2024 Computer Science Standards of Learning

Grade 7 Instructional Guide

VIRGINIA DEPARTMENT OF EDUCATION

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

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 of Learning* 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.

Seventh Grade: Computer Science *Standards of Learning*

In Seventh Grade, students delve into the complexities of physical and digital security measures, critically evaluating threats and vulnerabilities associated with Internet use. They hone their programming skills through continued construction of algorithms, sequences, loops, and functions. Students leverage computational thinking and programming skills while applying an iterative design process to create programs as a means of creative expression and innovation. Through the use of purposeful computing devices students will collect data and define patterns, make inferences, and continue to develop computational models. Through the application of computational thinking, students will resolve hardware and software issues methodically. Students will deepen their understanding of concepts like machine learning, broadening their understanding of advanced technologies and their applications. Students will examine and evaluate the impact of computing technologies within society and globally. Additionally, students will identify individual strengths that can be used within computer science careers.

Algorithms and Programming (AP)

7.AP.1 The student will apply computational thinking to design 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. Contrast various algorithms to solve reasoning problems when accomplishing a task.

7.AP.2 The student will plan and implement algorithms that include sequencing, loops, variables, user input, conditional control structures, and functions using a block-based or text-based programming tool.

a. Describe the concept of functions for use in a computer program.
b. Plan an algorithm using plain language, pseudocode, or diagrams.
c. Read and write programs that collect and use numeric and text data from users.
d. Read and write programs that contain nested conditionals and nested loops.

7.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.

7.AP.4 The student will apply proper attribution when incorporating other sources into original work.

a. Apply proper methods of attribution when using work from the Internet and other sources.
b. Incorporate information or assets from the Internet into a program with proper attribution.

Computing Systems (CSY)

7.CSY.1 The student will design projects that use computing devices to collect and exchange data.

a. Apply project management skills to distribute tasks and maintain project timeline.
b. Generate ideas combining hardware and software components that can be used to collect and exchange data.
c. Describe how hardware and software can be used together to collect and exchange data.
d. Evaluate the usability of hardware and software to collect and exchange data.
e. Select the hardware and software components for project designs by considering factors such as functionality, cost, size, speed, accessibility, and aesthetics.

7.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. Compile and record successful methods used to resolve problems for common hardware, software, and connectivity-related problems.

Cybersecurity (CYB)

7.CYB.1 The student will differentiate physical and digital security measures that protect electronic information.

a. Compare and contrast physical and digital security measures.
b. Research and synthesize the tradeoffs between usability and security.
c. Identify common threats and vulnerabilities associated with Internet use and Internet-based systems.

d. Identify potential solutions to address common threats and vulnerabilities.

Data and Analysis (DA)

7.DA.1 The student will utilize computational tools to visualize and evaluate data to draw conclusions and make predictions.

a. Develop computational models that simulate real-world phenomena, considering relevant variables and relationships.
b. Refine and modify computational models based on observed data and feedback, ensuring alignment with empirical evidence.
c. Analyze patterns and trends within observed data, comparing them with the predictions made by computational models.
d. Evaluate the effectiveness and accuracy of computational models in capturing and explaining the observed data.

7.DA.2 The student will explain the process and application of computational thinking in machine learning.

a. Explain how supervised, unsupervised, and/or reinforcement learning methods utilize decomposition, pattern recognition, abstraction, and algorithms to learn from and make decisions.
b. Explore neural networks and its role in machine learning and artificial intelligence.

Impacts of Computing (IC)

7.IC.1 The student will assess the national and global impact of computing technologies.

a. Discuss specific examples of how computing technologies have influenced various national and global industries and sectors.
b. Analyze the implications of emerging technologies and potential real-world impact nationally and globally.
c. Evaluate the environmental impact of computing technologies nationally and globally.

7.IC.2 The student will describe and explain the impact of screen time on interactions with others.

a. Describe the positive and negative impact of social media on socialization.
b. Research the type of data collected on social media and online platforms that monitor social interactions.
c. Describe and explain the evolution of screen time and the impact it has had on social interactions.
d. Create a social media usage plan that demonstrates safe practices, meaningful use, and a balanced approach.

7.IC.3 The student will identify individual preferences, skillset, and experiences and determine how these relate to a chosen computer science career field.

a. Use a career interest assessment to identify and categorize preferences, skillsets, and experiences.
b. Evaluate and connect personal skillsets, interests, talents, and values to a computer science career.

7.IC.4 The student will identify and apply strategies to prevent personal and public works from being pirated and plagiarized.

a. Discuss and describe intellectual property protections.
b. Research and list safeguards used to prevent intellectual property infringement.

7.IC.5 The student will evaluate the effect of Artificial Intelligence (AI) in various professions.

a. Research AI integration in various professions and evaluate its impact on the job market and society.
b. Examine and analyze the impact on job creation and changes in employment needs based on the use of AI.
c. Evaluate and explain the benefits and drawbacks of the implementation of AI technologies in various professions.

Networks and the Internet (NI)

7.NI.1 The student will describe and explain why protocols are essential in data transmission.

a. Define packet, router, and protocol.
b. Describe the process of sending a file through a network.
c. Explain the role of Internet Protocol (IP) addresses in transmitting information.
d. Explain how packets ensure reliable communication among computing devices.
e. Model how data is transmitted over networks and the Internet.

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.

Seventh Grade: Computer Science Instructional Guide

In Seventh Grade, students delve into the complexities of physical and digital security measures, critically evaluating threats and vulnerabilities associated with Internet use. They hone their programming skills through continued construction of algorithms, sequences, loops, and functions. Students leverage computational thinking and programming skills while applying an iterative design process to create programs as a means of creative expression and innovation. Through the use of purposeful computing devices students will collect data and define patterns, make inferences, and continue to develop computational models. Through the application of computational thinking, students will resolve hardware and software issues methodically. Students will deepen their understanding of concepts like machine learning, broadening their understanding of advanced technologies and their applications. Students will examine and evaluate the impact of computing technologies within society and globally. Additionally, students will identify individual strengths that can be used within computer science careers.

Algorithms and Programming (AP)

7.AP.1 The student will apply computational thinking to design 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. Contrast various algorithms to solve reasoning problems when accomplishing a task.

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).

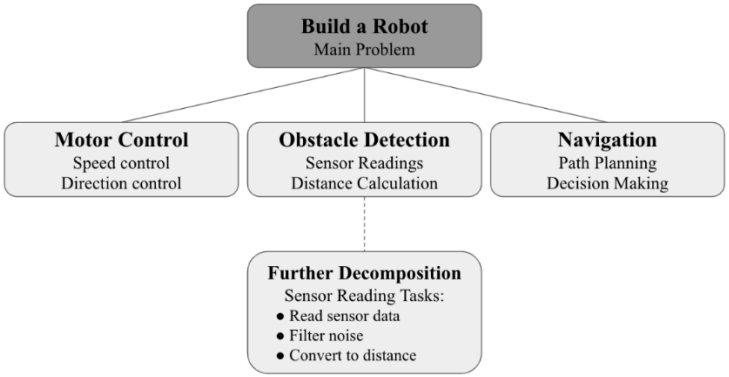
Teachers should intentionally design learning experiences that foster the practical application of computational thinking, such as hands-on programming projects and complex problem-solving tasks that reflect authentic, real-world contexts. The creation of meaningful, developmentally appropriate tasks 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.

