2024
Computer Science Standards of Learning

Grade 8 Instructional Guide

VIRGINIA DEPARTMENT OF EDUCATION

Copyright © 2025
Virginia Department of Education
P.O. Box 2120
Richmond, Virginia 23218-2120
http://www.doe.virginia.gov

All rights reserved. Reproduction of these materials for instructional purposes in public school classrooms in Virginia is permitted.

Superintendent of Public Instruction
Emily Anne Gullickson

Assistant Superintendent of Teaching and Learning
Michelle Wallace, Ph.D.

Office of Educational Technology and Classroom Innovation
Calypso Gilstrap, Associate Director
Keisha Tennessee, Computer Science Coordinator

NOTICE
The Virginia Department of Education does not unlawfully discriminate on the basis of race, color, sex, national origin, age, or disability in employment or in its educational programs or services.

Table of Contents

2024 COMPUTER SCIENCE *STANDARDS OF LEARNING*

Introduction

Virginia's Computer Science standards aim to raise our aspirations for computational instruction to enable students to engage and thrive in a digital world. Beginning in the earliest grades and continuing through 12th grade, students must develop a foundation of computer science knowledge and learn new approaches to problem solving that harness the power of computational thinking to become both users and creators of computing technology.

It is important for every student to engage in computer science education from the earliest ages. This early and sustained access equips students with foundational problem-solving practices, develops their understanding of how current and emerging computer science technologies work, and fosters curiosity, interest, and innovation with computer science.

Foundational Principles

Computer Literacy is foundational to learning and post-secondary success as technology becomes increasingly incorporated into all aspects of everyday life. Computer Literacy provides critical knowledge and skills for all subject areas including mathematics, science, history, English, and fine arts. By applying computer science as a tool for learning and expression in a variety of disciplines and interests, students will actively and proficiently participate in a world that is increasingly influenced by digital technology.

Computer Science fosters problem solving skills that are essential to all educational disciplines and post-secondary employment opportunities. Understanding how multi-step solutions are executed within computer programs allows students the opportunity to use metacognitive strategies with tasks they are performing as they work and study in any topic area. Computer Science should become an essential part of Virginia K-12 education, accessible by all, rather than a vocational part of education only for those headed to technology-based employment.

Computer Science instruction must maintain the pace of technology evolution to prepare students for the workforce. Computer science is a core technology component for students to have the ability to adapt to the future evolution of work. The workforce of the future will increasingly require that all adults effectively work in digital environments and utilize technology both ethically and responsibly. As a result, we must prioritize preparing all students with integral computer science learning opportunities throughout their academic career to ensure they are prepared for a post-secondary success in a digital world that includes computer-based problem solving, artificial intelligence and communication rooted in the use of digital tools.

Students should gain specific digital and computational concepts to harness the power of computer science and derivative applications, such as machine learning, online programming, virtual reality, and Artificial Intelligence (AI), to embrace innovation and chart the future of individuals, business, and government responsibly.

Instructional Intent and Integration

Computer science is an academic discipline that encompasses both conceptual foundations and applied practices. It can be taught effectively with or without computing devices, as many key skills, such as logical reasoning, pattern recognition, decomposition, and sequencing can be developed through with or without a computing device.

In primary grades, overlapping concepts between computer science and other content areas may be taught within the same instructional context. When doing so, it is essential that educators intentionally align instruction to ensure that the full intent and specifications of the computer science standard are addressed, even when the learning experience is shared with another content area.

As students' progress into upper elementary and beyond, instruction should be explicit, ensuring students are able to identify and understand the computer science concepts and practices embedded within those shared experiences. By naming the connections and calling out the domain specific elements of computer science, students can deepen their disciplinary understanding, build metacognitive awareness, and transfer their knowledge and skills across contexts.

It is important to recognize that not all computer science concepts will naturally overlap with other subjects. Concepts such as algorithms, data representation, networks, and programming require dedicated instructional time and may be taught independently of other content areas. Whether through integration or stand-alone instruction, computer science should be approached with the same level of intentionality and rigor as other academic subjects, ensuring students develop a coherent and comprehensive understanding from kindergarten through grade 12.

Disclaimer: The Virginia Department of Education (VDOE) does not endorse or recommend any commercial products, services, or platforms. Any trademarks, logos, or images displayed in this instructional guide are used solely for educational and illustrative purposes to support conceptual understanding. Their inclusion does not constitute an endorsement by the VDOE of the referenced products, services, companies, or organizations.

Eighth Grade: Computer Science *Standards of Learning*

In Eighth Grade, students show mastery of understanding of the role of computing systems and data transmission over the Internet. Students contrast physical and digital safeguards implemented to protect electronic information from potential threats. Moreover, students assess the social and ethical implications of computing technologies from the perspective of both the creator and the consumer of computing technologies. The design and development of computing technologies are evaluated to account for the needs and wants of end users. Students continue to build upon previous knowledge and skills and create programs that contain multiple control structures. Their computational thinking skills are honed through the development of programs utilizing diverse data types, the development of computational models, and the use of pattern recognition and abstraction to make recommendations and predictions. As computer science skills and concepts can be applied to various careers, students will build upon their awareness of their skillsets and create education and training plans that foster continued pursuits of expanding their computer skills and knowledge to foster aspirations for post-secondary opportunities within computer science.

Algorithms and Programming (AP)

8.AP.1 The student will apply computational thinking to construct programs to accomplish a task as a means of creative expression or scientific exploration.

a. Identify patterns and repeated steps in an algorithm, problem, or process.
b. Decompose an algorithm, problem, or process into sub-components.
c. Abstract relevant information to identify essential details.
d. Use pseudocode, decision trees, or flowcharts to illustrate complex problems as algorithms.

8.AP.2 The student will plan and implement algorithms that include sequencing, loops, variables, user input, conditional control structures, functions, and various data types.

a. Describe the concept of input and output of various data types for use in a computer program.
b. Plan an algorithm using plain language, pseudocode, or diagrams.
c. Write and test algorithms expressed using block-based or text-based programming languages.

8.AP.3 The student will use the iterative design process to create, test, and debug programs using a block-based or text-based programming language.

a. Create and test programs that contain multiple control structures.
b. Trace and predict outcomes of programs.
c. Analyze the outcomes of programs to identify logic and syntax errors.

d. Analyze and describe the results of a program to assess validity of outcomes.
e. Revise and improve an algorithm to resolve errors or produce desired outcomes.

8.AP.4 The student will incorporate work from others into programs and projects.

a. Explain the role of Creative Commons licensing for the use and modification or "remixing" of information.
b. Utilize Creative Commons assets in a programming project.
c. Use and remix code from other projects within a programming project and provide proper attribution.

Computing Systems (CSY)

8.CSY.1 The student will recommend and design improvements to computing devices based on the needs of various users.

a. Analyze existing computing devices for advantages and limitations.
b. Recommend and design improvements to computing devices based on user interactions.

8.CSY.2 The student will apply computational thinking to troubleshoot and document hardware and software-related problems.

a. Apply systematic processes to resolve hardware, software, and connectivity-related problems.
b. Design an end-user document/guide to resolve hardware, software, and connectivity-related problems.

Cybersecurity (CYB)

8.CYB.1 The student will investigate and describe ways to protect sensitive data from malware and other attacks.

a. Identify impacts of hacking, ransomware, scams, phishing, fake vulnerability scans and the ethical and legal concerns.
b. Describe how cyber-attacks can affect a computing system.
c. Compare and contrast safe and unsafe computing practices.
d. Explore how industries and emerging technologies are addressing cyber solutions.
e. Model common prevention practices for cyber-attacks.

8.CYB.2 The student will investigate and explain how physical and digital security measures can protect electronic information for businesses, governments, and organizations.

a. Investigate and explain how physical and digital security measures are used to safeguard electronic information.
b. Research the advantages and limitations of different security measures in protecting users against security threats.
c. Explore how emerging technologies may affect methods to safeguard personal and public data.

Data and Analysis (DA)

8.DA.1 The student will create computational models to simulate events or represent phenomena.

a. Compare and contrast the use of computational models and simulations to analyze patterns and replicate phenomena.
b. Design and create complex computational models that simulate dynamic systems (abstraction), incorporating multiple variables and interactions.
c. Refine computational models based on generated outcomes.

8.DA.2 The student will evaluate computational models to analyze patterns and make recommendations or predictions.

a. Define data biases within a dataset and the unintended consequences that may impact data reliability and final analysis.
b. Analyze patterns and interpret data generated by computational models and simulations, identifying meaningful patterns and relationships.
c. Utilize data visualization techniques to communicate and present findings derived from computational models and simulations.

Impacts of Computing (IC)

8.IC.1 The student will assess the social impacts and ethical considerations of computing technologies.

a. Analyze the impact of sharing data through computing technologies.
b. Critique the role the Internet plays in social life, the global economy, and culture.
c. Evaluate online and print sources for credibility and reliability.
d. Research and discuss factors that impact access and availability to computing technologies.
e. Discuss ethical issues around cybersecurity and networks: censorship, privacy, safety, and access.

8.IC.2 The student will analyze and evaluate the ramifications of screen time in one's life.

a. Analyze scenarios or case studies to assess the impact of screen time on one's physical and mental health.
b. Justify the argument that excessive screen time and video games can have significant consequences for the physical, emotional, and cognitive development of children and adolescents.

8.IC.3 The student will identify opportunities for education, training, and preparation to enter into a chosen computer science career field.

a. Identify an education and training plan for a chosen computer science career.
b. Outline the use of computer science skills required in a chosen career.
c. Develop short-and long-term goals for a chosen career.
d. Research emerging trends in a chosen career path.

Networks and the Internet (NI)

8.NI.1 The student will model and describe the role of computing devices in transmitting data in and on computing networks and the Internet.

a. Identify the roles of computing devices: routers, switches, servers, and clients communicating over a network.
b. Design a network topology of computing devices.
c. Demonstrate how data is transmitted over networks and the Internet.
d. Analyze factors that strengthen or weaken network connectivity.

Computer Science

Instructional Guide

This instructional guide, a companion document to the 2024 Computer Science *Standards of Learning*, amplifies each standard by defining the core knowledge and skills in practice, supporting teachers and their instruction, and serving to transition classroom instruction from the 2017 Computer Science *Standards of Learning* to the newly adopted standards.

Computer Science Instructional Guide Framework

This instructional guide includes, Understanding the Standard, Concepts and Connections, Opportunities for Integration, and Skills in Practice aligned to each standard. The purpose of each is explained.

Understanding the Standard

This section is designed to unpack the standards, providing both students and teachers with the necessary knowledge to support effective instruction. It includes core concepts that students are expected to learn, as well as background knowledge that teachers can use to deepen their understanding of the standards and plan standards-aligned lessons.

Concepts and Connections

This section outlines concepts that transcend grade levels and thread through the K through 12 computer science as appropriate at each level. Concept connections reflect connections to prior grade-level concepts as content and practices build within the discipline as well as potential connections across disciplines. The connections across disciplines focus on direct standard alignment, where concepts and practices in computer science overlap with similar ideas in other disciplines.

Computer Science connections are aligned to the: 2024 English *Standards of Learning*, 2023 History and Social Science, 2023 Mathematics *Standards of Learning*, 2020 Digital Learning Integration *Standards of Learning*, and 2018 Science *Standards of Learning*.

These cross-disciplinary concepts and practices are foundational for effective interdisciplinary integration.

Opportunities for Integration

This section provides lesson ideas for integrating computer science with English, history and social science, mathematics, and science through multidisciplinary, interdisciplinary, and transdisciplinary approaches. Lesson ideas may involve the integration of standards that may or may not be directly aligned yet are strategically taught together to achieve a purposeful and authentic learning experience that supports meaningful student outcomes such as deeper understanding, skill transfer, and real-world application.

Skills in Practice

This section focuses on instructional strategies that teachers can use to develop students' skills, deepen their conceptual understanding, and encourage critical thinking. These practices are designed to support curriculum writers and educators in weaving pedagogical approaches that deepen student understanding of unit and course objectives, ultimately enhancing learning outcomes. This section provides a framework for planning effective and engaging lessons.

Eight Grade: Computer Science Instructional Guide

In Eighth Grade, students show mastery of understanding of the role of computing systems and data transmission over the Internet. Students contrast physical and digital safeguards implemented to protect electronic information from potential threats. Moreover, students assess the social and ethical implications of computing technologies from the perspective of both the creator and the consumer of computing technologies. The design and development of computing technologies are evaluated to account for the needs and wants of end users. Students continue to build upon previous knowledge and skills and create programs that contain multiple control structures. Their computational thinking skills are honed through the development of programs utilizing diverse data types, the development of computational models, and the use of pattern recognition and abstraction to make recommendations and predictions. As computer science skills and concepts can be applied to various careers, students will build upon their awareness of their skillsets and create education and training plans that foster continued pursuits of expanding their computer skills and knowledge to foster aspirations for post-secondary opportunities within computer science.

Algorithms and Programming (AP)

8.AP.1 The student will apply computational thinking to construct programs to accomplish a task as a means of creative expression or scientific exploration.

a. Identify patterns and repeated steps in an algorithm, problem, or process.
b. Decompose an algorithm, problem, or process into sub-components.
c. Abstract relevant information to identify essential details.
d. Use pseudocode, decision trees, or flowcharts to illustrate complex problems as algorithms.

Understanding the Standard

Computational thinking (CT) is a structured, solution-driven problem-solving approach that leverages logical reasoning and systematic analysis to address complex challenges. It often results in the creation of a computational artifact or the implementation of an algorithmic process. Computational thinking is universally applicable across disciplines that can be used to deconstruct complex problems, identify essential patterns, and formulate efficient, scalable solutions. Within computer science, computational thinking serves as a foundational skill, underpinning algorithm design, data analysis, and the development of technology-driven solutions to real-world problems. Computational thinking consists of decomposition, pattern analysis, abstraction, and algorithmic thinking.

- Decomposition is the problem-solving practice of breaking apart a problem or process into subcomponents. This involves identifying and recognizing relationships among the parts.

- Abstraction is the problem-solving practice of representing and simplifying relevant information to identify essential details. This involves hiding less important details.
- Pattern analysis is the problem-solving practice of recognizing commonalities, differences, and predictable relationships like sequences.
- Algorithmic thinking is the problem-solving practice of sequencing information.

Computational thinking (CT) is universally applicable across various fields, allowing individuals to break down complex problems and develop efficient solutions. Its role in computer science is particularly important, as it serves as the foundation for designing algorithms, analyzing data, and solving real-world challenges through technology.

Computational thinking is evident when students approach problem-solving by employing strategies such as:
- Breaking a problem or task into smaller steps or manageable parts(decomposition),
- Identifying repeated, reoccurring actions or characteristics(patterns),
- Focusing on key information and disregarding irrelevant details (abstraction),
- Creating step-by-step instructions or algorithms to solve a problem or task and the ability to compare different methods for efficiency (algorithm comparison).

Teachers should intentionally design learning experiences that foster the practical application of computational thinking, such as hands-on programming projects and complex problem-solving tasks that reflect authentic, real-world contexts. The creation of meaningful, developmentally appropriate tasks accompanied by differentiated levels of challenges to ensure that all students can engage with and advance their computational thinking competencies. Assessment practices should prioritize the evaluation of students' cognitive processes, strategies, and metacognitive reflections rather than focusing solely on the final product. Structured collaborative activities not only cultivate teamwork and communication skills but also simulate authentic practices within professional computing environments. Furthermore, integrating computational thinking across disciplinary boundaries promotes deeper conceptual understanding and highlights its interdisciplinary relevance.

[8.AP.1a] Pattern identification involves analyzing an algorithm, problem, or process to detect recurring structures, sequences, or behaviors. A pattern refers to a recurring sequence or set of operations that exhibit predictable repetition. This concept is often exemplified through iterative control structures such as loops, which enable the execution of a code block multiple times based on defined conditional expressions. Recognizing these patterns supports the abstraction and generalization of solutions, allowing for the elimination of redundancy, enhancement of algorithmic efficiency, and development of reusable components. Instruction should emphasize how loop constructs, conditional logic, and control flow structures operationalize pattern recognition in practice, optimizing problem-solving strategies and enabling scalable, dynamic computation.

- Consider the following example: A loop that calculates the sum of numbers, the repeated action is adding the next number in the list. This pattern of repetition makes the process more efficient, allowing programmers to automate tasks that would otherwise require multiple lines of code. Understanding how patterns work in everyday life helps students recognize how these same principles apply in programming, particularly in using loops to handle repetitive tasks.

[8.AP.1b] Decomposition is the systematic process of breaking down a complex algorithm, problem, or process into smaller, more manageable sub-components. This modular approach enhances computational efficiency and clarity by enabling focused analysis and incremental solution development for each part. Rather than attempting to address the entire problem holistically, each functional unit or procedural element is isolated, examined, and addressed individually. This stepwise strategy improves computational efficiency, supports parallel development, and lays the groundwork for designing scalable, maintainable algorithms.

- Decomposition example: When designing a robot to navigate a maze, the overall task is too complex to solve holistically. To manage this complexity, the problem is decomposed into distinct subcomponents such as detecting walls, controlling motor movement, processing sensor input, and determining navigation paths. Each module represents a focused task within the broader system. This structured breakdown allows algorithmic development for each subcomponent to proceed independently, improving clarity, debugging, and eventual system integration. Through decomposition, the problem is made tractable and aligned with modular design principles essential in computer science.

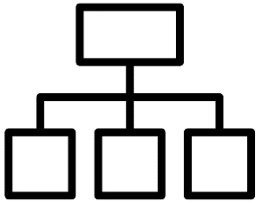
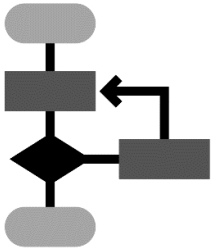
[8.AP.1c] Abstraction is the cognitive process of distilling a problem or system to its most critical elements by filtering out extraneous or non-essential information. This practice reduces complexity and enhances clarity, allowing for the development of simplified representations or models that retain only the features relevant to the task at hand. By focusing on high-level functionality rather than low-level implementation details, abstraction enables more efficient problem-solving, facilitates the design of generalized algorithms, and supports the creation of modular, reusable components.

