and websites. For example, when creating a side-scrolling game, students may incorporate portions of code that create a realistic jump movement from another person's game. They may also import Creative Commons-licensed images to use in the background. Students should give attribution to the original creators to acknowledge their contributions.* *Practice(s): Developing and Using Abstractions, Creating Computational Artifacts, Communicating About Computing: 4.2, 5.2, 7.3*
6.AP.PD.3	**Test programs using a range of inputs and identify expected outputs.** *At this level, testing should become a deliberate process that is more iterative, systematic, and proactive. For example, having students enter data into Microsoft Excel or Google Sheets to see what outputs are produced.* *Practice(s): Testing and Refining Computational Artifacts: 6.1*
6.AP.PD.4	**Maintain a timeline with specific tasks while collaboratively developing computational artifacts.** *Collaboration is a common and crucial practice in program development. Often, many individuals and groups work on the interdependent parts of a project together. For example, students should assume pre-defined roles within their teams and manage the project workflow using structured timelines.* *Practice(s): Collaborating Around Computing: 2.2*

6.AP.PD.5	**Document programs in order to make them easier to follow, test, and debug.** *Documentation allows creators and others to more easily use and understand a program. Students should provide documentation for end users that explains their artifacts and how they function. For example, students could provide a project overview and clear user instructions. They should also incorporate comments into their programs and communicate their process throughout the design, development, and user experience phases.* *Practice(s): Communicating About Computing: 7.2*
Subconcept: Algorithms (A)	
7.AP.A.1	**Use planning strategies, such as flowcharts or pseudocode, to develop algorithms to address complex problems.** *Complex problems are problems that would be difficult for students to solve computationally. Students should use pseudocode and/or flowcharts to organize and sequence an algorithm that addresses a complex problem, even though they may not actually program the solutions. For example, students might follow an algorithm that produces a recommendation for purchasing sneakers based on inputs such as size, colors, brand, comfort, and cost.* *Practice(s): Developing and Using Abstractions: 4.4, 4.1*
Subconcept: Variables (V)	
7.AP.V.1	**Compare and contrast variables that represent different data types and perform operations on their values.** *A variable is like a container with a name, in which the contents may change, but the name (identifier) does not. When planning and developing programs, students should decide when and how to declare and name new variables. Students should use naming conventions to improve program readability. For example, possible operations include adding points to the score, combining user input with words to make a sentence, changing the size of a picture, or adding a name to a list of people.* *Practice(s): Creating Computational Artifacts: 5.1, 5.2*
Subconcept: Control (C)	
7.AP.C.1	**Design and develop programs that combine control structures, including nested loops and compound conditionals.** *Control structures can be combined in many ways. Nested loops are loops placed within loops. Compound conditionals combine two or more conditions in a logical relationship (e.g., using AND, OR, and NOT), and nesting conditionals within one another allows the result of one conditional to lead to another. For example, when programming an interactive story, students could use a compound conditional within a loop to unlock a door only if a character has a key AND is touching the door.* *Practice(s): Creating Computational Artifacts: 5.1*
Subconcept: Modularity (M)	
7.AP.M.1	**Decompose problems into parts to facilitate the design, implementation, and review of programs.** *In order to design, implement and evaluate programs students will break down problems into smaller parts. For example, students might code one part of a game at a time (sprites, motion, interaction, backgrounds, etc).* *Practice(s): Recognizing and Defining Computational Problems: 3.2*