[7.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.

[7.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.



[7.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.

[7.AP.1d] The ability to design algorithms using abstraction requires one to identify the essential information needed to complete the task and omit unnecessary details. This abstraction enables the creation of structured, high-level representations such as pseudocode, flowcharts, or modular functions that illustrate the logic of the process without relying on low-level implementation specifics. Teachers can guide students to define key inputs, outputs, and conditions, then organize that distilled information into a complete and logically ordered sequence of steps. This approach simplifies complex problems and supports the development of scalable, adaptable, and reusable solutions. Algorithm design through abstraction promotes modularity, enhances readability, and aligns with best practices in computational problem-solving and software engineering, such as hierarchical decomposition and separation of concerns.

To further develop algorithmic thinking, it is essential to engage learners in evaluating and contrasting multiple algorithmic approaches to the same problem, particularly those that require reasoning and decision-making. Through comparative analysis, students learn to consider trade-offs in efficiency, clarity, scalability, and resource usage, fostering a deeper understanding of how abstraction shapes design choices.

Reasoning problems often have multiple algorithmic solutions, each with varying levels of efficiency, scalability, and implementation complexity. Selecting the most appropriate algorithm can significantly affect computational performance. For example, when calculating the sum of numbers, one algorithm might add them sequentially, while another could group and add them in pairs. By comparing these methods, students can determine which is faster or easier to implement depending on the size of the problem or context.

- Sequential Addition:
  - $1 + 2 + 3 + 4 = 10$
  - (Each number is added one by one in sequence.)
- Pairwise Addition:
  - $(1 + 4) + (2 + 3) = 10$
  - (Numbers are grouped and added in pairs, which can make the process faster for larger datasets.)

Solving reasoning problems such as sorting numbers, solving equations, or determining the most efficient way to complete a task often requires comparing multiple algorithms to identify the most effective approach. This involves analyzing algorithmic efficiency using metrics like time and space complexity to determine optimal performance under given constraints.

Encouraging students to reason through multiple algorithmic paths also supports computational decision-making, where selecting the most effective solution depends not only on correctness but on how well the algorithm meets contextual demands such as execution speed, memory constraints, or simplicity of implementation. This critical comparison cultivates both algorithmic fluency and computational literacy, preparing students to design solutions that are not only functional but optimized and justified through reasoned evaluation.

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 such as decomposition, pattern recognition, abstraction, and algorithmic thinking to design programs for creative expression or scientific exploration. These skills are not confined to computer science; they enhance problem-solving and critical thinking across various subjects, fostering a cohesive and interdisciplinary approach to learning.
Vertical Progression: In Grade 6, Students expand on abstraction by designing algorithms for the purposes of accomplishing a task or expressing a computational process.(6.AP.1). In Grade 8, Students will apply all they have learned about decomposition and abstraction to illustrate complex problems as algorithms. (8.AP.1)

ACROSS CONTENT AREAS

Mathematics

- **7.CE.1** The student will estimate, solve, and justify solutions to multistep contextual problems involving operations with rational numbers.

Science

- **7.1** The student will demonstrate an understanding of scientific and engineering practice. *7.1 standard is integrated within science content and not taught in isolation. Potential science concepts to apply 7.1 include: LS.2.e:The student will model how materials move into and out of cells in the processes of osmosis, diffusion, and selective permeability.*

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.
- **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.
  C. Students break problems into component parts, identify key pieces and use that information to problem solve using technologies, when appropriate.
- **6-8.CC** Students communicate clearly and express themselves creatively for a variety of purposes using appropriate technologies (including assistive technologies), styles, formats, and digital media appropriate to their goals.
  D. Students select and use appropriate technologies to design, publish, and present content that effectively convey their ideas, conclusions, and evidence for specific audiences.

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 flowcharts or pseudocode to plan the structure of their expository writing, mapping out their introduction, body paragraphs, and supporting details in a logical sequence before drafting. This mirrors the decomposition and organization process used in programming.

History and Social Science

- Students create an interactive simulation that models economic decision-making based on scarcity and opportunity cost. Using block-based or text-based programming, they can design a program where users make choices about resource allocation (e.g., budgeting for a town or business) and see the outcomes of their decisions.

Mathematics

- Students design a step-by-step problem-solving program using block-based coding to estimate and solve multistep contextual problems involving rational numbers. They can input equations, have the program show step-by-step solutions, and justify the final answer with explanations.

Science

- Students create an animation or interactive model of mitosis using a block-based programming tool. They will program step-by-step transitions between the phases, illustrating key processes like chromosome alignment and cell division. Alternatively, they can design a flowchart or decision tree to model how cells progress through each stage of mitosis.

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 Complex Real-World Problems
2. Explore Common Features, Relationships, and 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 Computational Artifacts
3. Create, Communicate, and Document Solutions

7.AP.2 The student will plan and implement algorithms that include sequencing, loops, variables, user input, conditional control structures, and functions using a block-based or text-based programming tool.

a. Describe the concept of functions for use in a computer program.
b. Plan an algorithm using plain language, pseudocode, or diagrams.
c. Read and write programs that collect and use numeric and text data from users.
d. Read and write programs that contain nested conditionals and nested loops.

Understanding the Standards

Students extend their understanding of algorithms by developing more complex programming solutions. These solutions incorporate sequencing, iteration, variables, user input, conditional statements, and modular design through functions. Block-based and text-based programming environments are used to implement these components. This progression strengthens algorithmic design and supports the development of computational problem-solving skills.

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.

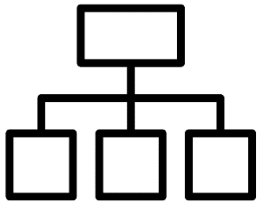
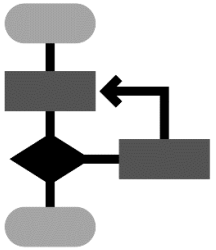
- Sequence is the specific order in which instructions are executed in a program. The flow of control refers to this order, and incorrect sequencing can affect whether the program runs properly
- Loops are the repeated execution of a block of code until a certain condition is met (e.g., "for" loops or "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 or numbers.
- 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.”

[7.AP.2a] Functions are blocks of code that perform specific tasks and can be reused throughout a program. Functions divide complex problems into smaller, manageable parts and provide a modular representation of program logic. Functions define clear sections of code that support structured organization and consistent execution. A function is typically defined once and can be called or invoked multiple times throughout a program, often with different inputs, known as parameters. When executed, the function processes the inputs and may return an output, allowing for consistent and predictable behavior.

In computer science, functions serve as foundational tools for modeling relationships among variables and values, enabling the representation of how changes in one element influence others within a computational system. This mirrors the role of mathematical functions, where a defined input produces a corresponding output based on a specific rule or transformation. By engaging students in the design and use of functions, educators foster an understanding of abstraction, modular design, and data flow, which are core principles that underpin scalable and maintainable software development. Instruction in functions not only supports problem decomposition but also equips learners with the structural thinking necessary to construct and manage increasingly complex programs and computational systems.

[7.AP.2b] Planning is a vital step in programming that begins with a design document. As students generate and organize ideas and algorithms using design documents (e.g., planning, drafting, revising, editing), they learn to develop and organize their algorithms. Students should be given a variety of opportunities to plan (e.g., outline, brainstorm) and draft (e.g., compose a first draft anticipating mistakes and corrections) using plain language, pseudocode, or diagrams to create their design documents. They also revise (e.g., add, remove, or rearrange ideas) and edit (e.g., proofread for grammar, punctuation, and spelling) their design documents to ensure that the algorithm is clear, sequential, and that their variables work as intended.