- Abstraction example: Encryption in cybersecurity exemplifies abstraction by concealing the underlying complexity of cryptographic algorithms. Through this process, readable information (plaintext) is transformed into an unreadable format (ciphertext) using mathematically sophisticated procedures. End users interact with encryption tools such as secure messaging apps or file protection systems—without needing to comprehend the algorithmic details or cryptographic protocols involved. This abstraction enables the application of advanced security measures while simplifying user interaction, illustrating how abstraction supports both usability and robust digital protection in modern computing systems.

[8.AP.1d] Planning is a critical phase in algorithm and program development, often initiated through the creation of a design document. This process helps structure students' thinking as they generate, organize, and refine ideas into well-formed algorithms. Design documents may take various forms such as outlines, brainstorming notes, pseudocode, diagrams, or other pre-coding artifacts and provide a structured space for

planning, drafting, revising, and editing algorithmic solutions. During drafting, students articulate the logic of their solution using plain language or structured representations, anticipating possible errors and areas for refinement. Through revision and editing, they refine the sequence of steps, adjust variable usage, and ensure clarity and logical flow.

To support this process, abstraction tools such as pseudocode, decision trees, and flowcharts allow students to illustrate complex problems as algorithms. These representational tools help externalize and communicate the logic and control flow of a solution without requiring full implementation in a programming language. By engaging in iterative design using these formats, students deepen their computational thinking, develop precision in algorithm construction, and build foundational practices aligned with real-world software development workflows.

Pseudocode	**Decision Tree**	**Flowchart**
A simplified, plain-language description of the steps in an algorithm, bridging the gap between human logic and actual code.	A flowchart-like structure is used for decision-making, where each node represents a choice or condition, and branches lead to possible outcomes.	A diagram that represents a sequence of steps and decisions needed to perform a process or solve a problem, using symbols to denote actions and control flow.
Example: For a program to find the largest number in a list: 1. Set largest to first number in list. 2. For each number in list, if number > largest, set largest to number. 3. Print largest.	Example: A decision tree for choosing transportation: - If the distance is less than 1 mile, walk. - If more than 1 mile and less than 5 miles, bike. 	Example: A flow chart for a birthday cake with candles animation: - Start, play birthday song, have candles been blown out? Yes – stop song, No – play birthday song 

Concepts and Connections

CONCEPTS

Computational thinking is a structured approach to problem-solving that applies logical reasoning and systematic analysis to address complex challenges. It involves identifying patterns and repeated steps in processes, decomposing problems into smaller, manageable components, and abstracting essential details to reduce complexity, and designing algorithmic solutions using formal representations such as pseudocode, decision trees, and flowcharts to clearly define logic and control flow.

CONNECTIONS

Within the grade level/course: At this grade level, students engage in computational thinking practices like decomposition, abstraction, and algorithmic thinking to construct programs for solving real-world problems. These practices not only enhance problem-solving skills in computer science but also support critical thinking and organizational strategies across subjects like writing narratives, analyzing geospatial tools, modeling mathematical functions, and investigating scientific processes. (8.AP.1).

Vertical Progression: In Grade 7, students expand on abstraction by designing algorithms for the purposes of accomplishing a task for creative expression or scientific exploration. (7.AP.1).

ACROSS CONTENT AREAS

History and Social Science

- **WG.1** The student will apply history and social science skills to explain how geographic information and geospatial tools are used to make decisions.

Mathematics

- **8.PFA.3** The student will represent and solve problems, including those in context, by using linear functions and analyzing their key characteristics.

Science

- **8.1** The student will demonstrate an understanding of scientific and engineering practice. *8.1 standard is integrated within science content and not taught in isolation. Potential science concepts to apply 8.1 include: PS.3.b The student will investigate and understand that matter has properties and is conserved in chemical and physical processes by understanding that pure substances can undergo physical and chemical changes that may result in a change of properties.*

DIGITAL LEARNING INTEGRATION

- **6-8.CT** Students develop and employ strategies for understanding and solving problems in ways that leverage the power of technological methods, including those that leverage assistive technologies, to develop and test solutions.
  D. Students demonstrate an understanding of how automation works and use algorithmic thinking to design and automate solutions.

Opportunities for Integration

Curriculum integration strengthens conceptual understanding and skill application. This can be done through multidisciplinary, interdisciplinary, and transdisciplinary approaches to integration. The examples below illustrate multiple ways to integrate computer science.

English

- Students use pseudocode or flowcharts to plan the structure of a narrative before drafting. By organizing key events, character actions, and transitions in a structured sequence, they reinforce computational thinking strategies such as decomposition and pattern recognition while improving narrative coherence

History and Social Science

- Students use computational tools to analyze geographic data and model decision-making scenarios. Students create a program or flowchart that helps users determine the best locations for settlements based on factors like climate, natural resources, and transportation routes, reinforcing how geospatial tools aid in decision-making.

Mathematics

- Students design a program using a block-based or text-based coding tool to model a linear function. The program could take user input values, calculate outputs based on an equation, and display how changes in input affect the function graphically, helping students visualize key characteristics of linear relationships.

Science

- Students create a decision tree or flowchart to classify changes in matter as physical or chemical. By following a structured series of questions (such as whether a substance changes or forms a new substance)students can systematically determine the type of change. The visual tool will guide students through key decision points and reinforce their understanding of observable signs that distinguish physical changes from chemical reactions.

Skills in Practice

Students should engage in the following practices to deepen their conceptual understanding and enhance the application of skills aligned with the Computer Science *Standards of Learning*. These practices are explained in more detail in Appendix A.

A. Fostering Collaborative Computing Practices:

2. Include Multiple Perspectives
3. Create and Accept Feedback

B. Fostering Computational Thinking Practices:

1. Decompose Real-World Problems
2. Explore Common Features and Extract Patterns
3. Apply Abstraction to Simplify, Represent, and Problem Solve
4. Apply Algorithmic Thinking to Problem Solve and Create
5. Apply Computational Thinking Practices to Select, Organize, and Interpret Data

C. Fostering Iterative Design Practices:

1. Decompose Real-World Problems
2. Plan and Design Artifacts.
3. Create, Communicate, and Document Solutions.

D. Fostering Digital Literacy Practices:

1. Responsible Use Practices
3. Evaluate Resources and Recognize Contributions

8.AP.2 The student will plan and implement algorithms that include sequencing, loops, variables, user input, conditional control structures, functions, and various data types.

a. Describe the concept of input and output of various data types for use in a computer program.
b. Plan an algorithm using plain language, pseudocode, or diagrams.
c. Write and test algorithms expressed using block-based or text-based programming languages.

Understanding the Standard

Students advance their understanding of algorithmic design by planning, constructing, and implementing increasingly complex programming solutions. This developmental progression involves the integration of foundational and intermediate programming constructs including control flow mechanisms (sequencing, selection, and iteration), variable manipulation, user input handling, modular decomposition through functions, and the use of diverse data types (e.g., Boolean, integer, floating-point, and string). As students develop programs in either block-based or text-based programming environments, they deepen computational fluency which fosters the development of problem-solving strategies, logic construction, and a conceptual understanding of abstraction, decomposition, and algorithmic efficiency

Algorithm implementation is the process of translating a planned solution into a programming language. This involves encoding the algorithm using constructs such as variables, conditionals, loops, and functions. The result is a set of executable instructions that a computer can interpret and perform.

- Sequence is the specific order in which instructions are executed in a program. The flow of control refers to this order, and incorrect sequencing can affect whether the program runs properly.
- Loops are the repeated execution of a block of code until a certain condition is met (e.g., "for" loops or "while" loops). By repeating commands, programmers write fewer lines of code and reduce the chances of making errors.
- Variables are placeholders in a program that store and represent data values (e.g., numbers or text). Effective variable use makes the problem-solving process easier and faster.
- User input refers to data provided by the user of the program, such as entering text (e.g., a username) or numbers (e.g., selecting a quantity for purchase).
- Output refers to the data or information a program produces and presents to the user based on its processing. An example of output would be if a user inputs a number, the program might output "even" if the number is divisible by 2 or "odd" otherwise.
- Conditional control structures are structures (e.g., "if-else" statements) analyze variables and control whether certain commands are run based on conditions. They act like doors in a program—if the condition is true, the commands are run; if false, they are skipped. Compound conditionals allow for two or more conditions to be tested in a single statement, such as “if-and” and “if-or.”
- Functions are blocks of code that can be reused, designed to perform a specific task.
- Block-based programming uses visual blocks to represent code.

- Text-based programming involves the translation of an algorithm into code using a programming language's syntax.

[8.AP.2a] Programs are collections of code organized in algorithms that can accomplish a variety of tasks. Programs can be developed to perform calculations, manipulate data, or to be creative. Programs that collect and use numeric and text data from users rely on core concepts of input handling, data typing, and data manipulation. Input refers to information provided by the user during program execution, typically received as text and stored for further use. A key conceptual distinction exists between text data (strings) and numeric data (integers or floating-point numbers), with appropriate data types determining how the information can be processed. These processes require a foundational understanding of variables, data flow, and the structural design of programs that manage user interaction and input-driven behavior.

In computer science, data types define the kind of data that can be stored and manipulated within a program. Three fundamental data types are Boolean, integer, and decimal (also known as floating-point).

- Boolean data types represent binary states and hold only two possible values: true or false. In most programming languages, these are often implemented using binary representations, where 1 denotes true and 0 denotes false. Boolean variables are commonly used in conditional statements and logic operations, such as determining whether a condition is met or controlling the flow of a program (e.g., if statements, loops).
- Integer data types represent whole numbers, both positive and negative, without any fractional component. They are typically used in programs where precise counting, indexing, or discrete numeric values are required, for example, tracking the number of iterations in a loop or storing a user's age.
- Decimal data types, also referred to as floating-point numbers, are used to represent numbers that include a fractional component (e.g., 3.14, -0.75). These are essential in situations that require greater precision, such as scientific calculations, financial data processing, or measurement-based applications.

Input and output (I/O) are fundamental components of program execution, enabling interaction between the user and the computer. Input refers to the process by which a program receives data typically from a user, sensor, file, or other external source for processing. Output is the result produced by the program, delivered through a screen, file, or other interface, to communicate information back to the user or system.

Programs must manage data types appropriately during I/O operations. For example, when a program accepts user input, it may initially interpret the user input as a string and may require conversion to a numeric data type for computation.

[8.AP.2b] Planning in programming begins with the creation of a design document. Students generate and organize ideas and algorithms using methods such as outlining, brainstorming, or diagramming. Design documents may include plain language, pseudocode, or visual representations to define program structure.

Drafting involves composing an initial version of the algorithm, anticipating errors and adjustments. Revising includes adding, removing, or rearranging elements to improve sequence and logic. Editing focuses on clarity, correctness, and alignment of variables and control structures with the intended function of the algorithm.

Plain Language	Pseudocode	Diagram
Writing structured, step-by-step instructions in everyday language, incorporating conditions and user input.	Creating more detailed pseudocode with logical conditions, loops, and clear organization.	Developing flowcharts or visual models that illustrate decision-making processes and loops in an algorithm.
Example: Ask the user for a number. If the number is divisible by 2, print 'even.' Otherwise, print 'odd.	Example: Start Input number If number % 2 == 0 Print "even" Else Print "odd" End	Example: Start → Input number → Is number divisible by 2? → Yes: Print 'even' / No: Print 'odd' → End

[8.AP.2c] Translate algorithmic thinking into executable code by mapping logical steps into programming constructs. Algorithmic thinking involves defining the sequence, conditions, and repetition required to solve a problem. Translating this into a program requires selecting appropriate control structures, managing data through variables and data types, and applying logical and Boolean expressions to guide program flow. Programs incorporate previously learned constructs, including conditionals, nested conditionals, loops, nested loops, user input handling, and data type conversions to implement algorithmic logic accurately. Refer to Grade 6 and Grade 7.

Concepts and Connections

CONCEPTS

Programming is the process of creating a set of instructions that a computer executes to perform specific tasks. It begins by defining the data a program will receive (inputs), how that data will be processed using logical operations and control structures, and the expected results (outputs). Programmers use algorithms to map out the program's logic using tools such as pseudocode or flowcharts, then translate those algorithms into a programming language either block-based or text-based to build, test, and refine the program for accuracy and efficiency.

CONNECTIONS

Within the grade level/course: At this grade level, students plan, write, and test algorithms, utilizing computational thinking skills like abstraction and decomposition. These skills align with organizing ideas in writing, analyzing resources across regions, generating algebraic

expressions, and describing motion in science. By integrating these practices, students enhance their ability to create structured solutions applicable across disciplines while developing logical reasoning and critical thinking. (8.AP.2)
Vertical Progression: In Grade 6, In Grade 7, students implement algorithms with sequencing, loops, variables, user input, conditional control structures and functions. (7.AP.2)

Across Content Areas

English

- **8.W.2.A** The student will generate and organize ideas using the writing process (planning, drafting, revising, editing) to develop multi-paragraph texts.

Mathematics

- **8.PFA.1** The student will represent, simplify, and generate equivalent algebraic expressions in one variable.

Science

- **8.1** The student will demonstrate an understanding of scientific and engineering practice. *8.1 standard is integrated within science content and not taught in isolation. Potential science concepts to apply 8.1 include: PS.8.a: The student will investigate and understand that work, force, and motion are related by describing motion using position and time.*

Digital Learning Integration

- **6-8.ID** Students use a variety of technologies, including assistive technologies, within a design process to identify and solve problems by creating new, useful or imaginative solutions or iterations.
  A. In collaboration with an educator, students use appropriate technologies in a design process to generate ideas, create innovative products, or solve authentic problems.
- **6-8.CT** Students develop and employ strategies for understanding and solving problems in ways that leverage the power of technological methods, including those that leverage assistive technologies, to develop and test solutions.
  D. Students demonstrate an understanding of how automation works and use algorithmic thinking to design and automate solutions.

Opportunities for Integration

Curriculum integration strengthens conceptual understanding and skill application. This can be done through multidisciplinary, interdisciplinary, and transdisciplinary approaches to integration. The examples below illustrate multiple ways to integrate computer science.

English

- Students use pseudocode or flowcharts to outline their writing process before drafting a multi-paragraph text. By breaking down the steps of planning, drafting, revising, and editing into a structured sequence, students apply decomposition and algorithmic thinking to organize and refine their ideas. Each stage of the process includes defining specific inputs such as research notes, writing prompts, or peer feedback and corresponding outputs like topic sentences, paragraph drafts, or edited revisions.

History and Social Science

- Students design a program or flowchart that simulates the distribution of natural resources across different world regions. Using input variables such as climate, geography, and population density, the program will apply sequencing, selection, and conditional logic to generate output predicting resource availability. Students will demonstrate abstraction by identifying key factors, decomposition to break the system into manageable components, and algorithmic thinking to design a rule-based model.

Mathematics

- Students can write a program using block-based coding that takes an algebraic expression as input and simplifies it step by step using programmed rules. This task allows them to apply sequencing, conditional control structures, and iteration to model the logic behind algebraic simplification. Students will also apply abstraction by identifying key simplification rules (such as combining like terms or applying the distributive property) and engage in debugging as they test and revise their program to ensure accurate outputs.

Science

- Students will create a flowchart or decision tree to model the motion of an object over time using position and speed as primary variables. By applying a step-by-step structure, they will incorporate decomposition to break down motion into stages and use pattern recognition to identify how changes in force, mass, or time intervals influence an object's movement. The visual representation should include input variables (e.g., initial speed, applied force, time duration) and output outcomes (e.g., acceleration, displacement).

Skills in Practice

Students should engage in the following practices to deepen their conceptual understanding and enhance the application of skills aligned with the Computer Science *Standards of Learning*. These practices are explained in more detail in Appendix A.

A. Fostering Collaborative Computing Practices:

3. Create and Accept Feedback
4. Select and Use Collaboration Tools

B. Fostering Computational Thinking Practices:

1. Decompose Real-World Problems
2. Explore Common Features and Extract Patterns
3. Apply Abstraction to Simplify, Represent, and Problem Solve
4. Apply Algorithmic Thinking to Problem Solve and Create

C. Fostering Iterative Design Practices:

1. Identify, Define, and Evaluate Real-World Problems
2. Plan and Design Artifacts
3. Create, Communicate and Document Solutions
4. Test and Optimize Artifacts

D. Fostering Digital Literacy Practices:

1. Responsible Use Practices
3. Evaluate Resources and Recognize Contributions

8.AP.3 The student will use the iterative design process to create, test, and debug programs using a block-based or text-based programming language.

a. Create and test programs that contain multiple control structures.
b. Trace and predict outcomes of programs.
c. Analyze the outcomes of programs to identify logic and syntax errors.
d. Analyze and describe the results of a program to assess validity of outcomes.
e. Revise and improve an algorithm to resolve errors or produce desired outcomes.

Understanding the Standard

The iterative design process is a systematic method used to develop and refine products or solutions through repeated cycles of planning, implementation, evaluation, and revision. In programming, this process includes designing algorithms, writing code, testing functionality, identifying errors, and making improvements. Each cycle builds upon previous iterations to improve accuracy, efficiency, or functionality. This development method supports continuous refinement rather than finalizing solutions in a single step. Iterative programming allows for adjustments based on observed behavior, test results, or updated requirements. Structured repetition reinforces computational thinking and highlights the evolving nature of program development.

[8.AP.3a] Create and test programs that include multiple control structures such as loops, conditionals, and functions. These control structures define program behavior by managing repetition, decision-making, and modular execution. Integrating multiple control structures supports complex logic and interactive functionality. Testing these programs verifies that the combined structures execute as intended, identify logic or syntax errors, and confirm that program output aligns with expected results.

[8.AP.3b] Tracing is the process of reviewing an algorithm or program step by step to observe how variable values change during execution. This involves analyzing control flow and logic structures to determine how data is processed at each stage. Tracing may be done manually or with debugging tools to identify variable states, output values, and execution paths. Reviewing variable changes supports verification of correctness and identification of inconsistencies in the program's behavior.

A recommended approach in tracing the flow of an algorithm or program includes the following:

1. Read the entire program to understand its overall structure and purpose.
2. Identify the variables used and note their initial values.
3. Start at the first line of code and move line by line in order the program runs.
4. Record the value of each variable as it changes after every line of execution.
5. Follow control structures like conditionals (e.g., *if/else*) and loops, and note which path or iteration is taken based on the conditions.

6. Predict the program's output based on the variable values and logic you've traced.
7. Compare traced output to the actual result when the program runs to check for accuracy and identify any logic or syntax errors.