7.AP.M.2	**Use procedures with parameters to organize code and make it easier to reuse.** *Students should use procedures and/or functions that are used multiple times within a program to repeat groups of instructions. These procedures can be generalized by defining parameters that create different outputs for a wide range of inputs. For example, a procedure to draw a circle involves many instructions, but all of them can be invoked with one instruction, such as "drawCircle." By adding a radius parameter, the user can easily draw circles of different sizes.* *Practice(s): Developing and Using Abstractions, Creating Computational Artifacts: 4.1, 4.3, 5.1, 5.2*
Subconcept: Program Development (PD)	
7.AP.PD.1	**Seek and incorporate feedback from team members and users to refine a solution that meets user needs.** *Development teams that employ user-centered design create solutions (e.g., programs and devices) that can have a large societal impact, such as an app that allows people with speech difficulties to translate hard-to-understand pronunciation into understandable language. Students should begin to seek diverse perspectives throughout the design process to improve their computational artifacts. Considerations of the end-user may include usability, accessibility, age-appropriate content, respectful language, user perspective, pronoun use, color contrast, and ease of use.* *Practice(s): Collaborating Around Computing, Fostering an Inclusive Computing Culture: 2.3, 1.1*
7.AP.PD.2	**Incorporate existing code and media into programs, and give attribution.** *Building on the work of others enables students to produce more interesting and powerful creations. Students should use portions of code and/or digital media in their own programs and websites. For example, when creating a side-scrolling game, students may incorporate portions of code that create a realistic jump movement from another person's game. They may also import Creative Commons-licensed images to use in the background. Students should give attribution to the original creators to acknowledge their contributions.* *Practice(s): Developing and Using Abstractions, Creating Computational Artifacts, Communicating About Computing: 4.2, 5.2, 7.3*
7.AP.PD.3	**Systematically test and refine programs using a range of possible inputs.** *At this level, testing should become a deliberate process that is more iterative, systematic, and proactive students should begin to test programs by considering potential errors, such as what will happen if a user enters invalid input (e.g., negative numbers and 0 instead of positive numbers).* *Practice(s): Testing and Refining Computational Artifacts: 6.1*
7.AP.PD.4	**Distribute and execute tasks while maintaining a project timeline when collaboratively developing computational artifacts.** *Collaboration is a common and crucial practice in program development. Often, many individuals and groups work on the interdependent parts of a project together. Students should assume pre-defined roles within their teams and manage the project workflow using structured timelines. With teacher guidance, they will begin to create collective goals, expectations, and equitable workloads. For example, students may divide the design stage of a game into planning the storyboard, flowchart, and different parts of the game mechanics. They can then distribute tasks and roles among members of the team and assign deadlines.* *Practice(s): Collaborating Around Computing: 2.2*

7.AP.PD.5	**Document programs to make them easier to follow, test, and debug.** *Documentation allows creators and others to more easily use and understand a program. Students should provide documentation for end users that explains their artifacts and how they function. For example, students could provide a project overview and clear user instructions. They should also incorporate comments into their programs and communicate their process throughout the design, development, and user experience phases.* *Practice(s): Communicating About Computing: 7.2*
Subconcept: Algorithms (A)	
8.AP.A.1	**Develop planning strategies, such as flowcharts or pseudocode, to develop algorithms to address complex problems.** *Complex problems are problems that would be difficult for students to solve computationally. Students should use pseudocode and/or flowcharts to organize and sequence an algorithm that addresses a complex problem, even though they may not actually program the solutions. For example, students might express an algorithm that produces a recommendation for purchasing sneakers based on inputs such as size, colors, brand, comfort, and cost. Testing the algorithm with a wide range of inputs and users allows students to refine their recommendation algorithm and to identify other inputs they may have initially excluded.* *Practice(s): Developing and Using Abstractions: 4.4, 4.1*
Subconcept: Variables (V)	
8.AP.V.1	**Create named variables that represent different data types and perform operations on their values.** *A variable is like a container with a name, in which the contents may change, but the name (identifier) does not. When planning and developing programs, students should decide when and how to declare and name new variables. Students should use naming conventions to improve program readability. Examples of operations include adding points to the score, combining user input with words to make a sentence, changing the size of a picture, or adding a name to a list of people.* *Practice(s): Creating Computational Artifacts: 5.1, 5.2*
Subconcept: Control (C)	
8.AP.C.1	**Design and iteratively develop programs that combine control structures, including nested loops and compound conditionals.** *Control structures can be combined in many ways. Nested loops are loops placed within loops. Compound conditionals combine two or more conditions in a logical relationship (e.g., using AND, OR, and NOT), and nesting conditionals within one another allows the result of one conditional to lead to another. For example, when programming an interactive story, students could use a compound conditional within a loop to unlock a door only if a character has a key AND is touching the door.* *Practice(s): Creating Computational Artifacts: 5.1, 5.2*
Subconcept: Modularity (M)	
8.AP.M.1	**Decompose problems into parts to facilitate the design, implementation, and review of programs.** In order to design, implement and evaluate programs, students will break down problems into smaller parts. For example, students might code one part of a game at a time (sprites, motion, interaction, backgrounds, etc). *Practice(s): Recognizing and Defining Computational Problems: 3.2*