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 

Design documents are planning documents that outline the steps and/or processes required to create a product, including the problem statement, input and output requirements, and algorithmic procedures. They can include descriptions of the problem being solved, the input and output requirements, and the step-by-step processes the algorithm will follow. Design documents can be in the form of flowcharts, diagrams, or story maps that students use to brainstorm the algorithm they want to create.

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

[7.AP.2c] Programs consist of structured code designed to execute specific tasks. These tasks may include performing arithmetic operations, modifying or organizing data, or generating content. Programs often rely on user input during execution. Input is typically captured as text and stored in variables for subsequent operations.

Numeric and textual input require appropriate data types. Strings represent sequences of characters, while integers and floating-point numbers represent numerical values. Correct data typing ensures input can be processed accurately.

Managing input-driven behavior requires understanding variables, control structures, and data flow. Program design must account for how user input is captured, validated, stored, and used in algorithmic operations.

Data context determines how values are stored, interpreted, and used in a program. A single data value may serve different purposes depending on how it is applied. For example, a student's age may be treated as numeric data when performing calculations, such as computing averages or analyzing distributions. The same value may be stored as text when used in labels, grouped categories, or descriptive reports where no arithmetic operations occur.

Selecting the appropriate data type based on context ensures accurate processing, efficient storage, and reliable output. Misinterpreting data types can result in logic errors, formatting issues, or incorrect program behavior.

- Numeric data consists of numbers, such as integers and decimals, that are used in mathematical calculations. A numerical expression includes only numbers, operation symbols (like +, −, ×, ÷) and grouping symbols (such as parentheses) to define the order of operations.

- Text data is non-numeric data that refers to sequences of characters, including words and sentences, that are primarily used for displaying or processing information rather than for numerical calculations.

[7.AP.2d] Nested control structures are programming constructs that place one control structure inside another, allowing a program to evaluate multiple conditions or perform repeated actions within a defined hierarchy. These structures include nested loops and nested conditionals, which extend the logic and functionality of a program by introducing layered decision-making and repetition.

A conditional structure evaluates whether a specific condition is true or false and executes a corresponding block of code. A nested conditional is an "if-else" or "if-elif-else" structure placed inside another conditional, enabling a program to check additional criteria only when a previous condition is met. Nested conditionals are used when outcomes depend on multiple, interrelated conditions.

A loop structure repeats a set of instructions while a specified condition holds true. Nested loops involve placing one loop inside another, allowing the program to perform a repeated action within each iteration of an outer loop. This structure is often used for tasks such as traversing multi-dimensional arrays or generating patterns. Nested control structures support multi-level processing and are essential for building complex logic into programs.

Nested conditional example: a student choosing a ride at a theme park might rely on both height requirements and wait time, requiring a layered conditional structure to guide the decision.

python example:

```
if height >= 48:
    if wait_time < 30:
        print("Go on the roller coaster!")
    else:
        print("Try a shorter line, like the bumper cars.")
else:
    print("You can ride the carousel or the mini train.")
```

Nested loops are control structures in which one loop is placed inside another loop. The inner loop runs through all its iterations for each single iteration of the outer loop. This structure is used when a program must perform a repeated action within another repeated action.

Nested loops are essential for solving problems that require multi-level iteration, such as processing data arranged in a two-dimensional format like grids, tables, or matrices. They are commonly used in tasks that involve traversing rows and columns, generating structured patterns, or manipulating nested data sets.

python example:

```
for row in range(3):        # Outer loop for rows
    for col in range(3):    # Inner loop for columns
        print(f"Row {row}, Column {col}")
```

Teachers should guide students in understanding the flow of execution in nested loops, emphasizing that the inner loop completes all of its iterations for each single iteration of the outer loop. It is also important to highlight that nested loops can significantly increase the total number of iterations, which may impact program performance. To support logical accuracy and efficiency, students should be encouraged to trace their loops systematically, checking for unintended repetition, infinite loops, or logic errors resulting from incorrect loop bounds or conditions.

Nested Conditionals	Nested Loops
Used to evaluate multiple criteria within another conditional statement, enabling more complex decision-making.	A loop within another loop is useful for processing multi-dimensional data structures (e.g., tables or arrays).
Example: if temperature > 80: if weather == "Sunny": print("Go to the beach.") else: print("Stay indoors.") else: print("Wear a jacket.")	Example: for row in table: for column in row: print(column) # Process each element in the table

Nested conditionals and nested loops when used effectively, allow programs to make layered decisions by evaluating multiple interdependent criteria. These structures are particularly useful when one decision is dependent on the outcome of a previous condition. By organizing logic in this hierarchical manner, programmers can condense code, making it more efficient, concise, and easier to maintain.

At the core of nested conditionals are relational expressions. Relational expressions are comparisons between two or more values to determine if the result is true or false. Some relational operators are outlined in the chart below:

Operator	Meaning
>	Greater than
<	Less than
==	Equal to
!=	Not equal to

Relational expressions compare two values using operators such as equals (==), greater than (>), or less than (<) to produce a Boolean result of true or false. These expressions control the flow of execution by determining whether specific code blocks should run.
Logical expressions are constructed by combining relational expressions using Boolean operators such as and, or, and not. These operators enable compound decision-making by evaluating multiple conditions simultaneously. Understanding how logical expressions influence program behavior is essential for analyzing and predicting the outcome of conditional statements and overall control flow.

[7.AP.2e] Correct structure and indentation are essential when implementing nested constructs in text-based programming languages. In languages like Python, indentation is not just stylistic but determines the execution of code blocks. Improper indentation can result in syntax errors or unintended program behavior.

Code readability and clarity are necessary when using nested conditionals or loops. Excessive nesting can reduce comprehension and increase the complexity of identifying logical errors. Structured formatting and clear labeling of each control block support maintainability and reduce the likelihood of implementation errors.

Concepts and Connections

CONCEPTS

Algorithms should be planned using plain language, pseudocode, or visual diagrams to outline logical steps and clarify the overall structure of the program before implementation. Once the logic is established, programs can be written to accept and process both numeric and textual input from users, enabling interactive and data-driven functionality. Functions are reusable blocks of code that perform specific tasks and help promote modularity, organization, and easier debugging. As programs become more complex, the use of functions, along with nested conditionals and nested loops, supports advanced control flow and multi-layered decision-making, resulting in efficient and well-structured code.

CONNECTIONS

Within the grade level/course: At this grade level, students plan and implement algorithms incorporating sequencing, loops, variables, user input, conditional control structures, and functions using block-based or text-based programming tools. These computational skills are integral across various subjects, enhancing students' ability to organize, analyze, and interpret information systematically.
Vertical Progression: In Grade 6, students plan and implement algorithms that include conditional control structures and collection of numeric data using a block-based or text-based tool. (6.AP.2). In Grade 8, Students will plan and implement algorithms that include sequencing, loops, variables, user input, conditional control structures, functions, and various data types (8.AP.2).

ACROSS CONTENT ALIGNMENT

English

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

Science

- **7.1** The student will demonstrate an understanding of scientific and engineering practice. *7.1 standard is integrated within science content and not taught in isolation. Potential science concepts to apply 7.1 include: LS.4.b:The student will explain how the processes of photosynthesis and cellular respiration serve to make energy available for life processes within living systems.*

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.