[8.AP.3c] Evaluating a program's results involves checking the output against expected values, verifying logical flow, and identifying errors or unintended behavior. This includes testing different inputs, tracing variable changes, and confirming that the program meets defined specifications and functional requirements.This evaluation informs code refinements that improve accuracy, correctness, and alignment with intended objectives.

- Syntax defines the structural rules of a programming language, specifying how code must be written for successful interpretation and execution. A syntax error occurs when code violates these structural rules, causing the program to fail during compilation or runtime. Common syntax errors include missing punctuation such as commas or parentheses, incorrect use of keywords, and improper command formatting.

  Text-based programming languages are more prone to syntax errors due to the manual entry of code. In contrast, block-based programming environments minimize syntax errors by using pre-structured visual blocks that only connect in syntactically correct ways. This reduces the likelihood of mistakes related to grammar and formatting, allowing learners to focus on logic and structure.

    - Consider the following example: A syntax error would be forgetting to close a parenthesis in a function.

- Logic defines the reasoning and decision-making processes within a program, guiding the flow of execution through the use of conditionals, comparisons, and Boolean expressions. Logical structures determine which actions a program takes based on specific criteria or input conditions.

  A logic error occurs when a program runs without interruption but produces incorrect or unintended output due to flawed reasoning or mis-ordered control flow. Unlike syntax errors, logic errors do not generate explicit error messages and must be identified by reviewing the program's behavior, tracing variable changes, and analyzing control structures to locate the incorrect logic.

    - Consider the following example:A logic error might be adding numbers incorrectly in a loop, causing the wrong total to be calculated.

[8.AP.3d] After a program executes, the output must be evaluated to determine whether it aligns with expected results. This evaluation involves verifying program correctness and identifying deviations such as incorrect variable updates, unmet conditions, or logic errors. When an algorithm does not produce the intended result, a systematic review process is used to identify and resolve faults. This includes tracing

execution flow, analyzing each logical step, and comparing actual outputs to expected values. Tools such as debugging environments, print statements, or manual walkthroughs support error detection and correction.

Common modifications include correcting variable assignments, revising control structures such as loops or conditionals, and refining logic expressions. These changes are made to restore functional accuracy and align the algorithm's behavior with its original design specifications.

Following revisions, the algorithm is retested to verify that the issue has been resolved. This process reinforces the iterative nature of software development, where repeated testing, analysis, and refinement are used to improve performance and reliability.

- For example, in a game design project, students may analyze how a score variable changes based on player actions within a loop and evaluate how conditional statements impact the game's difficulty. By examining these results, students determine whether the scoring logic aligns with the intended game mechanics, verify that conditions trigger the correct responses, and make adjustments to improve gameplay balance. This process reinforces the importance of reviewing program behavior to ensure logical consistency and user engagement.

[8.AP.3e] Revising and improving an algorithm is a fundamental part of the debugging process, which involves identifying and correcting errors that prevent a program from functioning as intended. Debugging focuses on isolating logical or syntactical faults, commonly referred to as "bugs", within a program's code or structure.

Debugging is the process of identifying and correcting errors within a program. It involves tracing variable states, monitoring control flow, and analyzing outputs at specific execution points. Built-in debugging tools, breakpoints, and output statements are used to locate faults in logic, syntax, or runtime behavior.

Effective debugging isolates the cause of an error and supports targeted revisions. It contributes to functional accuracy, consistency with program specifications, and overall program stability. Debugging is a core component of iterative development and is repeated throughout the programming cycle to ensure the final solution operates as intended. Consider the following example checklist for debugging a program:

1. Syntax Errors
    - Are there any typos (spelling mistakes, incorrect keywords)?
    - Are parentheses, braces, and quotes properly placed?
    - Is there correct indentation or block structure?
2. Logic Errors
    - Does the program do what I expect? (i.e. correct math? right output?)
    - Are the if condition (if-statements) correct?
    - Are loops running too many or too few times?

3. Variables
    - Is the variable name spelled correctly?
    - Is the variable being used consistently throughout the code?
    - What is stored in the variable?
    - Is it the correct type of value?

Concepts and Connections

CONCEPTS

The program development process follows an iterative design approach, where solutions are continuously planned, implemented, tested, and refined. Programs incorporate multiple control structures to direct logic and control execution flow. Developers trace code to predict behavior, analyze results to detect syntax and logic errors, and validate outputs against expected results. Iterative refinement of algorithms and code is essential for enhancing program efficiency, resolving issues, and ensuring the solution aligns with functional requirements.

CONNECTIONS

Within the grade level/course: At this grade level, students use iterative testing and debugging to improve computational programs, mirroring revision and refinement processes across disciplines. In history, they analyze geographic tools to refine decision-making strategies. In English, they revise writing for clarity and coherence. In math, they apply logical reasoning and validation techniques to solve problems, while in science, they test and refine experimental procedures to ensure accurate results. By integrating these practices, students strengthen their ability to evaluate, revise, and optimize solutions in both computational and real-world contexts.

Vertical Progression: In Grade 7 students used the iterative process to create, test and debug programs with a goal of identifying logic and syntax errors and resolving them to produce desired outcomes. (7.AP.3).

ACROSS CONTENT AREAS

English

- **8.W.3.A** The student will revise writing for clarity of content, word choice, sentence variety, and transition among paragraphs.

History and Social Science

- **WG.1** The student will apply history and social science skills to explain how geographic information and geospatial tools are used to make decisions.

Mathematics

- **8.PS.1** The student will use statistical investigation to determine the probability of independent and dependent events, including those in context.

DIGITAL LEARNING INTEGRATION

- **6-8.ID** Students use a variety of technologies, including assistive technologies, within a design process to identify and solve problems by creating new, useful or imaginative solutions or iterations.
  C. In collaboration with an educator, students use appropriate technologies in a cyclical design process to develop prototypes and demonstrate the use of setbacks as potential opportunities for improvement.

Opportunities for Integration

Curriculum integration strengthens conceptual understanding and skill application. This can be done through multidisciplinary, interdisciplinary, and transdisciplinary approaches to integration. The examples below illustrate multiple ways to integrate computer science.

English

- Students use pseudocode or flowcharts to outline their writing process before drafting a multi-paragraph text. By breaking down the stages of planning, drafting, revising, and editing into a structured sequence, students apply decomposition and algorithmic thinking to organize their ideas. They will also practice tracing by stepping through each part of their flowchart to ensure logical consistency and completeness. As they translate their flowchart into written output, students reflect on potential syntax errors such as misplaced transitions, unclear topic sentences, or inconsistent formatting and revise accordingly.

History and Social Science

- Students create a simple program or flowchart that models the distribution of natural resources across different world regions using mathematical reasoning and computational logic. The program will take user input on variables such as climate, geography, and population, and apply sequencing, conditional logic, and basic mathematical operations (e.g., ratios, comparisons, or proportional reasoning) to generate output that predicts resource availability. Students will use quantitative data to build algorithms that simulate decision-making based on numerical patterns, enhancing their understanding of math concepts like data representation, input/output relationships, and prediction modeling within real-world environmental and economic contexts.

Skills in Practice

Students should engage in the following practices to deepen their conceptual understanding and enhance the application of skills aligned with the Computer Science *Standards of Learning*. These practices are explained in more detail in Appendix A.

C. Fostering Iterative Design Practices:

1. Identify, Define, and Evaluate Real-World Problems:
2. Plan and Design Artifacts
3. Create, Communicate and Document Solutions
4. Test and Optimize Artifacts

8.AP.4 The student will incorporate work from others into programs and projects.

a. Explain the role of Creative Commons licensing for the use and modification or "remixing" of information.
b. Utilize Creative Commons assets in a programming project.
c. Use and remix code from other projects within a programming project and provide proper attribution.

Understanding the Standard

Programs may integrate existing algorithms, code segments, libraries, and media assets to optimize development time and improve functionality. Incorporating pre-existing components allows developers to utilize proven solutions for common computational tasks, reducing redundancy and supporting modular, scalable design. These resources may include pre-written functions, reusable modules, or third-party packages that perform specific operations such as sorting, encryption, or user interface rendering.

Code libraries provide structured collections of functions and classes that abstract complex processes into simpler calls. For example, libraries for data visualization, machine learning, or graphics rendering enable developers to implement advanced features without writing low-level code. Code snippets may also be reused to implement frequently used patterns or algorithms, increasing consistency and reducing the likelihood of introducing errors.

Media assets, such as images, audio files, or animation sequences, are commonly imported to enhance user interface or user experience components. The integration of these digital assets into programs requires appropriate referencing, licensing compliance, and optimization for system performance. Responsible use of shared digital resources promotes digital citizenship by encouraging respect for intellectual property, collaboration, and legal compliance.

[8.AP.4a] Incorporating external work into programs and projects involves using or modifying assets created by others while providing attribution. Creative Commons (CC) licenses are standardized legal tools that specify the conditions under which creative content may be used, adapted, and shared. These licenses define permitted uses such as commercial reuse, modification, and redistribution, and may include requirements such as attribution, share-alike provisions, or non-commercial use restrictions. Use of CC-licensed media in programming or digital projects requires adherence to the terms defined by the license type. This ensures compliance with intellectual property requirements and supports the lawful integration of external resources.

[8.AP.4b] Creative Commons assets are materials made available under standardized licenses that specify conditions for reuse, modification, and distribution. To utilize these assets in programming, users must identify the applicable license type, verify permissions for use or adaptation, and include proper attribution according to license terms.Understand the license terms: This includes knowing if attribution is required, whether modifications are permitted, and if derivative works must carry the same license. CC examples below:

- **CC BY**: Requires attribution.

- **CC BY-NC**: Requires attribution but prohibits commercial use.
- **CC BY-SA**: Requires attribution and that any derivative works use the same license.
- **CC0**: Public domain—no attribution required but still encouraged.

Proper attribution is a legal and ethical requirement that acknowledges the original creator of an external resource. This may involve including license text in project documentation, referencing the author in code comments, or displaying credit within the user interface. Attribution ensures compliance with usage terms and supports transparency in the development process. In programming projects, attribution is often provided through comments within the code, a README file, or a designated credits section.

Key Steps to Proper Attribution:

- Identify the source: Ensure students know where their materials come from this includes the website, author, or source URL.
- Use correct citation format: Just like citing a book or article, students should include key details such as the author's name, the title of the work, and the URL.
- Place the attribution properly: In coding, attribution often happens in comments within the code or in a designated "Credits" section of the program.

[8.AP.4c] Using and remixing code from other projects refers to the practice of incorporating existing code into new programs and modifying it as needed to meet specific functional requirements. This process involves identifying reusable components, integrating them into the new codebase, and adapting logic or structure as necessary. Proper attribution is required to acknowledge original authorship and comply with licensing conditions, including those defined by Creative Commons or open-source licenses.

The following illustrate how Creative Commons (CC) licensing may work:

- A student is developing an educational website. The student finds a photograph online that is licensed under Creative Commons Attribution 4.0 (CC BY 4.0). This license permits the use, adaptation, and redistribution of the image including for commercial and educational purposes provided that proper attribution is given to the original creator. Proper attribution would typically include the creator's name, the title of the work (if available), a link to the source, and a reference to the CC BY 4.0 license.

By including this attribution, the student acknowledges the original creator and specifies the license type, ensuring compliance with the CC license terms while respecting the creator's work.

Concepts and Connections

CONCEPTS

Creative Commons licensing establishes guidelines for how digital content can be used, modified, and shared while requiring attribution to the original creator. In programming projects, this includes incorporating or adapting code, media, or other digital assets created by others in compliance with the license terms. Using others' work responsibly ensures legal and ethical reuse, supports collaboration, and reinforces respect for intellectual property through proper attribution.

CONNECTIONS

Within the grade level/course: At this grade level, students incorporate external works and code snippets while adhering to legal and ethical standards by properly attributing the original author and source. (8.AP.4)
Vertical Progression: In Grade 7, students apply proper methods of attribution when incorporating information from the internet or other sources into their programs. (7.AP.4)

ACROSS CONTENT AREAS

English

- **8.R.1** Evaluation and Synthesis of Information: Formulate appropriately narrow questions about a research topic and refocus the inquiry when appropriate.

History and Social Science

- **CE.13** The student will apply social science skills to understand the role of government in the United States economy by e) describing how governments regulate to protect consumers, labor, the environment, competition in the marketplace, and property rights.

Opportunities for Computer Science Integration

Curriculum integration strengthens conceptual understanding and skill application. This can be done through multidisciplinary, interdisciplinary, and transdisciplinary approaches to integration. The examples below illustrate multiple ways to integrate computer science.

English

- Students will analyze, remix, and ethically re-use existing content under Creative Commons licenses to develop original written pieces.

History and Social Science

- Students explore how Creative Commons licenses contribute to the sharing of historical and civic information in digital spaces. They will examine primary sources and public domain materials to understand how licensing affects access, remixing, and the preservation of cultural heritage. Students design and organize a virtual exhibit using CC-licensed content, learning how ethical citation empowers knowledge-sharing.

Skills in Practice

Students should engage in the following practices to deepen their conceptual understanding and enhance the application of skills aligned with the Computer Science *Standards of Learning*. These practices are explained in more detail in Appendix A.

A. Fostering Collaborative Computing Practices:

3. Create and Accept Feedback

B. Fostering Computational Thinking Practices:

3. Apply Abstraction to Simplify, Represent, and Problem Solves

C. Fostering Iterative Design Practices:

3. Create, Communicate, and Document Solutions

D. Fostering digital Literacy Practices:

1. Responsible Use Practices
3. Evaluate Resources and Recognize Contributions

Computing Systems (CSY)

8.CSY.1 The student will recommend and design improvements to computing devices based on the needs of various users.

a. Analyze existing computing devices for advantages and limitations.
b. Recommend and design improvements to computing devices based on user interactions.

Understanding the Standard

[8.CSY.1a] Computing devices perform a wide range of functions, including communication, data processing, information retrieval, content creation, and automation. These devices include desktops, laptops, tablets, and smartphones, and support use across personal, academic, and professional settings. Advancements in device architecture have increased processing speed, memory capacity, portability, and display resolution. Wireless connectivity, integrated sensors, and high-performance hardware enable real-time collaboration, mobile computing, and interactive applications. Assistive technologies integrated into computing devices expand accessibility for users with disabilities.

Computing devices rely on continuous power and stable internet access for optimal functionality. Limitations include physical health risks associated with prolonged use, such as eye strain and musculoskeletal issues. Device performance may be reduced in low-bandwidth or offline environments. Security vulnerabilities, such as malware, phishing, and unauthorized access, threaten data integrity and user privacy. Increased reliance on digital systems contributes to electronic waste and environmental strain. Disparities in access to devices and connectivity contribute to the digital divide, impacting education, employment, and social participation. Device complexity may also present challenges for users with limited technical proficiency.

[8.CSY.1b] Optimization of computing devices involves the analysis of human-computer interaction (HCI), which examines user engagement with system interfaces, input mechanisms, and output responses. This process identifies usability issues, interaction inefficiencies, and patterns of behavior that affect user performance. HCI analysis incorporates studies of interface navigation, response time, cognitive load, and accessibility features. Evaluation methods include user surveys, observational data, and task-based usability testing. These findings are used to refine interface elements, system design, and functionality. The goal is to align computing device performance with user requirements through iterative design based on empirical data. Some areas in which design improvements might be considered include, but are not limited to:

- User Interface is the part of a computer system that a user interacts with. It includes all the visual elements like windows, menus, icons, and buttons, as well as the non-visual elements like sounds or haptic feedback. A good UI is intuitive, easy to use, and visually appealing.
- Accessibility refers to the design and inclusion of features in technology that make it usable by people with disabilities. This includes features like screen readers for the visually impaired, voice control for those with motor impairments, and keyboard navigation for those who cannot use a mouse.

- Security and Privacy involves protecting systems and data from unauthorized access, while privacy focuses on protecting personal information. Strong passwords, firewalls, and antivirus software are examples of security measures.
- Battery life refers to the duration a device can operate on a single charge
- Performance relates to how quickly and efficiently a device can process tasks. A balance between the two is often sought, as high-performance tasks can drain battery life faster.
- Ergonomics is the study of designing equipment and furniture to fit the human body and its capabilities. In computing, it involves designing devices and workspaces to reduce the risk of musculoskeletal disorders. This includes factors like chair height, monitor placement, and keyboard and mouse position.

Concepts and Connections

Concepts

Computing device analysis examines the strengths and limitations of a device's design and performance to assess its effectiveness. The process includes identifying areas for improvement and proposing enhancements aligned with user needs, functionality requirements, and system capabilities.

Connections

Within the grade level/course: At this grade level, students examine how computing devices address user needs by analyzing usability vs. security trade-offs and identifying internet threats and vulnerabilities. They develop solutions that build digital literacy and foundational skills for protecting data and systems. (8.CSY.1)

Vertical Progression: In Grade 7, students design projects that collect and exchange data and evaluate the effectiveness of hardware and software components. (7.CSY.1)

Across Content Areas

English

- **7.DSR** The student will build knowledge and comprehension skills from reading a range of challenging, content-rich texts. This includes fluently reading and gathering evidence from grade-level complex texts, reading widely on topics to gain purposeful knowledge and vocabulary, and using reading strategies when comprehension breaks down.

Opportunities for Computer Science Integration

Curriculum integration strengthens conceptual understanding and skill application. This can be done through multidisciplinary, interdisciplinary, and transdisciplinary approaches to integration. The examples below illustrate multiple ways to integrate computer science.

English

- Students craft a persuasive essay or presentation that recommends thoughtful design improvements to a computing device, demonstrating their command of argumentative writing and digital literacy. They will evaluate real-world user needs and analyze the strengths and shortcomings of existing device features, constructing a clear argument supported by evidence from research and user feedback.

History and Social Science

- Students examine and analyze the advantages and limitations of early computing devices, such as punch card machines, typewriters, and early microprocessors, to better understand user needs and social context.

Skills in Practice

Students should engage in the following practices to deepen their conceptual understanding and enhance the application of skills aligned with the Computer Science *Standards of Learning*. These practices are explained in more detail in Appendix A.