8.AP.M.2	**Create procedures with parameters to organize code and make it easier to reuse.** *Students should create procedures and/or functions that are used multiple times within a program to repeat groups of instructions. These procedures can be generalized by defining parameters that create different outputs for a wide range of inputs. For example, a procedure to draw a circle involves many instructions, but all of them can be invoked with one instruction, such as "drawCircle." By adding a radius parameter, the user can easily draw circles of different sizes.* *Practice(s): Developing and Using Abstractions: 4.1, 4.3*
Subconcept: Program Development (PD)	
8.AP.PD.1	**Seek and incorporate feedback from team members and users to refine a solution that meets user needs.** *Development teams that employ user-centered design create solutions (e.g., programs and devices) that can have a large societal impact, such as an app that allows people with speech difficulties to translate hard-to-understand pronunciation into understandable language. Students should begin to seek diverse perspectives throughout the design process to improve their computational artifacts. Considerations of the end-user may include usability, accessibility, age-appropriate content, respectful language, user perspective, pronoun use, color contrast, and ease of use.* *Practice(s): Collaborating Around Computing, Fostering an Inclusive Computing Culture: 2.3, 1.1*
8.AP.PD.2	**Incorporate existing code, media, and libraries into original programs, and give attribution.** *Building on the work of others enables students to produce more interesting and powerful creations. Students should use portions of code, algorithms, and/or digital media in their own programs and websites. At this level, they may also import libraries and connect to web application program interfaces (APIs). For example, when creating a side-scrolling game, students may incorporate portions of code that create a realistic jump movement from another person's game. They may also import Creative Commons-licensed images to use in the background. Students should give attribution to the original creators to acknowledge their contributions.* *Practice(s): Developing and Using Abstractions, Creating Computational Artifacts, Communicating About Computing: 4.2, 5.2, 7.3*
8.AP.PD.3	**Systematically test and refine programs using a range of possible inputs.** *At this level, testing should become a deliberate process that is more iterative, systematic, and proactive. Students should begin to test programs by considering potential errors, such as what will happen if a user enters invalid input (e.g., negative numbers and 0 instead of positive numbers).* *Practice(s): Testing and Refining Computational Artifacts: 6.1*
8.AP.PD.4	**Distribute and execute tasks while maintaining a project timeline when collaboratively developing computational artifacts.** *Collaboration is a common and crucial practice in program development. Often, many individuals and groups work on the interdependent parts of a project together. Students should assume pre-defined roles within their teams and manage the project workflow using structured timelines. With teacher guidance, they will begin to create collective goals, expectations, and equitable workloads. For example, students may divide the design stage of a game into planning the storyboard, flowchart, and different parts of the game mechanics. They can then distribute tasks and roles among members of the team and assign deadlines.* *Practice(s): Collaborating Around Computing: 2.2*