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 flowcharts or pseudocode to outline their writing process, breaking it into steps such as brainstorming, drafting, revising, and editing. This mirrors how algorithms are planned and structured in programming, helping students organize their ideas systematically before writing a multi-paragraph text.

Mathematics

- Students use block-based coding or text-based programming language to create a probability simulation that models real-world scenarios, such as rolling dice or flipping coins. The program will allow students to input the number of trials, run simulations, and compare the experimental probability with the theoretical probability. By using loops and conditionals, students can analyze patterns and understand probability concepts more interactively.

Science

- Students design an interactive program or animation that models the steps of photosynthesis and cellular respiration. The program could use sequencing and conditional control structures to simulate how plants absorb sunlight, convert energy, and release oxygen, helping students visualize the interconnected processes.

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 Collaboration Tools and Develop Project Management Practices

B. Fostering Computational Thinking Practices:

1. Decompose Complex Real-World Problems
2. Extract Common Features, Relationships, and Patterns
3. Apply Abstraction to Simplify, Visually 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 Computational Artifacts
3. Create, Communicate, and Document Solutions
4. Test and Optimize Computational Artifacts

B. Fostering Computational Thinking Practices:

1. Responsible Use Practices
2. Safeguard Well-Being or Self and Others

7.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.

[7.AP.3a] Programs that contain multiple control structures support the development of complex, interactive behaviors. These components determine the program's ability to make decisions, repeat operations, and execute defined procedures. Combining control structures enables programs to respond dynamically to user input, system state, or variable conditions.

Creating and testing such programs allows for the evaluation of how individual control structures interact and influence program behavior. Debugging efforts in this context focus on tracing execution flow, verifying logic, and refining conditional or iterative processes. Mastery of multiple control structures supports structured problem-solving and the accurate implementation of algorithms.

[7.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.

[7.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.

[7.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.

[7.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

Creating and testing programs with multiple control structures strengthens understanding of how different elements—such as loops and conditionals—work together to control program flow. Tracing code and analyzing results helps students identify logic and syntax errors, assess whether the program behaves as expected, and determine the validity of its outcomes. Revising algorithms based on this analysis supports iterative problem-solving and improves program accuracy and performance.

CONNECTIONS

Within the grade level/course: At this grade level, students can work collaboratively to maintain a project timeline and complete individual tasks on time. Using computational skills to create artifacts is integral across various subjects, enhancing students' ability to organize, analyze, and interpret information systematically.

Vertical Progression: In Grade 6, Students incorporate a collection of numerical data such as Boolean, integer, and decimal number variables in order to predict the results of logic expressions that use Boolean operators in their block or text-based programming (6.AP.2). In Grade 8, Students continue predicting results of data collection and plan algorithms using plain language, pseudocode, or diagrams to describe outcomes (8.AP.2).

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.
- **6-8.CC** Students communicate clearly and express themselves creatively for a variety of purposes using appropriate technologies (including assistive technologies), styles, formats, and digital media appropriate to their goals.
  D. Students select and use appropriate technologies to design, publish, and present content that effectively convey their ideas, conclusions, and evidence for specific audiences.

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.

Mathematics

- Students use the iterative design process to plan, create, test, and refine a program that models proportional relationships, such as cost per item or distance over time. Using a block-based or text-based programming language, students write code that calculates proportional values and displays them in a table or visual format, reinforcing key mathematics concepts. As they build their programs, students will apply the iterative design process by debugging errors, evaluating output accuracy, and improving functionality through multiple revisions.

Science

- Students use the iterative design process to develop a simulation or model that shows how changes in temperature, precipitation, carbon dioxide levels affect ecosystems over time. Using a block-based or text-based programming environment, students will model interactions between biotic and abiotic components.

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:

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

D. Fostering Computational Thinking Practices:

1. Responsible Use Practices

7.AP.4 The student will apply proper attribution when incorporating other sources into original work.

a. Apply proper methods of attribution when using work from the Internet and other sources.
b. Incorporate information or assets from the Internet into a program with 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.

[7.AP.4a] 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.

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.

[7.AP.4b] External algorithms, code segments, and media assets may be used in program development to support functionality and efficiency. These imported components can range in scale from small code snippets to full libraries that provide complex operations.

Editing and integrating pre-existing content introduces alternate approaches to solving problems and expands the range of program capabilities. Proper attribution must be provided for all externally sourced material. This includes citing original authors and referencing licensing terms in documentation or program interfaces to maintain academic integrity and comply with usage requirements.

Teachers should help students understand and use various sources appropriately in their work. This includes understanding licensing terms and selecting resources that are legally and ethically suitable for their projects. Open-source code: Help students understand what "open-source" means and introduce them to common licenses like MIT, GNU, and Creative Commons.

- Media (images, audio, video): Encourage students to use royalty-free or licensed media. Explain how to check the usage rights for any media they find online.
- Data sets: Direct students toward reliable, public datasets for use in their projects (e.g., government or educational sources).

For example, when creating a game, students may incorporate portions of code that create a realistic jump movement from another person's game, and they may also import Creative Commons-licensed images to use in the background. Creative Commons is a set of intellectual property laws that govern the use of media created by others. Proper attribution includes the author and source of the code or image.

Attribution may be provided as comments within the code, in a dedicated credits section of the user interface, or in accompanying documentation such as a README file. The placement of attribution should ensure it is clearly visible and accessible to others reviewing, using, or modifying the program. Proper attribution typically includes the original creator’s name, the title or description of the asset, a link to the source, and the applicable license (e.g., Creative Commons).

Concepts and Connections

Concepts

Attribution is the practice of giving proper credit to the original creators of content used from the Internet or other sources. When students incorporate external information or assets into a program, they must include appropriate citations to support ethical computing and respect for intellectual property rights.

Connections

Within the grade level/course: At this grade level, students incorporate information and images in their digital and print products across all content areas. Guidelines for citing sources may differ across content areas.

Vertical Progression: In Grade 6**,** students identify information needed to give proper attribution when using or remixing the work of others (6.AP.4). In Grade 8, students seek out ways to use and remix code from other projects while giving proper attribution to the originator (8.AP.4).

ACROSS CONTENT AREAS

English

- **7.C.2.v** Students need to be able to cite and acknowledge the sources that are referenced in a presentation. Students avoid plagiarizing information by utilizing referencing materials; this helps demonstrate integrity, credibility, and respect for intellectual property rights.

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.
- **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.
  C. Students break problems into component parts, identify key pieces and use that information to problem solve using technologies, when appropriate.
- **6-8.CC** Students communicate clearly and express themselves creatively for a variety of purposes using appropriate technologies (including assistive technologies), styles, formats, and digital media appropriate to their goals.
  D. Students select and use appropriate technologies to design, publish, and present content that effectively convey their ideas, conclusions, and evidence for specific audiences.

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 research a current topic and create a media message (e.g., infographic, video, or digital poster) that presents a clear claim supported by valid evidence from credible sources. They will evaluate the reliability of both images and textual information, verify accuracy across multiple sources, and apply proper citation practices.