A. Fostering Collaborative Computing Practices:

2. Include Multiple Perspectives
3. Create and Accept Feedback

B. Fostering Computational Thinking Practices:

1. Decompose Relevant Problems
2. Explore Common Features and Extract Patterns
3. Apply Abstraction to Simplify, Represent, and Problem Solve

C. Fostering Iterative Design Practices:

1. Identify, Define, and Evaluate Real-World Problems
2. Plan and Design Artifacts
3. Create, Communicate, and Document Solutions

D. Fostering Digital Literacy Practices:

1. Responsible Use Practices
3. Evaluate Resources and Recognize Contributions

8.CSY.2 The student will apply computational thinking to troubleshoot and document hardware and software-related problems.

a. Apply systematic processes to resolve hardware, software, and connectivity-related problems.
b. Design an end-user document/guide to resolve hardware, software, and connectivity-related problems.

Understanding the Standard

[8.CSY.2a] Troubleshooting is a systematic process used to identify and resolve issues in hardware, software, or network connectivity. It involves following logical steps to isolate the cause of a problem and apply targeted solutions. For hardware, issues may include system crashes, unresponsive devices, or degraded performance. Software problems often present as application errors, crashes, or abnormal behavior. Connectivity issues can result in slow speeds, intermittent access, or complete loss of network connection. A structured approach such as verifying settings, restarting components, and checking for updates enables efficient diagnosis and resolution. For example, resolving a Wi-Fi issue may involve confirming network settings, restarting the router, and ensuring the device is not in airplane mode. Applying consistent troubleshooting protocols increases reliability and minimizes downtime in computing systems

Example: A systematic process to diagnosing hardware issues may consist of the following steps:

Verify power and device activation	Inspect physical connections and peripherals	Check basic settings	Listen for diagnostic sounds
• Ensure the device is plugged into a functioning power source. • Confirm that the power switch is turned on. • Check that the display or monitor is powered and active.	• Ensure all cables are securely connected. • Disconnect unnecessary external devices to isolate the issue. • For portable devices, confirm battery charge or power adapter functionality.	• Perform a complete system reboot to reset hardware components and resolve temporary issues.	• Observe any startup beeps, fan noise, or drive activity that could indicate hardware function or failure.

Example: A systemic process for diagnosing and resolving software-related issues may include the following steps:

Identify the problem	Restart the application	Check for software updates	Test software on another device

• Determine the specific software or application experiencing issues. • Record any visible error messages, codes, or unexpected behavior to support further diagnosis.	• Close and relaunch the software to clear temporary glitches or memory conflicts.	• Ensure the application and operating system are updated to the latest version to resolve known bugs or compatibility issues.	• Run the same application on a different device to determine whether the issue is device-specific or related to the software itself.

Example: A systematic approach to diagnosing network or internet connectivity issues may include the following steps:

Verify Wi-Fi connection status	Check Airplane Mode and wireless settings	Test access to online content	Disconnect and reconnect to the network
• Ensure the device is connected to the intended Wi-Fi network. • Reconnect to the network if the connection is lost or unstable.	• Confirm that Airplane Mode is turned off and that Wi-Fi is enabled. • On some devices, ensure mobile data is enabled if Wi-Fi is not in use.	• Try loading multiple websites or using different internet-based apps to determine if the issue is isolated to one service or more widespread.	• Manually disconnect from the Wi-Fi network and reconnect to initiate a fresh session, which can resolve authentication or IP assignment issues.

[8.CSY.2b] Documenting the troubleshooting process includes recording the observed issue, the sequential actions taken, and the resolution achieved. This documentation creates a reference for recurring problems and supports the development of end-user guides or knowledge bases. Troubleshooting hardware, software, and connectivity issues involves structured procedures based on conditional logic, often using if/then frameworks. Steps are performed in a logical order, where each action depends on the result of the previous condition. Common issues include hardware failures (e.g., power supply problems, peripheral malfunctions), software errors (e.g., application crashes, update conflicts), and network disruptions (e.g., unstable Wi-Fi, slow connections). A systematic approach ensures accurate diagnostics and efficient resolution.

End-user documentation provides structured formats for conveying troubleshooting procedures and solutions. Examples include flowcharts, checklists, frequently asked questions (FAQs), and step-by-step instructions. These resources support repeatability, standardize responses to common issues, and promote user independence in educational and organizational environments.To design an effective end-user guide, a student must:

- Understand the common hardware, software, and connectivity issues users may face.
- Organize content logically, often following a decision-tree or if/then structure to mirror how troubleshooting typically unfolds.
- Use precise terminology and ensure clarity and readability, especially for non-technical audiences.
- Include visuals or labeled screenshots, where applicable, to support different learning preferences.
- Reinforce the importance of documenting problems and solutions during troubleshooting, so users can apply or adapt previous solutions to future issues.

Concepts and Connections

Concepts

Hardware, software, and connectivity issues are addressed using systematic troubleshooting methods to ensure accurate and efficient problem resolution. End-user documentation supports this process by offering structured, step-by-step guidance that helps users independently identify and resolve technical problems.

Connections

Within the grade level/course: At this grade level, students apply computational thinking to troubleshoot common problems with the goal of designing end user documents and guides for resolving issues with hardware, software and networks. (8.CSY.2)
Vertical Progression: In Grade 7, students apply computational thinking and troubleshooting skills to systemically process and resolve problems. They compile and record all successful methods used to identify common hardware, software and connectivity related problems. (7.CSY.2)

Opportunities for Computer Science Integration

Curriculum integration strengthens conceptual understanding and skill application. This can be done through multidisciplinary, interdisciplinary, and transdisciplinary approaches to integration. The examples below illustrate multiple ways to integrate computer science.

English

- Students develop a technical manual or troubleshooting guide that addresses common hardware, software, or connectivity issues using precise domain-specific language and structured procedural writing. They will craft clear, step-by-step instructions tailored to a specific audience, emphasizing accurate terminology (e.g., “driver conflict,” “network latency,” “firmware update”) and effective communication strategies.

History and Social Science

- Students investigate how governments and organizations respond to technological failures by analyzing real-world case studies, such as major cybersecurity breaches or infrastructure breakdowns. Students then create a historical comparison report explaining how systematic troubleshooting processes, similar to those used in computing, helped resolve these crises.

Skills in Practice

Students should engage in the following practices to deepen their conceptual understanding and enhance the application of skills aligned with the Computer Science *Standards of Learning*. These practices are explained in more detail in Appendix A.

B. Fostering Computational Thinking Practices:

1. Decompose Real-World Problems
2. Explore Common Features and Extract Patterns
3. Apply Abstraction to Simplify, Represent, and Problem Solve
4. Apply Algorithmic Thinking to Problem Solve and Create
5. Apply Computational Thinking Practices to Select, Organize, and Interpret Data

C. Fostering Iterative Design Practices:

1. Identify, Define, and Evaluate Real-World Problems
2. Plan and Design Artifacts
3. Create, Communicate and Document Solutions
4. Test and Optimize Artifacts

D. Fostering Digital Literacy Practices:

1. Responsible Use Practices
3. Evaluate Resources and Recognize Contributions

Cybersecurity (CYB)

8.CYB.1 The student will investigate and describe ways to protect sensitive data from malware and other attacks.

a. Identify impacts of hacking, ransomware, scams, phishing, fake vulnerability scans and the ethical and legal concerns.
b. Describe how cyber-attacks can affect a computing system.
c. Compare and contrast safe and unsafe computing practices.
d. Explore how industries and emerging technologies are addressing cyber solutions.
e. Model common prevention practices for cyber-attacks.

Understanding the Standard

[8.CYB.1a] Cybersecurity threats such as hacking, ransomware, scams, phishing, and fake vulnerability scans involve unauthorized access to systems, networks, or data. These actions compromise the confidentiality, integrity, and availability of digital resources, often leading to data theft, manipulation, service interruption, or financial loss. When sensitive data such as personal identifiers, financial information, or health records is exposed or exploited, the risks to individuals and organizations increase significantly.

Malware is malicious software designed to harm or exploit devices, systems, or networks. Some examples include:

- Viruses that replicate and spread, damaging files or systems.
- Worms that self-replicate and spread across networks.

Cybersecurity Attacks can come in the form of:

- Hacking is the unauthorized access to computer systems or networks.
- Ransomware is a type of malware that encrypts a victim's files and demands a ransom to decrypt them.
- Scams are deceptive tactics that are used to trick people into revealing personal information or sending money.
- Phishing is a type of scam that involves fraudulent emails or messages designed to steal sensitive information.
- Fake Vulnerability Scans- are malicious software disguised as a security tool to gain unauthorized access to a system.

These threats present ethical concerns related to privacy, consent, and responsible technology use, particularly in situations where individuals are unaware of or unable to control how their data is accessed or shared. Legally, such activities violate cybersecurity and data protection laws. Unauthorized access, extortion through ransomware, and deceptive practices like phishing and fake scans may constitute offenses such as fraud, identity theft, and computer misuse. Violations may result in criminal prosecution, civil liability, and regulatory sanctions. The protection of sensitive data and adherence to cybersecurity laws and ethical standards are critical to maintaining trust, safeguarding systems, and ensuring responsible digital conduct.

[8.CYB.1b] Cyberattacks impact computing systems by compromising confidentiality, integrity, and availability. Unauthorized access to sensitive data, data corruption, and data loss may occur. Ransomware encrypts system files and restricts access. Denial-of-service (DoS) attacks overwhelm network or system resources, causing operational interruptions. These attacks reduce system reliability, expose protected information, and disrupt essential digital services.

- Cyberattack is an attack that targets computing devices, networks, software, or users to disrupt operations, gain unauthorized access, or steal, alter, or destroy data.

[8.CYB.1c] Safe computing practices involve proactive behaviors and technical safeguards that protect the confidentiality, integrity, and availability of information systems. These include the use of strong, unique passwords; enabling multi-factor authentication; regularly updating operating systems and applications; using reputable antivirus and firewall software; and backing up critical data to secure locations. Safe practices also extend to physical security, such as restricting physical access to devices, securing hardware in controlled environments, and using screen locks when devices are unattended.

Unsafe computing practices include the use of weak or reused passwords, failure to apply security patches and updates, disabling security features, clicking on unverified links or attachments, and using unsecured public networks for sensitive transactions. Unsafe physical practices such as leaving devices unattended in public spaces or sharing access credentials can increase exposure to threats. Comparing these approaches highlights that safe computing involves both digital and physical measures to reduce vulnerabilities, while unsafe practices heighten the risk of unauthorized access, data breaches, and system compromise.

[8.CYB.1d] Industries and emerging technologies continue to evolve in response to the increasing complexity and frequency of cyberattacks. Artificial Intelligence (AI) and Machine Learning (ML) are central to this advancement, offering enhanced capabilities in threat detection, real-time response automation, and predictive analytics. These technologies support the identification of anomalous behavior, enable dynamic risk assessment, and contribute to the development of adaptive cybersecurity frameworks capable of addressing both known and novel threats.

- Artificial intelligence (AI) is a branch of computer science focused on creating algorithms and models that enable machines to perform tasks that typically require human intelligence. These tasks include learning from data, recognizing patterns, understanding language, solving problems, and making decisions. AI methods are embedded in a wide range of systems from standalone software applications to complex hardware environments, allowing them to operate autonomously, adapt over time, and support real-world functionality across disciplines and industries.

- Machine learning is a computational method used to develop algorithms and models that identify patterns in training data and apply those patterns to make predictions, classify data, or automate decision-making.

- Training data is the dataset used to build and refine machine learning models. It consists of labeled or unlabeled examples that enable the model to learn patterns, make predictions, or perform specific tasks.

Emerging technologies continue to reshape the cybersecurity landscape by introducing innovative solutions for data protection and threat mitigation. Blockchain technology, with its decentralized architecture and cryptographic security, offers robust mechanisms for ensuring data integrity, transparency, and privacy across distributed systems.

Quantum computing, though still in the developmental phase, presents the potential to transform cybersecurity through the advancement of quantum encryption techniques and enhanced threat detection capabilities.

In parallel, organizations are increasingly adopting Zero Trust security models, which operate on the principle that no user, device, or system—whether internal or external—should be inherently trusted. This model requires continuous authentication, strict access controls, and micro-segmentation of networks to minimize the risk of unauthorized access and data compromise. Together, these technologies and frameworks reflect a shift toward more adaptive, resilient, and proactive cybersecurity strategies.

[8.CYB.1e] Common prevention practices for cyberattacks include:

- Creating and managing strong passwords, including the use of passphrases and multi-factor authentication.
- Recognizing and avoiding phishing attempts, suspicious links, and fraudulent communications.
- Installing and updating antivirus and anti-malware software to detect and block malicious activity.
- Applying regular software and operating system updates to patch known vulnerabilities.
- Securing devices physically and digitally, such as locking screens when unattended and disabling unused ports or services.
- Using secure networks, avoiding public Wi-Fi for sensitive activities, and enabling firewalls.
- Backing up data regularly using secure, redundant storage solutions.
- Practicing responsible digital behavior, such as logging out of accounts, avoiding untrusted downloads, and following institutional security protocols.

Concepts and Connections

CONCEPTS

Cybersecurity is the practice of protecting systems and data from threats such as hacking, ransomware, phishing, and scams. It includes understanding how cyber-attacks compromise computing systems, distinguishing safe and unsafe computing practices, and exploring how industries and emerging technologies implement defense strategies. Preventive measures strengthen cybersecurity by mitigating vulnerabilities and preserving the core cybersecurity principles of confidentiality, integrity, and availability.

CONNECTIONS

Within the grade level/course: At this grade level, students evaluate when and how to share personal information online by weighing the risks and benefits across various situations. (8.CYB.1)

Vertical Progression: In Grade 7, students differentiated physical and digital security measures that protect electronic information. (7.CYB.1)

ACROSS CONTENT AREAS

History and Social Science

- **CE.6.d** The student will examine how civic participation can address community needs and serve the public good, including staying informed about current issues.

DIGITAL LEARNING INTEGRATION

- **6-8.DC** Students recognize the rights, responsibilities and opportunities of living, learning and working in an interconnected digital world, and they act in ways that are safe, legal, and ethical.
  B. Students demonstrate and advocate for positive, safe, legal, and ethical habits (established behaviors) when using technology and interacting with others online.
  D. Students demonstrate an understanding of what personal data is, how data collection technologies work, tradeoffs of sharing personal data, and best practices for keeping it private and secure.

Opportunities for Computer Science Integration

Curriculum integration strengthens conceptual understanding and skill application. This can be done through multidisciplinary, interdisciplinary, and transdisciplinary approaches to integration. The examples below illustrate multiple ways to integrate computer science.

English

Students write a persuasive essay or news article on the importance of cybersecurity awareness. They will argue for stronger digital safety measures, citing real-world examples of hacking, phishing, and ransomware attacks, while providing evidence-based solutions.

History and Social Science

Students research major cybersecurity incidents that have impacted governments, businesses, or individuals, such as data breaches at financial institutions or large-scale ransomware attacks on critical infrastructure. Students analyze how these attacks influenced laws, policies, and security strategies, drawing connections between cybersecurity and global security.

Skills in Practice

Students should engage in the following practices to deepen their conceptual understanding and enhance the application of skills aligned with the Computer Science *Standards of Learning*. These practices are explained in more detail in Appendix A.

A. Fostering Collaborative Computing Practices:

2. Include Multiple Perspectives
3. Create and Accept Feedback

B. Fostering Computational Thinking Practices:

1. Decompose Relevant Problems
2. Explore Common Features and Relationships to Extract Patterns
3. Apply Abstraction to Simplify, Represent, and Problem Solve

C. Fostering Iterative Design Practices:

1. Identify, Define, and Evaluate Real-World Problems

3. Create, Communicate, and Document Solutions

D. Fostering Digital Literacy Practices:

1. Responsible Use Practices
2. Safeguard Well-Being of Self and Others
3. Evaluate Resources and Recognize Contributions

8.CYB.2 The student will investigate and explain how physical and digital security measures can protect electronic information for businesses, governments, and organizations.

a. Investigate and explain how physical and digital security measures are used to safeguard electronic information.
b. Research the advantages and limitations of different security measures in protecting users against security threats.
c. Explore how emerging technologies may affect methods to safeguard personal and public data.

Understanding the Standard

Electronic information must be protected from unauthorized access, manipulation, theft, and damage to maintain its security, integrity, and availability. Protection involves the use of both physical security measures, such as restricted access to hardware, and digital safeguards, including encryption, firewalls, and access controls.

- Physical security measures protect computing devices and hardware from unauthorized access, theft, or physical damage. Physical security measures consist of locked server rooms, surveillance systems, and controlled access to hardware to help prevent tampering, theft, or destruction of devices that store sensitive information

- Digital security measures are protections integrated into software, applications, and network systems to prevent unauthorized access, data breaches, and malicious activity. Digital security measures include strong passwords, firewalls, encryption, multi-factor authentication, intrusion detection systems, and regular software patching to prevent unauthorized access to networks and digital assets.

Together, these measures form a comprehensive defense strategy to reduce vulnerabilities and ensure the confidentiality, integrity, and availability of electronic information in the face of cyber threats. Cyberattacks can take many forms, ranging from large-scale breaches of sensitive information to targeted attempts to compromise individual devices. Among the most common and damaging types of cyber threats is malware, a broad category of malicious software designed to infiltrate and damage computer systems.

[8.CYB.2a] Physical and digital security measures work together to safeguard electronic information. Physical security involves protecting hardware and infrastructure from physical threats, such as theft, damage, or unauthorized access. This can be achieved through measures like secure facilities, access controls, surveillance systems, and environmental controls. Digital security focuses on protecting data and systems from cyber threats. This includes strong passwords, regular software updates, antivirus software, firewalls, encryption, and user education. By combining physical and digital security measures, organizations can create a layered defense to protect their valuable data and systems.