8.AP.PD.5	**Document programs to make them easier to follow, test, and debug.** *Documentation allows creators and others to more easily use and understand a program. Students should provide documentation for end users that explains their artifacts and how they function. For example, students could provide a project overview and clear user instructions. They should also incorporate comments into their programs and communicate their process throughout the design, development, and user experience phases.* *Practice(s): Communicating About Computing: 7.2*
Subconcept: Algorithms (A)	
HS.AP.A.1	**Create prototypes that use algorithms for practical intent, personal expression, or to address a societal issue** *A prototype is a computational artifact that demonstrates the core functionality of a product or process. Prototypes are useful for getting early feedback in the design process, and can yield insight into the feasibility of a product. The process of developing computational artifacts embraces both creative expression and the exploration of ideas to create prototypes and solve computational problems. Students create artifacts that are personally relevant or beneficial to their community and beyond. Students should develop computational artifacts in response to a task or a computational problem that demonstrate the performance, reusability, and ease of implementation of an algorithm.* *Practice(s): Creating Computational Artifacts: 5.2*
Subconcept: Variables (V)	
HS.AP.V.1	**Use lists to simplify solutions, generalizing computational problems instead of repeatedly using simple variables.** *Students should be able to identify common features in multiple segments of code and substitute a single segment that uses lists (arrays) to account for the differences.* *Practice(s): Developing and Using Abstractions: 4.1*
Subconcept: Control (C)	
HS.AP.C.1	**Justify the selection of specific control structures and explain the benefits and drawbacks of choices made, when tradeoffs involve readability and program performance.** *Readability refers to how clear the program is to other programmers and can be improved through documentation. The discussion of performance is limited to a theoretical understanding of execution time; a quantitative analysis is not expected. Control structures at this level may include conditional statements, loops, event handlers, and recursion. Students might compare several implementations of the same algorithm with different structures and discuss the tradeoffs of each implementation.* *Practice(s): Recognizing and Defining Computational Problems: 5.2*

HS.AP.C.2	**Use events that initiate instructions to design and iteratively develop computational artifacts** *In this context, relevant computational artifacts include programs, mobile apps, or web apps for practical intent, personal expression, or to address a societal issue. Events can be user-initiated, such as a button press, or system-initiated, such as a timer firing. At previous levels, students have learned to create and call procedures. Here, students design procedures that are called by events.* *Practice(s): Creating Computational Artifacts: 5.1*
Subconcept: Modularity (M)	
HS.AP.M.1	**Decompose problems into smaller components using constructs such as procedures, modules, and/or objects.** *At this level, students should decompose complex problems into manageable subproblems that could potentially be solved with programs or procedures that already exist. A game program can be made up of objects representing different characters, methods defining how each behaves, and procedures for various events.* *Practice(s): Recognizing and Defining Computational Problems: 3.2*
HS.AP.M.2	**Use procedures within a program, combinations of data and procedures, or independent but interrelated programs to design and iteratively develop computational artifacts.** *Computational artifacts can be created by combining and modifying existing artifacts or by developing new artifacts. Complex programs are designed as systems of interacting procedures, each with a specific role, coordinating for a common overall purpose. The focus at this level is understanding a program as a system with relationships between procedures.* *Practice(s): Creating Computational Artifacts: 5.2*
Subconcept: Program Development (PD)	
HS.AP.PD.1	**Evaluate and refine computational artifacts to make them more usable and accessible.** *Testing and refinement is the deliberate and iterative process of improving a computational artifact. This process includes debugging (identifying and fixing errors) and comparing actual outcomes to intended outcomes. Students should respond to the changing needs and expectations of end users and improve the performance, reliability, usability, and accessibility of artifacts. At this level, students should work through a systematic process that includes feedback from broad audiences.* *Practice(s): Testing and Refining Computational Artifacts: 6.3*
HS.AP.PD.2	**Use team roles and collaborative tools to design and iteratively develop computational artifacts.** *Most software is developed in teams which can include pair programming or other collaborative structures. Team roles in pair programming are alternating driver and navigator but could be more specialized in larger teams. Students may choose to use collaborative tools to aid their team, such as a version control system or project management interface.* *Practice(s): Collaborating Around Computing: 2.1*

HS.AP.PD.3	**Document design decisions using text, graphics, presentations, and/or demonstrations in the development of complex programs.** *Complex programs are designed as systems of interacting modules, each with a specific role, coordinating for a common overall purpose. These modules can be procedures within a program; combinations of data and procedures; or independent, but interrelated, programs. Students might track their design decisions while developing a program, then choose a representation to communicate how each piece of their program contributes to the program as a whole.* *Practice(s): Communicating About Computing: 7.2*

Concept: Impacts of Computing (IC)