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 Solve

C. Fostering Iterative Design Practices:

1. Identify, Define, and Evaluate Real-World Problems
2. Plan and Design Computational Artifacts

Computing Systems (CSY)

7.CSY.1 The student will design projects that use computing devices to collect and exchange data.

a. Apply project management skills to distribute tasks and maintain project timeline.
b. Generate ideas combining hardware and software components that can be used to collect and exchange data.
c. Describe how hardware and software can be used together to collect and exchange data.
d. Evaluate the usability of hardware and software to collect and exchange data.
e. Select the hardware and software components for project designs by considering factors such as functionality, cost, size, speed, accessibility, and aesthetics.

Understanding the Standard

[7.CSY.1a] Project management refers to the structured coordination of tasks, resources, and timelines to complete a defined objective. It involves organizing project components, assigning responsibilities, establishing deadlines, and monitoring progress to ensure successful completion.

Effective project management skills include:

- Task delegation: Distributing responsibilities among team members.
- Time management: Keeping the project on schedule and meeting deadlines.
- Communication: Ensuring that all team members stay informed and can collaborate effectively.
- Problem-solving: Addressing issues that arise during the project.
- Organization: Tracking progress, setting goals, and adjusting plans as needed. When designing projects with computing devices to collect and exchange data, students must consider how hardware and software work together. For instance, a temperature sensor (hardware) can gather data that is logged and displayed by a software program, demonstrating effective integration.

[7.CSY.1bc] Combining hardware and software enables integrated systems to collect, process, and exchange data. Hardware components, such as sensors, cameras, and microcontrollers, detect and gather physical or digital input. Software components interpret and manage this input through computational processes, enabling system responses and decision-making.

This integration supports automation, real-time data processing, and system interaction. Communication between components occurs through wired or wireless connections, allowing for seamless information flow across devices. Integrated systems are foundational to technologies such as robotics, smart environments, and health monitoring, where simultaneous data sensing, processing, and transmission are required for coordinated action.
Consider the following examples:

- Using a temperature sensor (hardware) to gather temperature data and a software program to log and display the data.
- A smartphone (hardware) with a fitness app (software) that collects and exchanges data on the user's steps and heart rate.

[7.CSY.1de] Evaluating the usability of hardware and software requires analysis of functionality, compatibility, speed, cost, size, reliability, scalability, accessibility, security, and user interaction. Technical factors such as power efficiency, environmental constraints, and responsiveness to input are also part of the assessment. When systems are involved in collecting, storing, or transmitting data, privacy and security features must be reviewed.

The evaluation process begins by defining project requirements, including the type of data to be collected, the method of data exchange, and any operational constraints. Usability is then measured against these requirements using consistent criteria across all potential tools and systems.

Once options are identified, prototype testing is conducted to observe performance in a controlled setting. This phase allows for the identification of performance limitations and system inefficiencies. Trade-offs are analyzed to determine which tools best meet the technical goals. Documenting observations, decisions, and justifications provide clarity and supports future development or iteration.

Concepts and Connections

CONCEPTS

Project management skills help students organize tasks, collaborate effectively, and meet project deadlines. Combining hardware and software to collect and exchange data encourages creativity and real-world problem solving. Understanding how these components work together, evaluating their usability, and selecting them based on key factors such as cost, speed, and accessibility prepares students to design functional and user-centered computing solutions.

CONNECTIONS

Within the grade level/course: At this grade level, students design projects utilizing computing devices to collect and exchange data, applying project management skills to distribute tasks and maintain project timelines. These competencies are integral across various subjects, enhancing students' ability to manage projects, analyze data, and understand complex systems.

Vertical Progression: In Grade 6, Students explain how components of computing systems function and the limitations of software and operating systems (6.CSY.1). In Grade 8, students make recommendations and design improvements to computing devices based on user interactions (8.CSY.1).

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.
  B. In collaboration with an educator, students select and use appropriate technologies to plan and manage a design process that identifies design constraints and trade-offs and weighs risks.
- **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.
  B. 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 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 careers in the public and private sectors that rely on data collection and exchange, such as cybersecurity, healthcare, or environmental science. Using hardware (e.g., sensors, input devices, or surveys) and software tools (e.g., spreadsheets, databases, or data visualization platforms), students will design and manage a project that organizes and categorizes job roles, required skills, tools used, and career pathways.

Mathematics

- Students design and build programs that collect numerical data such as responses from a class survey using input hardware. The program will organize the data in ascending or descending order and compare rational numbers to support reasoning through numerical relationships. As part of the design process, students will evaluate and select appropriate hardware (e.g., device type) and software (e.g., coding platform or spreadsheet tool) based on criteria such as functionality, cost, speed, accessibility, and usability. Students will also analyze how well their chosen tools support accurate data collection, organization, and exchange, reflecting on how the integration of hardware and software enhances the effectiveness and reliability of their final product.

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
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
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

7.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. Compile and record successful methods used to resolve problems for common hardware, software, and connectivity-related problems.

Understanding the Standard

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.

Troubleshooting strategies for hardware, software, and connectivity rely on systematic procedures that mirror conditional logic structures used in programming. These protocols often follow an if/then decision-making process, where each step depends on the outcome of the previous condition (e.g., If the device powers on, then check the display; if not, test the power source).

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.

Following a systematic process is key to effective problem-solving. Instead of randomly guessing solutions, students should follow a logical path. This includes identifying the symptoms, isolating variables, testing hypotheses, and verifying results. Documenting each step helps track what has been attempted and prevents redundant efforts. Using diagnostic tools, system logs, or error codes can provide additional insights into the root cause. This methodical approach not only improves efficiency but also reinforces computational thinking and troubleshooting as repeatable, transferable skills.

[7.CSY.2b] Documenting the troubleshooting process is essential for tracking attempted solutions, identifying patterns, and avoiding repeated errors. It creates a clear record that can be referenced later, shared with others for support, or used to improve future problem-solving efficiency. In professional settings, thorough documentation is also critical for maintaining system reliability and accountability. Students should document the problem, outline the steps taken to resolve it, and record the final solution. This process creates a valuable reference that supports reflection, reinforces effective troubleshooting strategies, and enables both the individual and others to apply proven solutions to similar issues in the future.

Concepts and Connections

Concepts

Systematic problem-solving processes enable students to diagnose and fix issues related to hardware, software, and connectivity efficiently. Documenting successful solutions creates a valuable resource for troubleshooting common technical problems and supports ongoing learning and collaboration.

Connections

Within the grade level/course: At this grade level, students design projects utilizing computing devices to collect and exchange data, applying project management skills to distribute tasks and maintain project timelines. These competencies are integral across various subjects, enhancing students' ability to manage projects, analyze data, and understand complex systems.

Vertical Progression: In Grade 6, students identify and explain problems with computing systems and identify resources to help troubleshoot these problems. (6.CSY.2) In Grade 8, 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)

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.

Science

- Students design experiments involving collections of observational data with digital tools such as thermometers, cameras, and microscopes. Students present findings and compare their uses of hardware and software to enable data collection and analysis. As part of the process, students document troubleshooting steps taken when issues arise such as connectivity problems, calibration errors, or software glitches and compile effective solutions as a reference for future experimentation and simulation work.

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:

1. Decompose Relevant Problems
2. Explore Common Features and Relationships to 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:

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

Cybersecurity (CYB)

7.CYB.1 The student will ill differentiate physical and digital security measures that protect electronic information.

a. Compare and contrast physical and digital security measures.
b. Research and synthesize the tradeoffs between usability and security.
c. Identify common threats and vulnerabilities associated with Internet use and Internet-based systems.
d. Identify potential solutions to address common threats and vulnerabilities.