[8.CYB.2b] There are advantages and limitations of different security measures that can be used against threats. Examples are provided in the table below:

Security Measure	Advantages	Limitations
Firewalls	• blocks unauthorized access • filters incoming and outgoing traffic	• can be bypassed by sophisticated attackers • may slow down network performance • requires careful configuration.
Antivirus Software	• detects and removes malware • protects against viruses, worms, and ransomware	• may not detect all new threats • can slow down system performance • requires regular updates.
Intrusion Detection Systems (IDS)	• monitors network traffic for suspicious activity • detects intrusions in real-time • can trigger alerts	• may generate false positives • requires careful configuration • may not detect sophisticated attacks
Encryption	• protects sensitive data by converting it into unreadable code ensuring confidentiality and integrity	• can be computationally intensive • requires strong encryption keys • may not protect against all types of attacks
Multi-Factor Authentication (MFA)	• adds an extra layer of security by requiring multiple forms of identification, such as passwords, biometrics, or security tokens.	• can be inconvenient for users • may not be suitable for all applications

[8.CYB.2c] Industries and emerging technologies continue to evolve in response to the increasing complexity and frequency of cyberattacks. Artificial Intelligence (AI) and Machine Learning (ML) are central to this advancement, offering enhanced capabilities in threat detection, real-time response automation, and predictive analytics. These technologies support the identification of anomalous behavior, enable dynamic risk assessment, and contribute to the development of adaptive cybersecurity frameworks capable of addressing both known and novel threats.

AI and ML systems analyze vast volumes of data from network traffic, user behavior, and system logs to establish baselines of normal activity. When deviations occur, the system can flag them as potential threats. For example, behavioral analytics powered by ML can detect insider threats or compromised accounts by identifying patterns that deviate from a user's typical behavior. AI-driven security information and event management (SIEM) systems can automatically correlate security alerts across multiple data sources and prioritize incidents for human analysts. In advanced settings, automated response tools use AI to isolate affected systems, block malicious IP addresses, or revoke user

credentials in real time, significantly reducing response time and limiting damage. These applications demonstrate how AI and ML actively support scalable, intelligent, and responsive cybersecurity defense mechanisms.

Concepts and Connections

Concepts

Cybersecurity safeguards include both physical and digital techniques used to protect electronic information from unauthorized access, alteration, or theft. Understanding the advantages and limitations of these safeguards is essential for assessing their effectiveness against evolving security threats. Analyzing emerging technologies offers insight into how new tools and strategies influence the protection of personal and public data.

Connections

Within the grade level/course: At this grade level, students further examine physical and digital security measures that protect electronic information.(8.CYB.1).

Vertical Progression: In Grade 7, students compare and contract physical and digital security measures to safeguard information and identify potential threats and solutions to these vulnerabilities. (7.CYB.1)

Across Content Areas

English

- **8.R.1** The student will conduct research and read a series of conceptually related texts on selected topics to build knowledge on grade-eight content and texts, solve problems, and support cross-curricular learning.

History and Social Science

- **CE.6.d** The student will examine how civic participation can address community needs and serve the public good, including staying informed about current issues.

Digital Learning Integration

- **6-8.DC** Students recognize the rights, responsibilities and opportunities of living, learning and working in an interconnected digital world, and they act in ways that are safe, legal, and ethical.
  A. Students manage their digital identities and reputations, including demonstrating an understanding of their digital footprints.
  B. Students demonstrate and advocate for positive, safe, legal, and ethical habits (established behaviors) when using technology and interacting with others online.

Opportunities for Computer Science Integration

Curriculum integration strengthens conceptual understanding and skill application. This can be done through multidisciplinary, interdisciplinary, and transdisciplinary approaches to integration. The examples below illustrate multiple ways to integrate computer science.

History and Social Science

- Students investigate how governments and corporations have responded to cybersecurity threats and breaches, tracing the evolution of physical and digital security measures. Students analyze major data breaches and infrastructure attacks to evaluate the societal impacts and policy shifts that followed. By examining legislative milestones (e.g., GDPR, HIPAA, or cybercrime laws) and institutional strategies, students will assess how security frameworks have adapted to emerging threats, balancing privacy, innovation, and civil liberties.

English

- Students write an informational report or comparative analysis on physical vs. digital security measures used by businesses, governments, and organizations. They will research real-world examples of security breaches and evaluate the effectiveness of various protection strategies, incorporating evidence-based arguments. Additionally, students can explore how emerging technologies like artificial intelligence and biometrics impact security and ethical considerations in protecting personal and public data.

Skills in Practice

Students should engage in the following practices to deepen their conceptual understanding and enhance the application of skills aligned with the Computer Science *Standards of Learning*. These practices are explained in more detail in Appendix A.

B. Fostering Computational Thinking Practices:

1. Decompose Relevant Problems
2. Explore Common Features and Relationship to Extract Patterns
3. Apply Abstraction to Simplify, Represent, and Problem Solve

D. Fostering Digital Literacy Practices:

4. Responsible Use Practices
5. Safeguard Well-Being of Self and Others
6. Evaluate Resources and Recognize Contributions

Data and Analysis (DA)

8.DA.1 The student will create computational models to simulate events or represent phenomena.

a. Compare and contrast the use of computational models and simulations to analyze patterns and replicate phenomena.
b. Design and create complex computational models that simulate dynamic systems (abstraction), incorporating multiple variables and interactions.
c. Refine computational models based on generated outcomes.

Understanding the Standard

Computational tools and computational models are closely related, as both support problem-solving, data analysis, and the exploration of complex systems, but they serve different functions. Computational tools are software applications or platforms that assist users in collecting, organizing, analyzing, or visualizing data. These tools enable users to identify trends, make predictions, and better understand complex information.Computational tools include applications and software, like spreadsheets, data visualization platforms, and coding environments, They help make information more accessible and interpretable but may not inherently simulate dynamic systems. When used to build computational models, these tools allow students to simulate real-world systems by manipulating variables, observing outcomes, and testing hypotheses.

Computational models simulate real-world phenomena by using code, logic, and data to represent interactions between variables. These models allow for the analysis of outcomes based on changing conditions or inputs. Creating computational models involves simulating real-world phenomena using algorithms, data, and rule-based systems. These models represent variable relationships and system behaviors to support prediction, hypothesis testing, and decision-making. Applications include modeling weather patterns, analyzing population growth, and optimizing logistics processes. Constructing computational models develops conceptual understanding of dynamic systems and supports skill development in abstraction, decomposition, and algorithmic reasoning.

[8.DA.1a] Computational models and simulations are tools used to represent, analyze, and replicate real-world phenomena. A computational model defines the variables, parameters, and logical rules that describe the behavior of a system. A simulation uses that model to generate outcomes over time or under specific conditions. Computational models focus on the structure and relationships within a system, while simulations emphasize the execution of the model to observe and analyze resulting patterns or behaviors.

Computational Models

- Focus: Abstract representation of a system or process.
- Purpose: Understanding underlying principles and relationships.
- Approach: Mathematical equations and algorithms.
- Output: Numerical results, statistical data.

- Example: A mathematical model of population growth, where variables like birth rate, death rate, and migration are used to predict future population trends.

Simulations

- Focus: Replication of real-world behavior.
- Purpose: Predicting outcomes and testing hypotheses.
- Approach: Computer programs and software tools.
- Output: Visualizations, data logs, or physical representations.
- Example: A flight simulator, where the behavior of an aircraft is simulated in a virtual environment to train pilots.

Key Differences

- Level of Abstraction: Computational models often focus on the core principles and relationships, while simulations aim to replicate the specific details of a system.
- Output Format: Computational models typically produce numerical data, while simulations often generate visual representations or physical outputs.
- Application: Computational models are useful for understanding fundamental concepts and making theoretical predictions, while simulations are valuable for testing hypotheses, optimizing designs, and training.

Computational models and simulations are often used together to support iterative analysis and refinement. A computational model defines the variables, relationships, and logic governing a system, while a simulation executes the model to observe how it behaves under varying conditions. The outcomes from simulations can inform adjustments to the underlying model, enhancing predictive accuracy and system understanding. This iterative cycle supports continuous model validation, enabling more reliable forecasting and decision-making.

[8.DA.1b] Designing and constructing complex computational models entails developing abstracted representations of real-world systems that incorporate multiple interacting variables. These models use algorithms and structured logic to simulate system behaviors under varying conditions. Through abstraction, only the most relevant parameters are included, enabling efficient simulations and analysis. For example, a meteorological model may include atmospheric variables such as temperature, pressure, humidity, and wind vectors to project weather patterns, while a transportation model may integrate vehicle density, signal timing, and infrastructure constraints to analyze traffic flow. These models facilitate predictive analysis, hypothesis testing, and system optimization, providing a structured framework for understanding and interpreting complex phenomena.

To create a complex computational model, one must first identify a real-world system to simulate. This could range from a simple pendulum to a complex ecological system.

- Next, identify the key variables and parameters that influence the system's behavior. These variables might include factors like initial conditions, forces, or environmental conditions.
- Once the variables are identified, formulate mathematical equations to describe the relationships between these variables. These equations will form the core of the computational model.
- Choose a suitable modeling tool or programming language that will provide the necessary information for the task.
- Using the chosen tool, implement the mathematical model and define the simulation parameters, such as time step, initial conditions, and simulation duration.
- Finally, run the simulation and analyze the results. This involves visualizing the output data, identifying patterns, and drawing conclusions about the system's behavior.

To support students in developing computational models, teachers should provide instruction on identifying relevant real-world phenomena, selecting key variables, and defining the relationships between them. Students should learn to represent these relationships using logic and programming within a computational tool or environment. Instruction should emphasize the importance of testing, refining, and comparing model behavior to real-world outcomes, while reinforcing concepts such as input/output, iteration, and abstraction.

[8.DA.1c] Refining computational models involves systematically adjusting variables, parameters, or rules within the model to improve its alignment with observed data. This process includes comparing model outputs to real-world data or experimental results and identifying discrepancies. Revisions are made to the model’s structure or inputs to increase accuracy and better represent the behavior of the system being modeled.

Here are some specific techniques for refining computational models:

- Parameter calibration: Adjusting model parameters to match observed data.
- Model structure refinement: Modifying the model's underlying equations or algorithms.
- Data validation: Ensuring the quality and accuracy of the input data.
- Sensitivity analysis: Identifying the impact of different parameters on model outputs.
- Model validation: Comparing model predictions to experimental or real-world data.

Model refinement is the iterative process of adjusting variables, parameters, or structures within a computational model to improve its alignment with observed data or system behavior. This process supports the development of models that accurately represent complex systems and produce reliable outputs for analysis.

Concepts and Connections

CONCEPTS

Computational models and simulations are used to analyze patterns, test scenarios, and replicate real-world phenomena. Developing these models involves designing dynamic systems with multiple variables and interactions, applying abstraction to manage complexity, and refining the models based on observed outcomes to improve accuracy and reliability.

CONNECTIONS

Within this grade level: At this grade level, students develop and evaluate computational models to simulate real-world phenomena, using abstraction and critical thinking to interpret data and relationships. (8.DA.1)
Vertical progression: In Grade 7, students are developing computational models to simulate and observe data in order to draw conclusions and make predictions. (7.DA.1)

ACROSS CONTENT AREAS

History and Social Science

- **WG.12** The student will apply social science skills to understand the distribution, growth rates, and characteristics of human population.

Mathematics

- **8.PS.3** The student will apply the data cycle with a focus on scatterplots. 8.PS.2: The student will apply the data cycle with a focus on boxplots.

Science

- **8.1** The student will demonstrate an understanding of scientific and engineering practice. *8.1 standard is integrated within science content and not taught in isolation. Potential science concepts to apply 8.1 include: PS.3 The student will investigate and understand that matter has properties and is conserved in chemical and physical processes. PS.5 The student will investigate and understand that energy is conserved. PS.6. The student will investigate and understand that waves are important in the movement of energy. PS.7 The student will investigate and understand that electromagnetic radiation has characteristics. PS.8 The student will investigate and understand that work, force, and motion are related. PS.9 The student will investigate and understand that there are basic principles of electricity and magnetism.*

Opportunities for Computer Science Integration

Curriculum integration strengthens conceptual understanding and skill application. This can be done through multidisciplinary, interdisciplinary, and transdisciplinary approaches to integration. The examples below illustrate multiple ways to integrate computer science.

History and Social Science
Students use computational tools to develop a dynamic model that simulates population growth and demographic shifts over time, enhancing both mathematical reasoning and analytical rigor. By incorporating key variables such as birth rate and migration flows, students will construct algorithmic representations that reflect real-world population dynamics.

Mathematics
Students collect or analyze existing datasets to identify variables, organize data structures, and generate visual representations that highlight numerical relationships. Through this process, students will interpret patterns, evaluate correlations, and adjust their models to improve predictive accuracy.

Science
Students create a computational model or simulation that represents how energy moves through waves. Using a block-based coding tool, spreadsheet formulas, or an interactive diagram, students can adjust variables such as wave type, frequency, and amplitude to observe how energy transfer changes. By comparing longitudinal and transverse waves in their model, students analyze patterns and refine their understanding of how different wave properties impact energy movement, aligning with both computational modeling and scientific concepts.

Skills in Practice

Students should engage in the following practices to deepen their conceptual understanding and enhance the application of skills aligned with the Computer Science *Standards of Learning*. These practices are explained in more detail in Appendix A.

B. Fostering Computational Thinking Practices:

1. Decompose Real-World Problems
2. Explore Common Features and Extract Patterns
3. Apply Abstraction to Simplify, Represent, and Problem Solve
4. Apply Algorithmic Thinking to Problem Solve and Create
5. Apply Computational Thinking Practices to Select, Organize, and Interpret Data

C. Fostering Iterative Design Practices:

1. Identify, Define, and Evaluate Real-World Problems

2. Plan and Design Artifacts
3. Create, Communicate and Document Solutions
4. Test and Optimize Artifacts

8.DA.2 The student will evaluate computational models to analyze patterns and make recommendations or predictions.

a. Define data biases within a dataset and the unintended consequences that may impact data reliability and final analysis.
b. Analyze patterns and interpret data generated by computational models and simulations, identifying meaningful patterns and relationships.
c. Utilize data visualization techniques to communicate and present findings derived from computational models and simulations.

Understanding the Standard

The ability to evaluate computational models is a crucial skill in today's data-driven world. By analyzing patterns and trends within computational models, individuals can gain valuable insights and make informed decisions. Whether it's predicting future market trends, optimizing supply chain logistics, or understanding complex biological systems, computational models provide a powerful tool for exploring and interpreting data. By mastering the art of model evaluation, students can unlock the full potential of these tools and contribute to innovation and problem-solving in various fields.

[8.DA.2a] Data bias refers to systematic errors that occur when datasets do not accurately reflect the characteristics of the target population. This misrepresentation can influence the outcomes of analyses, resulting in skewed patterns or imbalanced conclusions. Bias may originate from sampling imbalances, incomplete data, or disparities embedded within the data sources. Its presence affects the reliability and generalizability of data-driven systems, potentially leading to inaccurate outputs in computational processes.

[8.DA.2b] Pattern analysis and data interpretation in computational models involve the examination of output data to identify trends, correlations, and irregularities. This process utilizes statistical methods, visualization techniques, and algorithmic tools to extract structured information from simulation results. The purpose is to assess relationships among variables, evaluate model behavior, and support verification of the model’s underlying assumptions. Pattern analysis contributes to the validation and refinement of computational representations.

[8.DA.2c] Data visualization is the representation of computational data in graphical formats to support analysis and communication of model outputs. Visualization techniques include bar charts, line graphs, scatter plots, histograms, box plots, heatmaps, and geospatial maps. These visual forms display variable relationships, highlight trends, reveal distributions, and identify outliers in datasets. Interactive visualizations allow for filtering, zooming, and dynamic adjustment of parameters to support exploratory analysis. Data visualization serves as a tool for interpreting results, identifying patterns, and conveying structured findings from computational models and simulations to varied audiences.

Students present findings from computational models and simulations by organizing outputs, summarizing results, and identifying patterns in the data. Outputs may include numerical results, observed trends, or comparisons between predicted and actual values. Presentations may use

reports, charts, labeled graphs, or data tables to display information clearly. Accurate terminology, consistent formatting, and alignment with model variables ensure that findings are communicated precisely and support interpretation by others.

Concepts and Connections

Concepts

Data analysis is the process of examining, interpreting, and presenting data to identify meaningful patterns and insights. It is applied to computational models and simulations to evaluate outcomes, identify biases that affect reliability, and interpret patterns and relationships.

Connections

Within this grade level: At this grade level, students analyze data generated by computational models, identifying patterns and addressing biases to enhance data reliability. (8.DA.2)
Vertical progression: In Grade 7, students explain the process and application of data's role in machine learning and artificial intelligence. (7.DA.2)

Across Content Areas

History and Social Science

- **WG.13** The student will analyze the causes, impacts, and responses related to migration.

Science

- **8.1** The student will demonstrate an understanding of scientific and engineering practice. *8.1 standard is integrated within science content and not taught in isolation. Potential science concepts to apply 8.1 include: PS.3 The student will investigate and understand that matter has properties and is conserved in chemical and physical processes. PS.5 The student will investigate and understand that energy is conserved. PS.6. The student will investigate and understand that waves are important in the movement of energy. PS.7 The student will investigate and understand that electromagnetic radiation has characteristics. PS.8 The student will investigate and understand that work, force, and motion are related. PS.9 The student will investigate and understand that there are basic principles of electricity and magnetism.*

Digital Learning Integration

- **6-8.CT** Students develop and employ strategies for understanding and solving problems in ways that leverage the power of technological methods, including those that leverage assistive technologies, to develop and test solutions.
  A. Students create, identify, explore, and solve problems using technology-assisted methods such as data analysis, modeling, or algorithmic thinking.

D. Students demonstrate an understanding of how automation works and use algorithmic thinking to design and automate solutions.

Opportunities for Computer Science Integration

Curriculum integration strengthens conceptual understanding and skill application. This can be done through multidisciplinary, interdisciplinary, and transdisciplinary approaches to integration. The examples below illustrate multiple ways to integrate computer science.

History and Social Science

- Students use computational tools to construct a data-driven visualization that models change in economic activity or labor distribution over time, drawing from historical datasets and applying analytical reasoning. Students examine how economic factors such as industrialization, automation, or policy shifts affect labor trends, while identifying patterns and interpreting correlations within the data. Throughout the process, students critically assess the reliability and potential biases of the data sources, evaluating how limitations in collection methods or scope may influence outcomes.

Science

- Students design a computational model or simulation to track how energy is transferred and transformed in a dynamic system, such as a roller coaster or pendulum. By adjusting key variables like height, mass, or velocity, they observe how changes affect kinetic and potential energy levels and analyze principles of energy conservation. Students apply data visualization techniques to translate complex computational outputs into accessible, compelling representations.

Skills in Practice

Students should engage in the following practices to deepen their conceptual understanding and enhance the application of skills aligned with the Computer Science *Standards of Learning*. These practices are explained in more detail in Appendix A.