Subconcept: Culture (C)	
K.IC.C.1	**Discuss how people lived and worked 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 information on the Internet about a topic or they can download e-books about it directly to a device.* Practice(s): Communicating About Computing: 7.2
Subconcept: Social Interactions (SI)	
K.IC.SI.1	**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 cyberbullying. Teachers should facilitate a discussion in how to avoid sharing information that is inappropriate or that could personally identify them to others, and how to. work in a kind and respectful manner.* Practice(s): Collaborating Around Computing: 2.1
Subconcept: Safety, Law, and Ethics (SLE)	
K.IC. SLE.1	**Keep login information private, and log off of devices appropriately.** *Using computers comes with a level of responsibility. Students should not share login information, keep passwords private, and log off when finished* Practice(s): Communicating About Computing:7.2
Subconcept: Culture (C)	
1.IC.C.1	**Discuss 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: 7.1

Subconcept: Social Interactions (SI)	
1.IC.SI.1	**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 cyberbullying. 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. They should tell an adult if others are sharing things they should not share or are treating others in an unkind or disrespectful manner on online. Privacy should be considered when posting information online: such information can persist for a long time and be accessed by others, even unintended viewers.* *Practice(s):* Collaborating Around Computing: 2.1
Subconcept: Safety, Law, and Ethics (SLE)	
1.IC. SLE.1	**Keep login information private, and log off devices appropriately.** *Using computers comes with a level of responsibility, such as not sharing login information, keeping passwords private, and logging off when finished. Rules guiding personal interactions in the world, apply to online environments as well. For example, students routinely practice logging in and logging out of online resources to protect their personal information. Students should also commit to interacting with only those they know in person in online environments.* *Practice(s):* Communicating About Computing: 7.2
Subconcept: Culture (C)	
2.IC.C.1	**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: 7.1

Subconcept: Social Interactions (SI)	
2.IC.SI.1	**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 cyberbullying. 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. They should tell an adult if others are sharing things they should not share or are treating others in an unkind or disrespectful manner on online. Privacy should be considered when posting information online: such information can persist for a long time and be accessed by others, even unintended viewers.* *Practice(s):* Collaborating Around Computing: 2.1
Subconcept: Safety, Law, and Ethics (SLE)	
2.IC. SLE.1	**Keep login information private, and log off of devices appropriately.** *Using computers comes with a level of responsibility, such as not sharing login information, keeping passwords private, and logging off when finished. Rules guiding personal interactions in the world apply to online environments as well. For example, students routinely practice logging in and logging out of online resources to protect their personal information. Students should also commit to interacting with only those they know in person in online environments.* *Practice(s):* Communicating About Computing: 7.2
Subconcept: Culture (C)	
3.IC.C.1	**Identify computing technologies that have changed the world.** *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. With guidance from their teacher, students 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, in areas, 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
3.IC.C.2	**With teacher guidance, 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

Subconcept: Social Interactions (SI)	
3.IC.SI.1	**Seek opportunities for local collaboration to facilitate communication and innovation.** *Computing influences many social institutions such as family, education, religion, and the economy. People can work in different places and at different times to collaborate and share ideas when they use technologies that reach across the globe. Computing provides the possibility for collaboration and sharing of ideas and allows the benefit of diverse perspectives. These social interactions affect how local and global groups interact with each other, and alternatively, these interactions can change the nature of groups. For example, a class can discuss ideas in the same class, school, or in another state or nation through interactive webinars or pen pals.* Practice(s): Fostering an Inclusive Computing Culture: 1.1
Subconcept: Safety, Law, and Ethics (SLE)	
3.IC. SLE.1	**Use material that is publicly available and/or permissible to use.** *Ethical complications arise from the opportunities provided by computing. The ease of sending and receiving copies of media, such as video, photos, and music, on the Internet, creates the opportunity for unauthorized use, such as online piracy and disregard of copyrights. Students should consider the licenses for the 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
Subconcept: Culture (C)	
4.IC.C.1	**Identify and discuss computing technologies that have changed the world.** *New computing technology is created and existing technologies are modified for many reasons, including to order 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. Students discuss how culture influences changes in technology. Topics could be based on current news content in areas, 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
4.IC.C.2	**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 share their own tastes.* Practice(s): Fostering an Inclusive Computing Culture: 1.2