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.

[7.CYB.1a] Physical and digital security measures are essential for maintaining the integrity and availability of technology systems.

Physical security includes securing hardware through locks, surveillance, restricted access, and controlled environments to prevent tampering, theft, or physical damage. Digital security involves protecting systems and data from cyber threats using tools such as authentication, encryption, firewalls, anti-malware software, and network security protocols. These tools are designed to prevent unauthorized access, data breaches, and malicious activity.

Together, these measures work to safeguard the hardware and the data it processes to ensure computing systems remain operational, data remains confidential, and resources are accessed only by authorized users.

- Physical security measures protect computing devices and hardware from unauthorized access, theft, or physical damage. These measures include tools such as locks, security cameras, biometric scanners, and secure storage facilities that restrict physical access to sensitive equipment. In both everyday computing such as school laptops or home Wi-Fi routers and in emerging technologies like AI enabled devices, smart sensors, and cloud-connected infrastructure, physical safeguards are essential for maintaining system integrity. As technology becomes more integrated into daily life, physical security remains a foundational layer in the broader effort to protect digital environments.

- Digital security measures are protections integrated into software, applications, and network systems to prevent unauthorized access, data breaches, and malicious activity. These measures include passwords, encryption, firewalls, multi-factor authentication, and antivirus software, all of which help safeguard data stored on devices or transmitted across online networks. In everyday computing, such as logging into personal email or using cloud-based apps, digital security ensures that user data remains private and protected. In emerging technologies, such as AI systems, smart home devices, and connected medical equipment digital safeguards are critical for safeguarding sensitive information.

[7.CYB.1b] Security and usability must be balanced to ensure systems remain functional while protecting data and resources. Excessive security measures may hinder accessibility or user experience, while insufficient protections increase the risk of unauthorized access and data breaches.

Effective system design evaluates user roles, data sensitivity, access points, and potential threats to determine appropriate safeguards. Authentication and authorization protocols are implemented to verify identity and regulate user permissions, while maintaining operational efficiency.

- Usability refers to how easily and efficiently a user can access and use a system. User-friendly systems are convenient but sometimes have fewer barriers to access, which can lower security.
- Security increases protection through barriers such as multi-factor authentication and complex passwords, which can sometimes make a system less accessible.

[7.CYB.1c] Digital security threats are tactics used to exploit system vulnerabilities and gain unauthorized access, disrupt operations, or compromise data integrity. Common threats include malware, phishing, ransomware, denial-of-service attacks, and unauthorized intrusions. These threats target flaws in software, hardware, network configurations, or user behavior.

- Malware is malicious software designed to harm or exploit devices, systems, or networks. Some examples include:
- Viruses replicate and spread, damaging files or systems.
- Worms self-replicate and spread across networks.
- Ransomware locks users out of their data and demands payment to restore access.
- Spyware secretly monitors user activity and collects sensitive data, such as login credentials or personal information.
- Phishing attacks deceive users into providing personal information by posing as a legitimate source. These attacks often come in the form of fake emails or websites that mimic trusted entities, like banks or social media platforms.
- Smishing, a subset of phishing delivered via SMS texts, demonstrating that these threats are not limited to emails or websites.

A data breach is an incident in which unauthorized individuals gain access to sensitive, protected, or confidential information stored on a system or network. Breaches often occur through exploited vulnerabilities such as weak authentication, outdated software, misconfigured systems, or social engineering tactics.

Exposed information may include personally identifiable information (PII), financial records, health data, or proprietary business content. The consequences of a data breach can include identity theft, financial fraud, operational disruption, and loss of trust in the affected organization. Understanding how breaches occur supports the implementation of protective measures to prevent unauthorized access and ensure data integrity.

[7.CYB.1d] Identifying solutions to address threats and vulnerabilities involves recognizing known attack methods and applying targeted safeguards to reduce risk. Security measures must be maintained and updated regularly to counter evolving threats and maintain data protection.

Users can mitigate risks by inspecting emails and messages for indicators of phishing or social engineering, including spelling errors, misleading links, and unsolicited requests for confidential information. Verifying the sender's identity and confirming the legitimacy of the request reduces the chance of compromise. Suspected threats should be reported immediately to prevent unauthorized access and support system integrity.

Concepts and Connections

CONCEPTS

Physical and digital security measures differ in approach, but both aim to protect assets from unauthorized access. Understanding the balance between usability and security helps evaluate how protective measures impact user experience. Recognizing common internet threats and vulnerabilities, along with potential solutions, equips students to safeguard systems and data effectively.

CONNECTIONS

Within the grade level/course: At this grade level, students differentiate between physical and digital security measures that protect electronic information. Students compare these measures, research the trade-offs between usability and security, identify common threats and vulnerabilities associated with Internet use, and propose potential solutions. These skills are essential for understanding how to safeguard data and systems in an increasingly digital world.

Vertical Progression: In Grade 6, students evaluate the risks and benefits of sharing information (6.CYB.1). In Grade 8, students identify ways to protect shared information from malware and other cyber-attacks and model common prevention practices (8.CYB.1).

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 footprint.

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 real-world examples of cybersecurity incidents and their impact on communities, such as data breaches. They can then create a public service announcement (PSA) or infographic that educates their peers on best practices for digital security, highlighting how civic participation, such as staying informed and advocating for better security policies, contributes to protecting online information.

Mathematics

- Students analyze real-world data on cybersecurity incidents and calculate percentages or trends (e.g., What % of breaches are due to weak passwords?).

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
2. Plan and Design Artifacts

D. Fostering Digital Literacy Practices:

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

Data and Analysis (DA)

7.DA.1 The student will utilize computational tools to visualize and evaluate data to draw conclusions and make predictions.

a. Develop computational models that simulate real-world phenomena, considering relevant variables and relationships.
b. Refine and modify computational models based on observed data and feedback, ensuring alignment with empirical evidence.
c. Analyze patterns and trends within observed data, comparing them with the predictions made by computational models.
d. Evaluate the effectiveness and accuracy of computational models in capturing and explaining the observed data.

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.

[7.DA.1a] 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.

Variables within a computational model represent measurable attributes of a system. Logic and conditional rules define how those variables influence one another. The model's output reflects how the system behaves over time or in response to different scenarios. This structure supports exploration, prediction, and analysis of complex processes.

- A computational model simulates real-world events, factoring in relevant variables and relationships.

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.

[7.DA.1b] Refining and modifying computational models involves analyzing output accuracy and making adjustments based on observed data. The process includes comparing model behavior with real-world conditions to identify inconsistencies.
Adjustments may involve modifying variable ranges, altering conditional logic, or updating interaction rules. Testing and evaluation help determine which elements require revision. Improvements may include tuning existing parameters, adding new constraints, or revising algorithms. The refinement process increases a model's predictive accuracy, reliability, and applicability.

- Consider the following example:If a climate model doesn't accurately predict temperatures, students might add variables like humidity or altitude to refine the results.

[7.DA.1c] Analyzing patterns involves comparing trends in data to the predicted outcomes generated by a computational model. This comparison reveals the extent to which the model accurately represents real-world behavior.

Data patterns are examined to determine whether model outputs align with observed or expected results. Discrepancies between the two indicate areas where the model may require refinement. Pattern analysis supports the evaluation of model accuracy and informs adjustments to improve predictive performance.