B. Fostering Computational Thinking Practices:

1. Decompose Real-World Problems
2. Explore Common Features and Extract Patterns
3. Apply Abstraction to Simplify, Represent, and Problem Solve
4. Apply Algorithmic Thinking to Problem Solve and Create
5. Apply Computational Thinking Practices to Select, Organize, and Interpret Data

C. Fostering Iterative Design Practices:

1. Identify, Define, and Evaluate Real-World Problems

2. Plan and Design Artifacts
3. Create, Communicate and Document Solutions
4. Test and Optimize Artifacts

Impacts of Computing (IC)

8.IC.1 The student will assess the social impacts and ethical considerations of computing technologies.

a. Analyze the impact of sharing data through computing technologies.
b. Critique the role the Internet plays in social life, the global economy, and culture.
c. Evaluate online and print sources for credibility and reliability.
d. Research and discuss factors that impact access and availability to computing technologies.
e. Discuss ethical issues around cybersecurity and networks: censorship, privacy, safety, and access.

Understanding the Standard

Computing technologies have transformed global systems by enabling new levels of innovation, efficiency, and connectivity across industries and daily life. From automation and data analytics to artificial intelligence and digital communication, these tools shape how people work, learn, and interact. Alongside these advancements are critical social and ethical considerations, including issues of privacy, security, and accountability. Understanding these implications is essential for making informed decisions about the development and use of technology in society.

[8.IC.1 a] Sharing data through computing technologies has significantly transformed workflows across industries, enabling rapid, large-scale information exchange that drives innovation in nearly every sector of modern society. Cloud platforms, networked systems, and real-time analytics have increased the speed, volume, and reach of data sharing. However, these capabilities also raise critical challenges, including privacy breaches, unauthorized data use, and algorithmic bias. Addressing these risks requires strong encryption, secure access controls, and transparent data policies. Clear accountability mechanisms and ongoing evaluation are essential to ensure responsible data use in an increasingly connected digital environment.

[8.IC.1b]. The Internet has impacted social life, the global economy, and culture by enabling global connectivity and rapid information exchange. It has redefined how people communicate, access resources, and participate in economic and cultural systems. While these changes have created new opportunities for collaboration, innovation, and inclusion, they have also introduced challenges related to privacy, misinformation, and digital disparities. Balancing the benefits of digital connectivity with its social and ethical implications is essential for navigating an increasingly interconnected world. Society.

[8.IC.1c] Evaluating online and print sources requires checking for credibility and reliability. Credibility refers to the trustworthiness of the source, based on the author's qualifications, institutional affiliation, and the publication's reputation. Reliability refers to the accuracy and consistency of the content, determined by verifying facts across reputable sources, identifying citations, assessing for evidence-based claims, and reviewing for bias or misinformation. Additional considerations include the date of publication and whether the content has undergone peer review.

[8.IC.1d] Access and availability of computing technologies are influenced by several factors, including geographic location, socioeconomic status, infrastructure, and policy. Rural or remote areas may lack broadband infrastructure, limiting internet access. Socioeconomic status affects the ability to purchase devices, maintain connectivity, or access digital services. Infrastructure includes the presence of reliable power, mobile networks, and public computing spaces such as libraries or schools. Policy decisions at local, state, or national levels also impact funding for technology access, digital inclusion programs, and public-private partnerships that expand infrastructure and device availability.

[8.IC.1e] Cybersecurity and networked systems raise complex ethical issues. Key concerns include:

- Censorship: Control over online content can restrict access to information and suppress viewpoints.
- Privacy: The collection, storage, and use of personal data introduce risks of surveillance, data misuse, and breaches that may result in identity theft or financial loss.
- Safety: Cyberattacks—including hacking, phishing, and malware—threaten individuals, organizations, and infrastructure by compromising data integrity and operational stability.
- Access: Disparities in internet access and digital resources reinforce existing inequalities, limiting opportunities for education, employment, and civic participation.

Addressing these issues requires coordinated strategies that uphold security, enforce privacy protections, and expand digital access. Ethical management of cybersecurity must balance these priorities to ensure a safe and transparent digital environment.

Concepts and Connections

CONCEPTS

The Internet plays a critical role in shaping social interaction, cultural development, and economic activity on a global scale. Data sharing is the transmission and exchange of information through computing technologies and it has direct implications for privacy, security, and accessibility. Addressing these concerns requires evaluating the credibility and accuracy of information sources, examining technical and socioeconomic factors that influence access to digital resources, and analyzing ethical issues related to cybersecurity and network use, including content control, data protection, system integrity, and access.

CONNECTIONS

Within this grade level: At this grade level, students explore the influence of computing technologies on social and cultural dynamics, as well as global trade and communication networks. They evaluate data sharing, ethical concerns, and access disparities, developing skills to critique sources for credibility and reliability. (8.IC.1)

Vertical progression: In Grade 7, students expand their assessment of the impact of computing technologies beyond local society to consider the global impact as well. (7.IC.1)

Across Content Areas

English

- **8.W.1.C** The student will write persuasively, supporting well-defined points of view effectively with relevant evidence and clear reasoning in ways that logically advance the claim(s) made.

History and Social Science

- **WG.16** The student will understand the impact of the growing interdependence of the world by analyzing global trade and communication networks.

Digital Learning Integration

- **6-8.DC** Students recognize the rights, responsibilities and opportunities of living, learning and working in an interconnected digital world, and they act in ways that are safe, legal, and ethical.
  D. Students demonstrate an understanding of what personal data is, how data collection technologies work, tradeoffs of sharing personal data, and best practices for keeping it private and secure.
- **6-8.KC** Students critically curate a variety of digital resources using appropriate technologies, including assistive technologies, to construct knowledge, produce creative digital works, and make meaningful learning experiences for themselves and others.
  B. Students practice and demonstrate the ability to evaluate digital sources for accuracy, perspective, credibility, and relevance, including considerations of social and cultural context and bias.

Opportunities for Computer Science Integration

Curriculum integration strengthens conceptual understanding and skill application. This can be done through multidisciplinary, interdisciplinary, and transdisciplinary approaches to integration. The examples below illustrate multiple ways to integrate computer science.

English

- Students write a persuasive essay or create a multimedia presentation exploring a specific ethical issue in computing, such as data privacy, internet censorship, or access to technology with a focus on its broader social impact. Students construct a compelling argument using relevant evidence and reasoning, while critically evaluating how computing practices affect individuals, communities, and societal structures. Students assess the credibility of online and print sources to support their claims and reflect on how digital technologies shape rights, responsibilities, and ethical norms in contemporary society.

History and Social Science

- Students research and analyze how global trade and communication networks have evolved with computing technologies, focusing on the ethical dimensions of cybersecurity and the societal impacts of internet connectivity. Using a computational tool, students create a visual timeline or interactive map that illustrates key moments in the expansion of digital communication.

Skills in Practice

Students should engage in the following practices to deepen their conceptual understanding and enhance the application of skills aligned with the Computer Science *Standards of Learning*. These practices are explained in more detail in Appendix A.

C. Fostering Iterative Design Practices:

1. Identify, Define, and Evaluate Real-World Problems
3. Create, Communicate, and Document Solutions

D. Fostering Digital Literacy Practices:

1. Responsible Use Practices
2. Safeguard Well-Being of Self and Others
3. Evaluate Resources and Recognize Contributions

8.IC.2 The student will analyze and evaluate the ramifications of screen time in one's life.

a. Analyze scenarios or case studies to assess the impact of screen time on one's physical and mental health.
b. Justify the argument that excessive screen time and video games can have significant consequences for the physical, emotional, and cognitive development of children and adolescents.

Understanding the Standard

[8.IC.2a] Screen time affects physical and mental health through its influence on activity levels, vision, cognitive load, and sleep. Extended use of digital devices reduces physical movement, increasing the risk of obesity, muscle weakness, and poor posture. Continuous screen exposure contributes to visual strain, with symptoms such as headaches, blurred vision, and dry eyes. Nighttime screen use disrupts circadian rhythms, impairing sleep quality and affecting cognitive function and emotional regulation.

[8.IC.2 b] Excessive screen time and frequent digital media use have been linked to negative outcomes in physical, emotional, and cognitive development, particularly in children and adolescents. High levels of screen exposure are associated with increased risks of anxiety, depression, attention difficulties, and disrupted sleep patterns. Cognitive impacts include reduced executive functioning, such as impaired working memory, diminished impulse control, and decreased problem-solving ability. Social consequences may include cyberbullying, reduced face-to-face interaction, and lowered self-esteem due to constant social comparison.

Concepts and Connections

CONCEPTS

Screen time affects physical health, mental well-being, and cognitive development. Excessive use, including extended periods of gaming, is associated with reduced physical activity, disrupted sleep patterns, and increased emotional or behavioral concerns in children and adolescents. Examining these effects through case studies or real-world scenarios provides evidence to better understand and address the risks linked to screen overuse.

CONNECTIONS

Within this grade level: At this grade level students evaluate the impact of screen time on health by analyzing real-world scenarios and case studies. (8.IC.2)
Vertical progression: In Grade 7, students examine the impact of screen time on their interactions with others and create a social media plan demonstrating safe and responsible use. (7.IC.2)

Across Content Areas

English

- **8.W.1.C** The student will write persuasively, supporting well-defined points of view effectively with relevant evidence and clear reasoning in ways that logically advance the claim(s) made.

History and Social Science

- **WG.14** The student will determine cultural patterns and interactions across time and place by explaining intellectual exchanges among cultures in areas such as science and technology.

Digital Learning Integration

- **6-8.KC** Students critically curate a variety of digital resources using appropriate technologies, including assistive technologies, to construct knowledge, produce creative digital works, and make meaningful learning experiences for themselves and others.
  A. Students practice and demonstrate the ability to effectively use research strategies to locate appropriate primary and secondary digital sources in a variety of formats to support their academic and personal learning and create a research product.

Opportunities for Computer Science Integration

Curriculum integration strengthens conceptual understanding and skill application. This can be done through multidisciplinary, interdisciplinary, and transdisciplinary approaches to integration. The examples below illustrate multiple ways to integrate computer science.

English

Students write a reflective essay or response analyzing a case study on the effects of screen time. They will use evidence from texts, research, or personal experiences to justify their argument on how excessive screen time can impact physical, emotional, and cognitive development, demonstrating critical thinking and structured reasoning.

History and Social Science

Students research how advancements in screen-based technology have influenced cultural interactions over time. They can create a comparative analysis that examines historical shifts in communication, entertainment, and social behaviors due to increased screen time, such as the impact of television, smartphones, and social media on different societies.

Science

- Students investigate how screen time influences physical health (e.g. sleep cycles, vision, posture) and mental well-being (e.g. mood, stress, attention). They'll collect data, analyze patterns, and communicate findings using scientific reasoning and computational tools.

Skills in Practice

Students should engage in the following practices to deepen their conceptual understanding and enhance the application of skills aligned with the Computer Science *Standards of Learning*. These practices are explained in more detail in Appendix A.

A. Fostering Collaborative Computing Practices:

1. Cultivate Relationships and Norms
2. Include Multiple Perspectives

C. Fostering Iterative Design Practices:

1. Identify, Define, and Evaluate Real-World Problems

D. Fostering Digital Literacy Practices:

1. Responsible Use Practices
2. Safeguard Well-Being of Self and Others
3. Evaluate Resources and Recognize Contributions

8.IC.3 The student will identify opportunities for education, training, and preparation to enter into a chosen computer science career field.

a. Identify an education and training plan for a chosen computer science career.
b. Outline the use of computer science skills required in a chosen career.
c. Develop short-and long-term goals for a chosen career.
d. Research emerging trends in a chosen career path.

Understanding the Standard

[8.IC.3abcd]A successful career in computer science requires selecting relevant educational pathways, specialized training programs, and industry-recognized certifications aligned with specific career goals. Building a strong foundation in core concepts such as algorithms, programming, and data structures is essential. As the field evolves, staying current with emerging technologies—such as artificial intelligence, cybersecurity, and cloud computing—is critical. Gaining hands-on experience through internships, projects, or open-source contributions further enhances readiness for a competitive and dynamic workforce.

Identifying opportunities for education, training, and preparation in a computer science career involves researching academic programs, technical certifications, and real-world experiences that align with a specific job role or field. This includes selecting appropriate high school or postsecondary courses, enrolling in degree programs or bootcamps, earning certifications in areas such as cybersecurity or cloud computing, and participating in internships, coding competitions, or mentorships. These opportunities help build the knowledge, skills, and experience required to enter and succeed in a chosen area of computer science.

Concepts and Connections

CONCEPTS

Effective career planning requires understanding educational pathways, required skills, and industry expectations. Investigating emerging trends within the field ensures readiness for advancing technologies and future workforce demands.

CONNECTIONS

Within the grade level/course: At this grade level, students develop education and training paths for emerging and high demand careers. They should examine that the core skills needed in emerging computer science fields related business, healthcare, and research.

Vertical Progression: In Grade 7, students self-assessed and examined individual preferences and related them to a chosen computer science career. (7.IC.3)

Opportunities for Integration

Curriculum integration strengthens conceptual understanding and skill application. This can be done through multidisciplinary, interdisciplinary, and transdisciplinary approaches to integration. The examples below illustrate multiple ways to integrate computer science.

English

- Students create a digital artifact that showcases their chosen computer science career pathway and communicates their future aspirations. This artifact integrates key elements of career research, including required education and certifications, relevant technical and soft skills, and a personal timeline with short- and long-term goals. Students synthesize their findings through compelling visual and written components, such as professional cover letters, tailored resumes, and career reflection narratives.

History and Social Science

- Students research the evolution of computer science careers and their impact on society. By analyzing how technological advancements have shaped industries over time, they can explore how different historical events influenced the demand for computer science professionals and the development of new career pathways.

Mathematics

- Students design and build a program that calculates and displays a financial plan based on a chosen computing career. They'll use variables, arithmetic operations, conditionals, and user input to explore budgeting, goal setting, and data visualization through code.

Skills in Practice

Students should engage in the following practices to deepen their conceptual understanding and enhance the application of skills aligned with the Computer Science *Standards of Learning*. These practices are explained in more detail in Appendix A.

A. Fostering Collaborative Computing Practices:

1. Cultivate Relationships and Norms
2. Include Multiple Perspectives

D. Fostering Digital Literacy Practices:

2. Safeguard Well-Being of Self and Others
3. Evaluate Resources and Recognize Contributions

Networks and the Internet (NI)

8.NI.1 The student will model and describe the role of computing devices in transmitting data in and on computing networks and the Internet.

a. Identify the roles of computing devices: routers, switches, servers, and clients communicating over a network.
b. Design a network topology of computing devices.
c. Demonstrate how data is transmitted over networks and the Internet.
d. Analyze factors that strengthen or weaken network connectivity.

Understanding the Standard

The internet has revolutionized social life, culture, and the economy. It connects people across the globe, fostering communication and collaboration more quickly than historically possible. It has transformed how we consume information, entertainment, and news. Additionally, it has driven economic growth, enabling e-commerce, remote work, and digital innovation. As such, the speed and reliability of data transfer are crucial for maximizing the potential of the internet in these areas.

[8.NI.1a] Computing devices communicate over a network by exchanging data between clients and servers, with network infrastructure components such as routers, switches, and servers that manage data transmission, direction, and access.

- Routers act as traffic directors, determining the best path for data packets to travel across the network. They connect multiple networks and ensure data reaches its destination efficiently.
- Switches connect devices within a local network, such as computers, printers, and servers. They facilitate communication between these devices, ensuring data is transmitted to the correct destination.
- Servers store and manage data and resources, providing services to clients. They can be file servers, web servers, database servers, or application servers, offering various services to network users.
- Clients are devices that request and receive services from servers. This includes computers, smartphones, tablets, and other devices that connect to a network to access resources and information.

[8.NI.1b] Network topology refers to the physical or logical arrangement of computing devices within a network. When selecting an appropriate topology, it is important to consider scalability, security, performance, and cost, as well as how easily the network can be maintained and how it handles faults or disruptions.

8.NI.1c] Data transmission across networks and the Internet involves dividing information into smaller packets, directing them through multiple interconnected networks, and reassembling them in the correct order at the destination device.

- Once a device, such as a computer or smartphone, generates data, it is segmented into packets, each containing a portion of the data and metadata like the source and destination addresses.
- These packets are then transmitted over the local network, often Wi-Fi, to the internet service provider (ISP). The ISP's network connects to the vast internet backbone, a global network of interconnected routers.
- Routers act as traffic directors, forwarding packets to the next hop based on their destination addresses. As packets traverse the internet, they may travel through multiple networks, crossing continents and oceans.
- Upon reaching their destination, the packets are reassembled into the original data, enabling the recipient to access the information. This intricate process, facilitated by protocols like TCP/IP, ensures efficient and reliable transmission of data across the digital landscape.

[8.NI.1d] Several factors can strengthen or weaken network connectivity. Consider the following examples:

Strengthens Network	Weakens Network
• Physical Proximity: Position devices closer to the router to minimize signal interference and improve connection strength. • Wired Connections: Use Ethernet cables for devices that require a stable, high-speed connection. • Router Placement: Place the router in a central location, away from obstacles and electronic interference. • Router Upgrades: Consider upgrading to a newer router with advanced features like beamforming and MU-MIMO. • Firmware Updates: Keep the router's firmware up-to-date to ensure optimal performance and security. • Physical Infrastructure: Maintaining high quality cables in an efficient configuration. • Security Measures: Implementing strong security measures, such as firewalls, intrusion detection systems, and encryption.	• Physical Damage: Physical damage to cables, routers, or other network devices can disrupt connectivity. • Software Issues: Malfunctioning network software, outdated drivers, or security vulnerabilities can compromise network performance. • Cyberattacks: Hackers can exploit network vulnerabilities to disrupt services, steal data, or launch other malicious attacks. • Overload: Excessive network traffic can slow down performance and lead to congestion. • Interference: Interference from other electronic devices, such as microwaves or cordless phones, can degrade Wi-Fi signal strength. • Poor Network Configuration: Incorrect network settings or misconfigurations can hinder performance and security. • Environmental Factors: Extreme temperatures, humidity, or dust can negatively impact network equipment.

Concepts and Connections

CONCEPTS

Networking refers to the interconnection of computing devices—such as routers, switches, servers, and clients—to enable data exchange across local and wide-area networks. It involves three core areas: designing network topologies, understanding data transmission through standardized protocols, and evaluating factors that impact network performance, including bandwidth, latency, reliability, and security.