Subconcept: Social Interactions (SI)	
4.IC.SI.1	**Seek opportunities for local and nationally collaboration to facilitate communication and innovation.** *Computing influences many social institutions such as family, education, religion, and the economy. People can work in different places and at different times to collaborate and share ideas when they use technologies that reach across the globe. Computing provides the possibility for collaboration and sharing of ideas and allows the benefit of diverse perspectives. These social interactions affect how local and global groups interact with each other, and alternatively, these interactions can change the nature of groups. For example, a class can discuss ideas in the same class, school, or in another state or nation through interactive webinars.* Practice(s): Fostering an Inclusive Computing Culture: 1.1
Subconcept: Safety, Law, and Ethics (SLE)	
4.IC.SLE.1	**Use material that is publicly available and/or permissible to use.** *Ethical complications arise from the opportunities provided by computing. The ease of sending and receiving copies of media, such as video, photos, and music, on the Internet, creates the opportunity for unauthorized use, such as online piracy and disregard of copyrights. Students should consider the licenses for the 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
Subconcept: Culture (C)	
5.IC.C.1	**Discuss computing technologies that have changed the world.** *New computing technology is created and existing technologies are modified for many reasons, including in order to increase their benefits, decrease their risks, and meet societal needs. Students discuss topics that relate to the history of technology and the changes in the world due to technology. Students discuss how culture influences changes in technology. Topics could be based on current news content in areas, such as robotics, wireless Internet, mobile computing devices, GPS systems, wearable computing, or how social media has influenced social, cultural and political changes.* Practice(s): Recognizing and Defining Computational Problems: 3.1
5.IC.C.2	**Design 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

Subconcept: Social Interactions (SI)	
5.IC.SI.1	**Seek opportunities for local and global collaboration to facilitate communication and innovation.** *Computing influences many social institutions such as family, education, religion, and the economy. People can work in different places and at different times to collaborate and share ideas when they use technologies that reach across the globe. Computing provides the possibility for collaboration and sharing of ideas and allows the benefit of diverse perspectives. These social interactions affect how local and global groups interact with each other, and alternatively, these interactions can change the nature of groups. For example, a class can discuss ideas in the same class, school, or in another state or nation through interactive webinars.* Practice(s): Fostering an Inclusive Computing Culture: 1.1
Subconcept: Safety, Law, and Ethics (SLE)	
5.IC. SLE.1	**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, such as video, photos, and music, on the Internet, creates the opportunity for unauthorized use, such as online piracy and disregard of copyrights. Students should consider the licenses for the 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
Subconcept: Culture (C)	
6.IC.C.1	**Identify some of the tradeoffs associated with computing technologies that can affect people's everyday activities and career options.** Advancements in computer technology are neither wholly positive nor negative. However, the ways that people use computing technologies have tradeoffs. Students should consider current events related to broad ideas, including privacy, communication, and automation. For example, driverless cars can increase convenience and reduce accidents, but they are also susceptible to hacking. The emerging industry will reduce the number of taxi and shared-ride drivers, but will create more software engineering and cybersecurity jobs. Practice(s): Communicating About Computing: 7.2
6.IC.C.2	**Identify issues of bias and accessibility in the design of existing technologies.** Students should identify, with teacher's guidance, how various technological tools have different levels of usability. For example, facial recognition software that works better for certain skin tones was likely developed with a homogeneous testing group and could be improved by sampling a more diverse population. For example, ways of improving accessibility of technological tools can include allowing a user to change font sizes and colors. This will make an interface usable for people with low vision and benefits users in situations, such as in bright daylight or a dark room. Practice(s): Fostering an Inclusive Computing Culture: 1.2