- Consider the following example:In a wildlife model predicting deer population growth, students could observe actual seasonal population changes and adjust their model if it only predicts steady growth, aligning the model with real data.

After comparing trends and validating computational models for accuracy and reliability, the next step is to apply those models to make predictions and draw meaningful conclusions. Once a model is shown to represent real-world behavior effectively, students can analyze its outputs to forecast future outcomes or test "what-if" scenarios. This application deepens student understanding of variable relationships and helps them interpret results in a broader context, reinforcing the model's role as both an analytical and a decision-making tool.

[7.DA.1d] Evaluating the effectiveness of computational models involves comparing predictions to real-world data and identifying discrepancies. This process supports determining how accurately a model represents the observed phenomena.

Error analysis is used to identify areas where the model diverges from expected outcomes. Based on this analysis, adjustments are made to improve alignment between the model and real-world behavior. This evaluation process strengthens model accuracy and informs future iterations.

- Consider the following example:If a model forecasting energy usage consistently underestimates peak usage, students might identify missing variables or incorrect assumptions and refine the model accordingly.

Concepts and Connections

Concepts

Computational models simulate real-world phenomena by representing key variables and their relationships to predict outcomes. These models are refined through feedback and observed data to better align with actual evidence. Analyzing data patterns and comparing them to model predictions enables evaluation of how accurately the models capture and explain real-world behavior.

Connections

Within this grade level: At this grade level, students utilize computational tools to visualize and evaluate data, enabling them to draw conclusions and make predictions. Students develop computational models that simulate real-world phenomena, considering relevant variables and relationships. These models are refined based on observed data and feedback to ensure alignment with empirical evidence. By analyzing patterns and trends within the data, students compare their observations with the predictions made by their models, evaluating the effectiveness and accuracy of these simulations. This process enhances their analytical skills and deepens their understanding of complex systems across various disciplines.
Vertical progression: In Grade 6, students begin using computational tools to visualize data, distinguish between which visual representations should be used, and synthesize and evaluate data from visual representations. (6.DA.2). In Grade 8, students use computational tools to visualize data and compare various visual representations to determine which should be used. (8.DA.2)

Across Content Areas

Mathematics

- **7.PS.2** The student will apply the data cycle (formulate questions; collect or acquire data; organize and represent data; and analyze data and communicate results) with a focus on histograms.

Science

- **7.1** The student will demonstrate an understanding of scientific and engineering practice. *7.1 standard is integrated within science content and not taught in isolation. Potential science concepts to apply 7.1 include:**LS.2** The student will investigate and understand that all living things are composed of one or more cells that support life processes, as described by the cell theory (cell). **LS.4** The student will investigate and understand that there are chemical processes of energy transfer which are important for life. (photosynthesis) **LS.5.a** The student will analyze the cycling of matter through ecosystems via the carbon, water, and nitrogen cycles. and **LS.6.a** The student will analyze and interpret data to predict and explain the effects of resource availability on organisms and populations in an ecosystem.*

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 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 use computational tools to analyze the structure of informational texts by identifying key ideas, transitions, and the relationships between paragraphs. By creating visual representations such as flowcharts, timelines, or cause-and-effect diagrams, students will organize textual evidence and examine how each section contributes to the development of the author's central idea or argument.

History and Social Science

- Students use data visualization tools to analyze economic trends in Virginia and the global economy. Students create graphs or computational models that illustrate the impact of technological innovations on industries, employment rates, or trade, then draw conclusions about economic patterns and predictions.

Mathematics

- Students collect or use existing datasets and apply the data cycle to create histograms. Using spreadsheet software or coding tools, they can organize, visualize, and analyze data, then compare trends to draw conclusions about distributions and variability.

Science

- Students design and refine a computational simulation that models the cycling of matter such as carbon, nitrogen, or water. By adjusting variables, students will analyze cause-and-effect relationships and interpret the impact of these changes. Students will compare their simulation outputs to real-world datasets, evaluate the validity of their model, and revise it to improve accuracy.
- Students use computational tools to model the frequency and impact of natural hazards, such as earthquakes, hurricanes, or floods, using real-world data. They will analyze patterns over time, identify trends, and use basic algorithms to simulate or predict possible future events. Students will present their findings, explaining how scientists and emergency planners use data, models, and computing to make informed decisions about risk and preparedness.

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:

1. Decompose Relevant Problems
2. Explore Common Features and Relationships to 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

7.DA.2 The student will explain the process and application of computational thinking in machine learning.

a. Explain how supervised, unsupervised, and/or reinforcement learning methods utilize decomposition, pattern recognition, abstraction, and algorithms to learn from and make decisions.
b. Explore neural networks and its role in machine learning and artificial intelligence.

Understanding the Standard

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.

Artificial intelligence (AI) is focused on designing intelligent agents, software systems that perceive their environment, make decisions, and act to achieve goals. An intelligent agent is software that has been designed to use artificial intelligence and machine learning to perform tasks without human intervention. AI systems are designed to mimic human intelligence, enabling them to perform tasks that would typically require human cognitive abilities.

Machine learning allows systems to automatically improve their performance by learning from data. Rather than following fixed rules, these systems identify patterns within the data, make predictions, and take actions based on what they've learned. For instance, a machine learning model can learn to recognize images of cats by analyzing numerous labeled images and identifying the unique patterns that define a cat.

- 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.Many

[7.DA.2a] Machine learning relies on several methods, each utilizing computational thinking in unique ways. These include supervised learning, unsupervised learning, and reinforcement learning. Each method uses decomposition, pattern recognition, abstraction, and algorithms to process data and make decisions.

- Supervised learning is a method where a computer learns to make decisions or predictions by analyzing examples with known, correct answers. In this approach, the system is trained using labeled data, meaning each input is paired with an expected output, such as images tagged with their corresponding objects or data points labeled with categories. This allows the system to recognize patterns and make accurate predictions when given new, unseen data.

Application of Computational Thinking: Supervised Learning

- Decomposition: Breaking down data into features that are easier to analyze.
- Pattern Recognition: Recognizing relationships between inputs and known outputs to make predictions on new data. Consider the following example:Training an algorithm to classify email as spam or not based on labeled examples.

- Unsupervised learning is when a computer learns from data without being told what the answers are. Allowing the system to find groups or patterns without given answers. In this approach, the model learns from unlabeled data, aiming to find hidden patterns without any predefined labels.

Computational Thinking Application: Unsupervised Learning

- Pattern Recognition: The model groups data based on similarities it finds, like sorting objects into categories without being told the names of those categories.
- Abstraction: Focusing on larger trends or similarities within data, rather than specifics. Consider the following example:Giving a program a collection of photos with no labels. The program might group the photos based on patterns, like grouping all the nature photos together and all the animal photos together. The program has learned to identify different types of images without knowing exactly what they are.

- Reinforcement learning involves teaching a computer through trial and error, where it learns by getting rewards or punishments for the actions it takes. The system takes actions and learns by receiving feedback (rewards or penalties).

Computational Thinking Application: Reinforcement Learning

- Algorithms: The model follows step-by-step instructions, adjusting its actions based on rewards (positive feedback) or penalties (negative feedback).
- Pattern Recognition: The model identifies patterns of successful actions that maximize rewards, helping it repeat desirable behaviors.Consider the following example: A robot learning to navigate a maze.