CONNECTIONS

Within the grade level/course: At this grade level, students explore how computing devices like routers, switches, and servers enable data transmission across networks and the Internet. Designing network topologies and analyzing connectivity factors mirrors the decision-making processes used in geography. (8.NI.1)

Vertical Progression: In Grade 7, Students describe why protocols are needed for data transmission and further expand the process for sending files over the internet. (7.NI.1)

ACROSS CONTENT AREAS

History and Social Science

- **WG.1** The student will explain how geographic information and geospatial tools are used to make decisions.

Opportunities for Computer Science Integration

Curriculum integration strengthens conceptual understanding and skill application. This can be done through multidisciplinary, interdisciplinary, and transdisciplinary approaches to integration. The examples below illustrate multiple ways to integrate computer science.

History and Social Science

- Students research the evolution of network technologies from early telegraph systems and ARPANET to modern-day internet protocols and wireless infrastructures. Students design a network topology map that models how computing devices like routers, switches, and servers transmit data. They will then compare this to how geospatial tools, such as GIS (Geographic Information Systems), help analyze and optimize real-world networks like transportation systems or communication infrastructure.

Mathematics

- Students design a hypothetical network for a school or smart city using graph theory. They calculate optimal routes, node connections, and assign bandwidth and cost using linear equations and ratios.

- Students build a basic spreadsheet or coded simulation that models different network topologies and calculates packet travel time, cost, or redundancy.

Science

- Students investigate how electrical energy moves through a power grid, modeled as a network system using topologies like star or mesh. They will simulate how power travels from a source to nodes (like cities or neighborhoods), analyze efficiency, and calculate energy loss, transmission cost, and load balancing using mathematical formulas.

Skills in Practice

Students should engage in the following practices to deepen their conceptual understanding and enhance the application of skills aligned with the Computer Science *Standards of Learning*. These practices are explained in more detail in Appendix A.

B. Fostering Computational Thinking Practices:

1. Decompose Real-World Problems
2. Explore Common Features and Extract Patterns
3. Apply Abstraction to Simplify, Represent, and Problem Solve
4. Apply Computational Thinking Practices to Select, Organize, and Interpret Data

C. Fostering Iterative Design Practices:

1. Identify, Define, and Evaluate Real-World Problems
2. Plan and Design Artifacts
3. Create, Communicate, and Document Solutions
4. Test and Optimize Artifacts

D. Fostering Digital Literacy Practices:

1. Responsible Use Practices
2. Safeguard Well-Being of Self and Others
3. Evaluate Resources and Recognize Contributions

Appendix A

Grade 6-12 Computer Science Skills and Practices Continuum

Students develop essential practices: collaboration, computational thinking, iterative design, and digital literacy. Students use these practices to engage with core computer science concepts, create artifacts, and problem-solve across disciplines. Artifacts can include but are not limited to prototypes, programs, planning documents, animations, or abstractions. Abstractions can include but are not limited to visualizations, storyboards, flowcharts, infographics, generalizations, decision trees, models, or computer simulations.

A. **Fostering Collaborative Computing Practices**
1. **Cultivate Relationships and Norms:** • **All levels**: Students take turns in different group roles. Recognize group member strengths, ask clarifying questions, and practice self-advocacy. • **Skill progression:** Students also rotate roles and encourage participation. Evaluate group dynamics and improve teamwork. Advocate for the needs of others, such as users needing assistive technology.
2. **Include Multiple Perspectives:** • **All levels:** Students discuss design choices, compare preferences and choices, and incorporate ideas from peers into designs. Ask questions, and seek input from users with diverse abilities, experiences, and perspectives. Reflect on how perspectives aid problem-solving across contexts. • **Skill progression:** Students also integrate user-centered considerations and test with varied audiences throughout the design process. Apply systems thinking, questioning, and data to improve designs.
3. **Create and Accept Feedback:** • **All levels:** Students seek and provide constructive feedback by asking peers probing questions like "why do you think that?" and sharing ideas to help improve. Practice active listening and paraphrasing. Begin to use evidence to support feedback and distinguish opinions. Reflect on how feedback aids problem-solving across contexts. • **Skill progression:** Students also tailor constructive feedback to support peers' stated goals. Create action steps from feedback to optimize designs and improve problem-solving practices, such as collaboration.
4. **Select and Use Collaboration Tools:** • **All levels:** Students use collaboration tools like whiteboards, digital tools, and paper. Use computational thinking practices like sequencing (algorithms) and representations (abstractions) to collaboratively plan and create. Use timelines and begin to make decisions about which collaboration tools to use. Reflect on what strategies worked as predicted and ways to improve.

- **Skill progression:** Students also choose collaboration tools and evaluate tradeoffs of various methods of project management. Refine through repeated cycles of development and feedback.

Instructional Considerations for Collaboration Practices

Possible **instructional approaches** to foster collaboration practices:

1. Design instruction around authentic problems that require collaboration. Assign roles, provide clarifying and probing question stems, and model strategies students can use to identify and advocate for their needs.
2. Provide resources to support exploring diverse viewpoints and end users. Model curiosity, perspective-taking, and empathy.
3. Model sentence stems for constructive feedback, establish routines for self and group reflection, and practice incorporating diverse viewpoints. Implement pair programming with opportunities to practice giving and receiving feedback.
4. Model tool selection and project management structures. Provide opportunities to practice various methods and reflect.

Instructional activities may include but are not limited to:

- **Classroom Discussions:** Organize discussions that engage students in respectful discourse to acknowledge opposing perspectives.
- **Timeline Creation:** Have students create or evaluate and modify timelines that illustrate the steps needed to complete a task as a group.
- **Simulated Shark Tank Innovation Challenge:** Create an innovation design challenge where students collaboratively apply computer science content to solve a problem or launch a new idea.
- **Case Studies:** Provide case studies of design decisions that real computer scientists face and have students analyze and present their recommended choices based on computer science content knowledge.

B. Fostering Computational Thinking Practices

1. **Decompose Relevant Problems:**

- **All levels:** Students break problems, information, and processes into parts. Identify relationships and connections among parts. Reflect on how decomposition aids problem-solving across contexts. Begin to identify subproblems. Apply systems thinking to explore interdisciplinary connections,
- **Skill progression:** Students further break problems into subproblems and integrate existing solutions or procedures. Apply algorithms, such as searching and sorting, to recursively break a problem into similar subtasks that can be solved and combined to solve the main problem.

2. **Explore Common Features and Relationships to Extract Patterns:**

- **All levels:** Students recognize, organize, and describe patterns in data; include relationships and repeated sequences (loops). Explore how visualizations can show relationships and patterns in data. Reflect on how pattern analysis aids problem-solving across contexts. Analyze patterns to develop generalizations and models and test their limits. Use patterns to identify trends and make predictions.

- **Skill progression:** Students also apply pattern analysis to validate inputs, analyze trends, justify design decisions, and create artifacts.

3. **Apply Abstraction to Simplify, Represent, and Problem Solve:**

- **All levels:** Students use and/or create abstractions (e.g. visualizations, storyboards, flowcharts, generalizations, decision trees, models, computer simulations) to simplify problems, represent information, organize thinking, communicate, and create artifacts. Identify and ask questions about the information hidden in an abstraction. Reflect and compare abstractions to explore strengths, limitations, and how they impact problem-solving across contexts.
- **Skill progression:** Students intentionally use abstractions to support throughout the problem-solving process to aid in understanding, planning, and predictions. Incorporate modularity into programs. Create models of complex systems to inform design solutions and understand how and why parts connect. Evaluate and systematically test the limits, opportunities, and uses of models and abstractions.

4. **Apply Algorithmic Thinking to Problem Solve and Create:**

- **All levels:** Students use algorithmic thinking to develop a sequence of steps to plan, create, test, and refine computational artifacts with and without technology. Reflect and identify ways algorithmic thinking aids problem-solving across contexts.
- **Skill progression:** Students use pseudocode and generalizations to organize, create and seek and incorporate feedback on more complex designs. Include automated solutions. Analyze algorithmic solutions and systematically test their limits. Validate their outputs and optimize for design criteria like accessibility.

5. **Apply Computational Thinking Practices to Select, Organize, and Interpret Data:**

- **All levels:** Students use patterns and algorithmic thinking to organize data, identify trends, and make predictions. Use decomposition to explore parts and relationships within data sets. Ask questions about available data sources, and compare and analyze test results to inform decisions, plan, and refine designs.
- **Skill progression:** Students use, organize, and analyze larger and more complex data sets to ask questions; make predictions; optimize problem statements and designs; support claims; and clearly communicate with evidence. Apply mathematical and scientific practices to analyze data, identify relationships within data, and optimize designs. Select among data visualization options and justify choices. Assess strengths and limitations of available data sources.

Instructional Considerations for Collaboration Practices

Possible **instructional approaches** to foster computational thinking practices:

1. Model strategies for breaking complex information into smaller parts. Provide opportunities to analyze and discuss the relationship among parts.
2. Support students with recognizing patterns. Model how to analyze, interpret, and display patterns to make predictions and draw conclusions.
3. Model the use of abstraction (e.g. visualizations, storyboards, flowcharts, decision trees, models, computer simulations) to simplify problems; represent information (e.g. data, patterns, processes, phenomena, systems); organize thinking; and support sense-making. Support students with creating and evaluating abstractions and their limitations.

4. Plan opportunities for students to use sequencing in problem solving, incorporate user feedback, and check for bias, accessibility, and other design criteria. Model ways to systematically test, validate, evaluate, refine, and optimize algorithmic solutions. Provide opportunities to reflect on how algorithms are used in solutions.
5. Model abstraction, pattern analysis, and decomposition. Use models to develop and test predictions. Identify limitations and benefits of models.

Instructional activities may include but are not limited to:

- **Create Artifacts**: Students could create an app, program, animation, simulations, etc. to solve a community problem or creatively express an idea.
- **Identify Patterns to Make Predictions**: Students notice repetition in sequences of numbers or parts of a process to make predictions about future events or missing components.
- **Create Abstractions:** Students create and compare when various abstractions support problem solving, such as models, visualizations, storyboards, flowcharts, decision trees, generalizations, simulations.
- **Modular Programming:** Students break down the steps needed to solve a problem then document subgoals of existing processes. Use those subprograms to simplify programming or reuse solutions for other problems.
- **Create Models:** Develop models to represent information such as patterns, relationships, inputs/outputs. Create models of systems (e.g. model networks, cybersecurity, emerging technologies) to understand how parts connect to perform a function. Students can create models to engage in systems thinking and modularization.
- **Evaluate existing models and programs:** Research technologies and evaluate outputs for bias, accessibility, reliability or other established design criteria. Students can identify applicable parts or modules of existing programs and reuse to solve different problems.
- **Reflection and Transfer:** Have students reflect on how each computational practice facilitates problem-solving and identify opportunities to transfer the practice to other situations. Support students in identifying key points in feedback.

C. Fostering Iterative Design Practices

1. **Identify, Define, and Evaluate Real-World Problems:**

- **All levels:** Students identify, define, and explore existing problems and potential solutions. Ask questions to clarify project goals and explore problems from different perspectives. Clarify success criteria, identify constraints, and uncover missing information. Explore patterns and develop generalizations about the types of problems that benefit from computational solutions.

- **Skill progression:** Students also define and analyze increasingly complex, interdisciplinary problems that can be addressed computationally. Evaluate the efficiency, ethical concerns, and feasibility of solutions. Examine how computer science and emerging technologies affect problem-solving.

2. **Plan and Design Artifacts:**

- **All levels:** Students generate ideas for new solutions incorporate ideas from peers into designs and reflect on how diverse perspectives affect plans and designs. Use tools like flowcharts, mind maps, storyboards, outlines, and decision trees to plan prototypes. Compare solutions to criteria and constraints. Use quantitative and qualitative data to choose a solution to prototype. Predict the performance and impacts of prototypes, including user needs, and accessibility.
- **Skill progression:** Students intentionally apply computational thinking and planning tools such as pseudocode with documentation, models, and decision trees to design solutions. Ask questions about data sources and outline methods for testing prototypes, including checking for errors, sources of bias, and alignment with design criteria. Evaluate and modify plans to address diverse user needs using empathy, feedback, and iterative refinement.

3. **Create, Communicate, and Document Solutions:**

- **All levels:** Students use plans to create artifacts such as algorithms, programs, and prototypes. Describe design choices and make connections to the design challenge, criteria, and constraints. Engage in giving and receiving feedback to refine solutions and enhance communication skills. Annotate their programs to explain design choices. (This known as "documenting code" in the computer science field.)
- **Skill progression:** Students also modify or remix parts of existing programs to develop their ideas, streamline designs, or add more advanced features and complexity. Provide documentation for end users that explains the functionality of artifacts. Seek input from broad audiences and intentionally apply iterative design. Develop detailed and clear comments, graphics, presentations, and demonstrations.

4. **Test and Optimize Artifacts:**

- **All levels:** Students test artifacts to ensure they meet criteria and constraints, comparing results to intended outcomes. Use computational thinking and other problem-solving strategies like trial and error to fix simple errors, debug, revise, and evaluate artifacts against design criteria. Reflect on how the iterative design and computational thinking practices facilitate program development.
- **Skill progression:** Students also employ systematic and iterative testing to identify and resolve issues and optimize program design. Anticipate and address potential errors and edge cases to improve reliability and functionality. Distinguish between syntax and logic errors. Identify strategies to improve implementation of the iterative design process.

Instructional Considerations for Collaboration Practices

Possible **instructional approaches** to foster iterative design practices:

1. Design learning experiences where students identify real-world problems and evaluate the appropriateness of using computational tools to develop solutions.

2. Provide instructional time and model strategies to support students with using an iterative process to plan the development of a computational artifact while considering key features, time and resource constraints, and user expectations. Design instructions to provide students with multiple paths to solve problems.
3. Provide instructional time for students to prototype**,** justify, and document computational processes and solutions using iterative processes. Model how to listen to differing ideas and consider various approaches and solutions.
4. Provide instructional time and model strategies for evaluating computational artifacts using systematic testing and iterative refinement to enhance performance, reliability, usability, and accessibility as outlined in the design criteria.

Instructional activities may include but are not limited to:

- **Class Debates:** Debate pros and cons of using computing technologies to solve real-world problems. Consider examples like drones monitoring the environment; AI-generated art; or personalized learning applications. Progressive examples include, using machine learning in self-driving cars to interpret road conditions and make decisions, and robots assisting in surgeries for precision and reduced recovery times.
- **Prototype and Improve:** Create simple animated stories; solve pre-existing problems; utilize coding platforms to simulate a solution; incorporate microcontrollers to create physical models. Use peer feedback to refine the design. Document changes and justifying improvements at each step.
- **Debug and Enhance:** Work with a pre-built program containing intentional errors and limited features to debug syntax and logic issues, optimize the program for performance, and enhance it with new capabilities.
- **Accessibility Upgrade:** Emphasize empathy and inclusion in design by analyzing an existing program or interface (e.g., a basic website). Evaluate it for usability and accessibility. Propose and implement iterative changes to improve the design, such as adding features like text-to-speech, adjustable font sizes, or simplified navigation.

D. Fostering Digital Literacy Practices

1. **Responsible Use Practices:**

- **All levels:** Students use technology in ways that are safe, legal, and ethical. Explore data privacy rights, data protections, terms of service and privacy policies. Implement strategies to protect their digital identity, personal data, and the data of others. Explore and ask questions about how computer science and emerging technologies work, and their benefits and risks. Consider the benefits and risks of sharing different types of information with different audiences.
- **Skill progression:** Students analyze data privacy rights, data protections, terms of service and privacy policies. Weigh tradeoffs and risks with actions and decisions involving computer science. Justify choices for responsible use of computing technology using accurate information of how current and emerging technologies work.

2. **Safeguard Well-Being of Self and Others:**

- **All levels:** Students reflect on their emotional response to the use of digital technology and identify how to use technology in ways that support personal well-being. Consider how the use of technology can impact others and make choices that benefit others and avoid harm. Identify the roles and responsibilities of humans in designing and using technologies and avoid anthropomorphizing tools like AI. Practice empathy and engage in positive, respectful online practices as an upstander.
- **Skill progression:** Students also reflect on their choices about technology use and assess how different technology support learning goals and the well-being of self and others. Practice setting boundaries for online communication.

3. **Evaluate Resources and Recognize Contributions:**

- **All levels:** Students apply strategies for evaluating the accuracy and relevance of digital sources. Begin to evaluate for reliability, credibility, perspective, limitations, appropriateness, and accessibility of digital sources. Keep track of sources of information and give credit to the creators of information. Identify false or misleading information.
- **Skill progression:** Students also evaluate validity, consistency, appropriateness for needs, data bias, importance, and social and cultural context. Consider assumptions, missing information, and perspectives. Identify and give attribution to ideas that are borrowed and modified, and check for licensing permissions.

Instructional Considerations for Collaboration Practices

Possible **instructional approaches** to foster digital literacy practices:

1. Model how to use technology in ways that are safe, legal, and ethical. Model how to make decisions about data privacy and information sharing that protect individual and peer identify and digital footprint. Incorporate learning activities like discussions of digital dilemmas that help students explore different perspectives, benefits, risks, and tradeoffs.
2. Incorporate opportunities for students to reflect on possible positive and negative impacts of how they use computing technologies. Choose instructional technology that aligns with learning goals and use data on students learning to reflect on and assess the extent to which the technology is supporting learning outcomes. Provide opportunities to identify the role of humans in developing and using technology and avoid anthropomorphizing tools like AI.
3. Model strategies for how to investigate the credibility of information sources and give appropriate attributions for content created by others.

Instructional activities may include but are not limited to:

- **Source Evaluation:** Assign students articles. Have students distinguish between fact and opinion within articles and evaluate the reliability of the sources.
- **Design Thinking:** Seek user-feedback throughout the design process to gain primary source information. Compare findings to predictions to help uncover and correct assumptions.
- **Tracing:** show **(**trace) where relevant evidence supports a claim, program, or model.

- **Comparative Analyses:** Encourage students to explore ethical dilemmas, compare different approaches to data privacy and possible impacts across different time periods using evidence to support arguments.
- **Class Debates:** Organize debates where students take on roles representing different perspectives and defend their positions.
- **Digital Dilemmas:** Discuss case studies of complex topics that do not have one right answer such as the CommonSense Education digital dilemmas.