Subconcept: Social Interactions (SI)	
6.IC.SI.1	**Identify the advantages of creating a computational product by collaborating with others using digital technologies.** Different digital technologies can be used to gather services, ideas, or content from a large group of people, especially from the online community. It can be done at the local level (e.g., classroom or school) or global level (e.g., age-appropriate online communities). For example, a group of students could combine animations to produce a digital community creation. They could also solicit feedback from many people though use of online communities and electronic surveys. Practice(s): Collaborating Around Computing, Creating Computational Artifacts: 2.4, 5.2
Subconcept: Safety, Law, and Ethics (SLE)	
6.IC. SLE.1	**Describe how some digital information can be public or can be kept private and secure.** Sharing information online can help establish, maintain, and strengthen connections between people. Students should consider current events related to broad ideas, including privacy, communication, and automation. For example, students can discuss how their privacy settings on social media affect who can view their information. Practice(s): Communicating About Computing: 7.2
Subconcept: Culture (C)	
7.IC.C.1	**Explain how some of the tradeoffs associated with computing technologies can affect people's everyday activities and career options.** *Advancements in computer technology are neither wholly positive nor negative. However, the ways that people use computing technologies have tradeoffs. Students should consider current events related to broad ideas, including privacy, communication, and automation. For example, driverless cars can increase convenience and reduce accidents, but they are also susceptible to hacking. The emerging industry will reduce the number of taxi and shared-ride drivers, but will create more software engineering and cybersecurity jobs.* Practice(s): Communicating About Computing: 7.2
7.IC.C.2	**Discuss how bias and accessibility issues can impact the functionality of existing technologies.** *Students should discuss the usability of various technology tools (e.g., apps, games, and devices) with the teacher's guidance. For example, facial recognition software that works better for certain skin tones was likely developed with a homogeneous testing group and could be improved by sampling a more diverse population.* Practice(s): Fostering an Inclusive Computing Culture: 1.2

Subconcept: Social Interactions (SI)	
7.IC.SI.1	**Describe the process for creating a computational product by collaborating with others using digital technologies.** *Crowdsourcing can be used as a platform to gather services, ideas, or content from a large group of people, especially from the online community. It can be done at the local level (e.g., classroom or school) or global level (e.g., age-appropriate online communities). For example, a group of students could combine animations to produce a digital community creation. They could also solicit feedback from many people though use of online communities and electronic surveys.* Practice(s): Collaborating Around Computing, Creating Computational Artifacts: 2.4, 5.2
Subconcept: Safety, Law, and Ethics (SLE)	
7.IC. SLE.1	**Identify the benefits and risks associated with sharing information digitally.** Sharing information online can help establish, maintain, and strengthen connections between people. For example, it allows artists and designers to display their talents and reach a broad audience. However, security attacks often start with personal information that is publicly available online. Social engineering is based on tricking people into revealing sensitive information and can be thwarted by being wary of attacks, such as phishing and spoofing. Practice(s): Communicating About Computing: 7.2
Subconcept: Culture (C)	
8.IC.C.1	**Compare and contrast tradeoffs associated with computing technologies that affect people's everyday activities and career options.** Advancements in computer technology are neither wholly positive nor negative. However, the ways that people use computing technologies have tradeoffs. Students should consider current events related to broad ideas, including privacy, communication, and automation. For example, driverless cars can increase convenience and reduce accidents, but they are also susceptible to hacking. The emerging industry will reduce the number of taxi and shared-ride drivers, but will create more software engineering and cybersecurity jobs. Practice(s): Communicating About Computing: 7.2
8.IC.C.2	**Develop a solution to address an issue of bias or accessibility in the design of existing technologies.** Students should test and discuss the usability of various technology tools (e.g., apps, games, and devices) with the teacher's guidance. For example, facial recognition software that works better for certain skin tones was likely developed with a homogeneous testing group and could be improved by sampling a more diverse population. When discussing accessibility, students may notice that allowing a user to change font sizes and colors will not only make an interface usable for people with low vision but also benefits users in various situations, such as in bright daylight or a dark room. Practice(s): Fostering an Inclusive Computing Culture: 1.2

Subconcept: Social Interactions (SI)	
8.IC.SI.1	**Collaborate with contributors by using digital technologies when creating a computational product.** Crowdsourcing can be used as a platform to gather services, ideas, or content from a large group of people, especially from the online community. It can be done at the local level (e.g., classroom or school) or global level (e.g., age-appropriate online communities). For example, a group of students could combine animations to produce a digital community creation. They could also solicit feedback from many people though use of online communities and electronic surveys. Practice(s): Collaborating Around Computing, Creating Computational Artifacts: 2.4, 5.2
Subconcept: Safety, Law, and Ethics (SLE)	
8.IC. SLE.1	**Evaluate the benefits and risks associated with sharing information digitally.** Sharing information online can help establish, maintain, and strengthen connections between people. For example, it allows artists and designers to display their talents and reach a broad audience. However, security attacks often start with personal information that is publicly available online. Social engineering is based on tricking people into revealing sensitive information and can be thwarted by being wary of attacks, such as phishing and spoofing. For example, students could brainstorm reasons why individuals would want to share information online and the potential risks of doing so. Practice(s): Communicating About Computing: 7.2
Subconcept: Culture (C)	
HS.IC.C.1	**Evaluate the ways access to computing impacts personal, ethical, social, economic, and cultural practices.** *Computing may improve, harm, or maintain practices. Equity deficits, such as minimal exposure to computing, access to education, and training opportunities, are related to larger, systemic problems in society. Students should be able to evaluate the accessibility of a product to a broad group of end users, such as people who lack access to broadband or who have various disabilities.* *Practice(s): Fostering an Inclusive Computing Culture: 1.2*
HS.IC.C.2	**Test and refine computational artifacts to reduce bias and equity deficits.** *Biases could include incorrect assumptions developers have made about their user base or data. Students should begin to identify potential bias during the design process to maximize accessibility in product design and become aware of professionally accepted accessibility standards to evaluate computational artifacts for accessibility.* *Practice(s): Fostering an Inclusive Computing Culture: 1.2*
HS.IC.C.3	**Demonstrate ways a given algorithm applies to problems across disciplines.** *Computation can share features with disciplines such as art and music by algorithmically translating human intention into an artifact. Students should be able to identify real-world problems that span multiple disciplines and can be solved computationally, such as increasing bike safety with new helmet technology.* *Practice(s): Recognizing and Defining Computational Problems: 3.1*

Subconcept: Social Interactions (SI)	
HS.IC.SI.1	**Analyze the impact of collaborative tools and methods that increase social connectivity.** *Many aspects of society, especially careers, have been affected by the degree of communication afforded by computing. The increased connectivity between people in different cultures and in different career fields has changed the nature and content of many careers. Students should explore different collaborative tools and methods used to solicit input from team members, classmates, and others, such as participation in online forums or local communities. For example, students could compare ways different social media tools could help a team become more cohesive.* *Practice(s): Collaborating Around Computing: 2.4*
Subconcept: Safety, Law, and Ethics (SLE)	
HS.IC. SLE.1	**Explain the beneficial and harmful effects that intellectual property laws can have on innovation.** *Laws govern many aspects of computing, such as privacy, data, property, information, and identity. These laws can have beneficial and harmful effects, such as expediting or delaying advancements in computing and protecting or infringing upon people's rights. International differences in laws and ethics have implications for computing. For examples, laws that mandate the blocking of some file-sharing websites may reduce online piracy but can restrict the right to access information. Students should be aware of intellectual property laws and be able to explain how they are can be used to protect the interests of innovators or can be potentially be misused.* *Practice(s): Communicating About Computing: 7.3*
HS.IC. SLE.2	**Explain the privacy concerns related to the collection and generation of data through automated processes that may not be evident to users.** *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. This automated and non-evident collection can raise privacy concerns, such as social media sites mining an account even when the user is not online. Students might review situations where this automated collection has led to unintended consequences or accidental breaches in privacy.* *Practice(s): Communicating About Computing: 7.2*
HS.IC. SLE.3	**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 might review case studies or current events which present an ethical dilemma when an individual's right to privacy is at odds with the safety, security, or wellbeing of a community.* *Practice(s): Communicating About Computing: 7.3*