Appendix B

Grade 8 Computer Science Vocabulary

Term	Definition
Abstract Data (ADT)	Models that use a set of possible values and operations to define data.
Abstraction	A filtering process used to create a simplified representation of relevant data to identify essential details, excluding less important details.
Acceptable Use Policy (AUP)	Rules and guidelines that define safe practices and responsible use of technology.
Accessibility	Refers to the design of products, devices, services, or environments for people with disabilities. These disabilities may include visual, auditory, motor, and cognitive disabilities.
Algorithm	Finite and specified set of step-by-step instructions designed to solve a problem or perform a task
American Standard Code for Information Interchange (ASCII)	A character encoding standard that represents text in numeric form, enabling multiple data representations in computing devices.
Application Software	A type of computer program designed to perform specific tasks for users. It is different from system software, which manages hardware and basic system operations.
Artificial Intelligence (AI)	A branch of computer science focused on the research and development of algorithms and systems that simulate tasks that typically require human intelligence, such as reasoning, learning, perception, and decision-making.
Assistive Technology	Any device, software, or system that is used to increase, maintain, or improve functional capabilities of a person with a disability.
Attribute	Characteristic or quality that helps us describe and differentiate objects or data (i.e. color, size, shape, weight, position, number, or texture).
Authentication	A process used to verify a user's identity before accessing a network or computer system.
Authorization	A process used to verify a user's level of access to a computer system.

Automated Decision Making	The use of predefined algorithms or programs that enable a computing device the ability to make decisions independently based on the information it receives.
Bias	Prejudice in favor of or against one thing, person, or group compared with another.
Binary	A number system that uses two digits, 0 and 1.
Block-Based Programming	A visual drag and drop programming tool that users can use to create programs using command blocks.
Bubble Sort Algorithms	Algorithms that compare consecutive items in a list and will switch the items that are in the wrong order.
Bug	An error or mistake in a program that results in unintended outcomes such as an incorrect response or crash.
Cipher	A secret or disguised way of writing; a code.
Classify	Is to arrange or organize a set of objects according to a predetermined category or attribute (a quality or characteristic).
Client	A computer or software application that retrieves and uses information, resources, or services from another device over a network.
Cloud Computing	Storing and accessing information on the internet.
Code	Any set of instructions expressed in a programming language.
Code Tracing	The practice of reading and analyzing a section of code and identifying the values of critical variables in the algorithm.
Collecting	Gathering the appropriate type of data needed.
Compound (or composite) Data	Data that combines two or more primitive data, such as a record that contains multiple variables. Sometimes called Composite Data.
Computational Artifacts	Any creation made by a human using a computing device. Can include but are not limited to prototypes, programs, planning documents, animations, or abstractions (e.g. visualizations, storyboards, flowcharts, decision trees, models, computer simulations).
Computational Thinking	A logical and systematic problem-solving process that uses decomposition, pattern recognition, abstraction, and algorithm thinking to foster creativity and develop solutions.

Computer	An electronic computing device that processes, stores, and retrieves data and capable of executing a wide range of tasks, from basic calculations to complex data processing.
Computer Science	The study of computers and algorithmic processes, including their principles, their hardware and software designs, their applications, and their impact on society.
Computer System	Integrated group of hardware and software that work together to store, process, and manage data.
Computing Device	Electronic tool or machine designed to receive, store, and process data to perform tasks.
Computing Technologies	Broad range of devices and tools that help us process information and perform tasks using computers and software.
Conditional Control Structures	Conditional logic (e.g., if-else statements) to make decisions within a computer program.
Conditions	A set of operators or conditions such as "if" or "and" that gives a program a path to follow when executing a programmed task.
Control Structures	The parts of a program that specify the order in which instructions are executed. Control structures can analyze variables within a program code.
Cookies	Small text files that track a user's browsing history on a website along with information about their geographic location, browsing software, and operating system.
Crash	When computer software, hardware, or programming stops functioning unexpectedly due to damaged parts, faulty coding, or a virus.
Cyberattack	An attack that targets computing devices, networks, software, or users to disrupt operations, gain unauthorized access, or steal, alter, or destroy data.
Cybersecurity	Protection of data and information on networks and computing devices from unauthorized access, attacks, damage, or theft.
Data	Individual pieces of information about people, things, or events that can be processed, stored, and analyzed by computing devices.
Data Center	Facilities that house servers and store vast amounts of information. They keep the Internet running by providing the physical space and equipment needed to store and process data.

Data Cycle	Process of formulating questions to be explored with data, collecting or acquiring data, organizing and representing data, and analyzing and communicating results.
Data Visualization	The representation of data through use of common graphics, such as charts, plots, infographics and even animations to make complex data more accessible and understandable.
Database	Using online databases (e.g., academic, scientific) to find structured data and studies.
Debug	Process of identifying, isolating, and fixing errors (often referred to as "bugs") in a set of instructions, code, or system. This can also include hardware and software.
Decision Tree	A mathematical model that uses numerous questions and answers to analyze and classify data before predicting a result. Decision Tree machine learning is a supervised approach that uses a defined set of values to simplify data into a tree-like structure to answer a question or make a prediction.
Decomposition	Process of breaking down a problem, process, or task into smaller, more manageable components.
Decryption	The process of converting encrypted data back into its original, readable form.
Design	Creation of a plan or prototype of a proposed solution.
Design Document	A detailed plan that outlines the structure, features, and implementation strategy of a project. It serves as a blueprint, providing clear specifications, goals, and guidelines for developers, designers, and stakeholders. Design documents often include diagrams, technical requirements, workflows, and rationale to ensure a shared understanding of the project's direction and execution.
Digital Citizenship	The rights, responsibilities, and opportunities of living, learning, and working in a interconnected digital world. This promotes responsible and ethical behavior in digital environments, including understanding data privacy, security, and the impact of digital actions.
Digital Literacy	The ability to use technology effectively and responsibly to access, evaluate, create, and communicate information.
Diverse	Variety of different elements, qualities, or characteristics.
Documentation	The act of annotating the program to explain the purpose of each section of code, also known as commenting code.
Domain Name System (DNS)	A “phonebook” for the Internet, translating website names (like google.com) into IP addresses that computers use to locate each other.
Email	A program used to send and receive messages over the Internet for online communication.

Email Server	Software applications that manage email exchange between users. They store, send, and receive email messages, enabling online communication.
Emerging Technology	New or developing technologies that are transforming.
Encode	Convert (information or an instruction) into a particular form.
Encryption	The conversion of electronic data into another form, called ciphertext, which cannot be easily understood by anyone except authorized parties.
Ethernet Cable	A type of network cable used to connect devices to create a local area network.
Evaluation	Assessment process that reviews test results and feedback to determine a design or product's effectiveness and identify necessary changes for improvement.
Expression	In a programming language, a combination of explicit values, constants, variables, operators, and functions interpreted according to the particular rules of precedence and of association which computes and then produces (returns, in a stateful environment) another value.
Extraction	Process of identifying, isolating, or deriving meaningful information.
Firewall	A network security system with rules to control incoming and outgoing traffic, providing security by blocking potentially harmful traffic from reaching a network.
Flowchart	A diagram that shows the steps in a process using shapes and arrows. It helps the user visualize how things happen in order.
Function	Like variables, except instead of storing data they store lines of code. Help to simplify the programming process and make code more readable.
Hardware	The physical components of a computing device that you can touch, such as the processor, memory, keyboard, and display.
Hypertext Transfer Protocol Secure (HTTPS)	A secure protocol used by a web browser application to transmit information from a website to a user.
Implementation	The development or execution of a functional prototype, program, or product.
Information	Facts provided or learned about something or someone.

Input	Data that is entered or received by a computing device for processing.
Intellectual Property Rights (IPR)	Legal protections granted to creators and inventors for their original works, designs, and inventions.
Internet	A global network of interconnected computing devices that allows devices to share information and resources.
Interpreter	A type of computer program that executes code line by line, translating it from a high-level programming language into machine code at runtime.
Iteration	Repeated actions.
Key	A distinct identifier used to differentiate data elements within a set.
Large Datasets	Collection of large amounts of data that require specialized tools or methods to store, process, and analyze.
Linear Regression	A mathematical model that analyzes linear relationships and variables to predict a result. This is a supervised learning method that uses labeled data to map points in an effort to predict future results.
Link	The connection between two different nodes that represent the data connection.
List	A data structure that stores an ordered collection of elements, which can be of any type (numbers, strings, objects, etc.).
Local Area Network (LAN)	A computer network that covers a confined area, such as an office building, university, or home.
Logic Errors	An error that occurs when a program is executed and produces incorrect or unintended output without causing the program to crash or display an error message.
Loop	A set of instructions that are repeated until a specified condition is met, or a predetermined number of repetitions has occurred.
Machine Learning	A process that occurs when computers learn from examples instead of following exact instructions. Examples of machine learning include supervised learning, unsupervised learning, and reinforcement learning.
Malware	Malware is malicious software that can steal data, corrupt files, disrupt services, and/or damage networks and computing systems.
Mathematical Model	A way for computers to predict a result or model the real world by using a set of given mathematical rules and data.

Media Access Control (MAC) Address	A unique hardware identifier assigned to each device on a network. The MAC address is embedded in the network card during manufacturing and cannot be altered.
Memory	The physical storage in computing devices where data is processed and instructions for processing are stored. Memory types include RAM (Random Access Memory), ROM (Read-Only Memory), and secondary storage like hard drives, removable drives, and cloud storage.
Merge Sort Algorithms	Algorithms that split data lists into halves called sublists repeatedly until there is only one item in each sublist.
Models	Simplified representation or abstraction of a concept, object, system, or process.
Modem	A device that connects a home or local network to the Internet. They convert data from digital form (used by computers) to a form that can travel over phone lines, cable lines, or fiber optics, allowing devices to access the Internet.
Modularity	A characteristic in programming where subcomponents (modules) with specific functions are identified and incorporated into the code or program.
Naming Convention	Guidelines for the use of descriptive names for variables, functions, and classes.
Nested Conditional Control Structures	Conditional statements placed inside the body of another conditional statement.
Network	Two or more computers that are connected together to communicate and share information.
Neural Network	A specific method of Artificial Intelligence (AI) that uses a network of computers to process data and produce an output in a way that imitates the human brain.
Node	Node is a computing device that is connected to a network and is able to communicate with other devices.
Non-Numeric Data	Refers to data that involves categories, qualities, or descriptions rather than numbers such as name, address, and favorite color.
Numeric Data	Refers to data that involves numbers and can be counted, measured, or quantified such as age, weight, or height.
Operating System (OS)	The software that manages hardware resources and provides a user interface. It enables the computer to run programs and ensures everything works together smoothly.
Operator	A symbol that establishes a relationship between two values. Common types include logical (AND, OR, NOT), relational (=, <, ≤, >, ≥), and arithmetic (+, -, ÷, ×) operators.

Output	Data or information produced by a computing device after processing input.
Packet	Packets are smaller pieces of data that can be sent across networks.
Parameter	Parameters are coding structures that identify the values that will be transferred when called.
Password	A secret word or phrase used to protect devices and information from unauthorized access.
Pattern Analysis	Process of identifying commonalities, differences, and predictable relationships within data to understand, interpret, and make predictions.
Pattern Recognition	Ability to identify commonalities, similarities, or differences in recurring elements.
Patterns	A repeated sequence or behavior that is observable in data, objects, or events.
Personal Identifiable Information (PII)	Any data that can be used to identify a specific individual. It includes both direct and indirect identifiers that, alone or combined, can reveal a person's identity.
Personal Information	Data or information about a person that relates to their identity, characteristics, or activities.
Phishing	Deceptive online attack where scammers pretend to be a trusted source and trick individuals to share personal identifiable information.
Pixel	Picture element or tiny square of color that, when combined with other pixels, comprises a larger image.
Plain Language	A description of the steps and logic in simple terms that anyone can understand through the use of familiar analogies, real-life examples, and simple terms.
Prediction	An educated guess about what will happen next, based on current facts or what is already known.
Primitive Data	Basic forms of data that can store single values.
Probeware	Scientific equipment that combines sensors (hardware) with software to collect scientific data (e.g. light, temperature, distance, motion).
Problem Definition	Clearly identifying the problem or challenge that needs to be solved.
Procedure	Broadly used to refer to a process, which may include a method, function, subroutine, or module, depending on the programming language.
Processing Speed	How fast a computer can take in information, think, and complete tasks.

Program	The implementation of an algorithm (set of instructions) translated into a programming language that a computer can follow and execute to perform a specific task.
Programming Language	Formal system of communication used to write instructions that a computer can execute. It consists of syntax (rules for structuring code) and semantics (meaning of the instructions). Programming languages allow developers to create software, automate tasks, and control hardware.
Proprietary Software	Software that is owned by an individual, company, or organization and is subject to restrictions on its use, modification, and distribution. The source code is typically not available to the public, meaning users cannot inspect, modify, or freely share it.
Protocols	Agreed upon rules that define how messages between computers are sent. They determine how quickly and securely information is transmitted across networks and the Internet, as well as how to handle errors in transmission.
Pseudocode	An algorithm written in plain language instead of a programming language.
Public Information	Information that is okay to share with anyone and is typically available for everyone to see.
Quick Sort Algorithms	Algorithms that can quickly and efficiently organize data in ascending or descending order by splitting the data into smaller pieces for processing.
Ransomware	A type of malicious software (malware) that encrypts a victim's data or locks them out of their system, demanding payment (a ransom) to restore access. It is commonly spread through phishing emails, malicious attachments, or software vulnerabilities.
Reinforcement Learning	A type of machine learning where a computer learns through trial and error. The computer receives feedback and adjusts its behavior to improve performance.
Reusability	A characteristic in programming where a subcomponent of a program has a clear, well-defined purpose that makes it possible to use it repeatedly in different parts of the program.
Robotics	Using robots to perform repetitive tasks with precision.
Router	A device that directs data from one network to another.
Routine	A series of activities or tasks that someone does regularly. A routine is like a special task or a set of instructions that a computer follows to do something over and over again. It is a way of organizing work so that the computer knows exactly what to do without having to be told every single time.

Screen Time	Time spent on a computing device.
Search Algorithm	A program or set of instructions that allows a user to find specific information with a defined set of data. Its role is to allow a user to search for the desired information.
Search Engines	A program that enables users to find information on the Internet.
Sensor	A computing component that detects, collects, or measure data that would otherwise be difficult to gather manually.
Sequence	The specific order in which instructions or steps are executed in an algorithm or program.
Server	A computer or software application that provides information, resources, or services to clients over a network.
Simulation	Replicating the behavior of a real-world process or system over a period of time.
Social Engineering	The process of bypassing computer security and gaining access to a device or system by tricking users into revealing passwords or other secure information.
Social Media	An application or website that an individual uses to communicate with other individuals in their community or around the world.
Software	A set of instructions that tells the computer how to act and respond but cannot be seen or touched.
Software Applications	Programs designed to perform specific tasks for the user, such as word processing, web browsing, gaming, or photo editing. These applications run on the operating system and make the device useful for various functions.
Sort	Used to compare a set of objects in order to find similarities and differences, so that they may be arranged and organized.
Spoofing	A method of copying, mimicking, or pretending to be a trustworthy source so that bad actors can use it for malicious purposes.
Spyware	A type of malware designed to secretly monitor and collect information about a user's activities without their knowledge or consent.
Storage	Location where data, programs, and files are kept permanently (until deleted).
Supervised Learning	A method of machine learning that involves a computer using large amounts of labeled datasets to recognize patterns, classify data, and make predictions.

Switch	Devices that act as a bridge within a network and connect multiple devices (like computers, printers, and routers) on a LAN (local area network) network allowing them to send and receive data to communicate directly with each other.
Syntax	Rules or structure of a programming language.
Syntax Errors	An error caused by a mistake in the rules or structure of a programming language, such as missing parentheses, commas, or incorrect keywords.
Table	A structured format to organize and record information in rows and columns.
Testing	The process of evaluating a program or system to assess its results and outputs, ensuring accuracy, performance, and reliability, while identifying errors.
Topology	The physical and logical configuration of a network; the arrangement of a network, including its nodes and connecting links.
Training Data	A collection of labeled or unlabeled data used to teach a machine learning model how to recognize patterns, make predictions, or perform tasks.
Translation	Conversion of symbolic representations or programming language into programming language.
Transmission Control Protocol/Internet Protocol (TCP/IP)	A framework of rules that allow a device to connect to and communicate with other devices on a network.
Trends	Are long-term directions or movements in data or behavior that indicate a general tendency or shift over time.
Troubleshoot	Process used to diagnose why a system or process is not working as expected and systematically testing solutions to resolve the issue.
Two-Factor Authentication (2FA)	Security mechanism that requires two types of credentials to verify authorization.
Unauthorized Access	Information is accessed without the permission of the owner.
Unsupervised Learning	When a computer learns to find groups or patterns without anyone telling it what's right or wrong.
User Input	Data or information that a user provides to a computer program during its execution (2024). Data that is taken in by a computer for processing (2017).

Username	A unique name that people use to log into a device or online account. It is like a nickname that helps the computer recognize who is logging in.
Variables	A programming element that is a named storage location in memory that holds a value, which can be modified during the execution of a program.
Version Control	A process in software design and programming where changes are tracked by documenting the improvements and incrementally changing the version number identifier.
Virtual Private Network (VPN)	A secure and encrypted network connection that creates a direct connection to a remote router, masks or hides the connection information, and encrypts the data that it transmits.
Virus	Malicious software (or malware) designed to spread from one computer to another, often without the user's knowledge.
Visualizations	Refer to graphical representations of data or information that help users understand patterns, trends, and relationships more effectively. Visualizations make complex data more accessible, interpretable, and actionable.
Web Browser	Software programs, like Chrome, Firefox, or Safari, that allow users to access and interact with websites and online content.
Webpage	A single document on the internet that is accessed through a web browser.
Websites	A collection of webpages on the Internet that people can visit using a web browser.
Wi-Fi	The device that allows computing devices to access the Internet without being connected to physical cables within a specific area using radio waves to send and receive data.
Workstation	A type of node that is typically a user's computer.
Worms	A type of malware designed to replicate itself and spread across computers or networks without needing to attach to a host program or file.