- When the robot moves closer to the exit, it receives a "reward" (e.g., positive points), encouraging similar actions.
- If the robot hits a wall or moves away from the exit, it gets a "penalty" (e.g., losing points), teaching it to avoid these actions.
- Through trial and error, the robot maximizes rewards and minimizes penalties, learning the most efficient path to the maze's exit.

These three main approaches to machine learning have been introduced at an elementary level through age-appropriate activities that emphasize foundational concepts. Students explored supervised learning by recognizing labeled examples, unsupervised learning by identifying patterns and groupings in data, and reinforcement learning by adjusting actions based on feedback or rewards. Understanding how the different approaches to machine learning function through the application of computational thinking provides students with a deeper understanding about how machines learn and make decisions.

Training data provides the foundation for machine learning by supplying the examples an algorithm uses to detect patterns, learn relationships, and generate accurate predictions or classifications.

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. The data provides structured input that guides the model in identifying relationships and improving accuracy over time. Sources include:

- Public datasets from government agencies, research institutions, and open repositories
- User-generated content such as social media posts, reviews, or multimedia uploads
- Sensor data from smartphones, wearables, or Internet of Things (IoT) devices
- Synthetic data generated by algorithms when real-world data is insufficient or sensitive

Training data must be accurate, diverse, and properly labeled to avoid bias and improve model performance. Data often undergoes preprocessing to ensure consistency. To ensure its effectiveness, training data must be carefully curated through processes such as cleaning (correcting errors and inconsistencies), labeling (assigning categories), and annotating (adding context or metadata). These steps improve data quality, reduce bias, and make the information more meaningful for machine learning algorithms. The quality of the training data directly influences how accurately and fairly an AI system performs.

Training data is distinct from testing and validation data, which are used to evaluate the model after learning is complete.

[7.DA.2d] Neural networks are a fundamental component of machine learning and artificial intelligence, mimicking the way human brains process information. They consist of layers of interconnected nodes (neurons) that analyze data through multiple steps to make decisions. Neural networks are computational models inspired by the structure of the human brain. They contain layers of nodes that work together to analyze input data, extract features, and make decisions.

- Consider the following example:In an image recognition model, a neural network might first detect basic shapes in an image, then combine them into more complex structures to identify the overall object.

Role of Neural Networks in Machine Learning and AI:

- Neural networks process large datasets and recognize complex patterns, enabling systems to make decisions with high accuracy.
- They are used in AI systems that learn and improve over time, such as virtual assistants or recommendation engines. Consider the following example:Virtual assistants use neural networks to process voice commands, recognize speech patterns, and respond accurately to user questions.

** This standard focuses on developing an understanding of what neural networks are and their role within artificial intelligence, specifically how neural networks are modeled after the human brain and are used to recognize patterns, make decisions, and support machine learning processes. The detailed study of different neural network types is beyond Grade 7.*

Concepts and Connections

CONCEPTS:

Supervised, unsupervised, and reinforcement learning methods use decomposition, pattern recognition, abstraction, and algorithms to analyze data, identify meaningful patterns, and make informed decisions. Neural networks, a key component of machine learning and artificial intelligence, mimic the structure of the human brain to process complex data and improve learning outcomes through layered computations.

CONNECTIONS

Within this grade level: At this grade level, students explore the application of computational thinking in machine learning. They learn how supervised, unsupervised, and reinforcement learning methods utilize decomposition, pattern recognition, abstraction, and algorithms to learn from data and make decisions. Students are introduced to neural networks, understanding their role in machine learning and artificial

intelligence. These concepts enhance their analytical skills and provide insights into the mechanisms driving modern technological advancements.
Vertical progression: In Grade 6,students investigate ways that humans provide training data, explore the role of human intervention in curating training data, and explain the accuracy of artificial intelligence systems based upon the training data used (6.DA.4). In Grade 8, students evaluate computational models to analyze patterns and make recommendations or predictions (8.DA.2).

Across Content Areas

Mathematics

- **7.PS.1** The student will investigate and describe the difference between the experimental and theoretical probability.

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 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 analyze how algorithms categorize and personalize content through machine learning, particularly in recommendation systems used by media platforms. They will evaluate how AI-driven pattern recognition influences the type and order of information presented to them. Using evidence from digital media and classroom discussion, students will compose an analytical reflection that explains how algorithms shape online experiences and assess the broader implications

Mathematics

- Students simulate a simple supervised machine learning task (e.g., classifying fruit by color/size), predict outcomes using training data (theoretical), then test with new data (experimental). Students then calculate the model's accuracy and discuss how closely it aligns with their predicted outcomes.

- Students use a programming tool to develop a simulation that models a repeated probability experiment, such as rolling dice or flipping coins. The program will initially generate predictions based on uniform random distributions, reflecting theoretical probability. As simulated trials accumulate, students will integrate statistical analysis to dynamically update predictions based on observed outcomes, representing experimental probability.

Science

- Students are given an unorganized or complex data set (e.g., climate, disease outbreaks, chemical reactions) and must clean, sort, and train a simplified model to find meaningful patterns. Students evaluate how model accuracy depends on data quality.
- Students investigate inheritance patterns by analyzing trait data sets and applying Punnett squares to model genetic outcomes. As part of the process, students simulate a supervised machine learning process by using labeled genotype data to predict offspring traits.

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
4. Apply Algorithmic Thinking to Problem Solve and Create
5. Apply Computational Thinking Practices to Select, Organize, and Interpret Data Sets

A. Fostering Iterative Design Practices:

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

A. Fostering Collaborative Computing Practices:

1. Responsible Use Practices

Impacts of Computing (IC)

7.IC.1 The student will assess the national and global impact of computing technologies.

a. Discuss specific examples of how computing technologies have influenced various national and global industries and sectors.
b. Analyze the implications of emerging technologies and potential real-world impact nationally and globally.
c. Evaluate the environmental impact of computing technologies nationally and globally.

Understanding the Standard

Computing technologies include hardware, software, and network systems used for various applications, such as smartphones, computers, cloud storage, artificial intelligence, and the Internet. These technologies facilitate global information sharing through devices like smartphones and computers, software such as social media platforms, and systems like cloud computing. Increasingly, these components are interconnected, forming integrated ecosystems that support real-time communication, data processing, and intelligent automation. Understanding how these technologies work together is essential, as they play a central role in everyday life, education, business, and the development of emerging fields such as robotics, cybersecurity, and machine learning.

[7.IC.1a] Computing technologies have both national and global impacts, affecting economies, industries, and personal lives around the world. Their influence extends across sectors such as healthcare, education, transportation, and communication. Each sector integrates emerging technologies to improve efficiency, access, and functionality. These impacts contribute to shifts in workforce demands, global collaboration, and the development of new systems and services.Consider the following example of the impact of computing technologies:

- Healthcare: Medical technologies like telemedicine and electronic health records improve patient care and accessibility globally.
- Education: Online learning platforms expand access to education, especially in remote areas.
- Communication: Social media and messaging apps allow instant global communication, changing how people connect and interact.

Industries and sectors use computing technologies to develop new products, streamline operations, and increase accessibility. Applications of computing span manufacturing automation, data analysis in finance, electronic health records in medicine, and cloud-based services in education and retail. These technologies support system optimization, digital transformation, and scalable growth across domains. Consider the following examples:

- Retail: E-commerce platforms Alibaba allow businesses to reach a global market.
- Transportation: GPS and ride-sharing apps transform how people navigate and access transportation.
- Finance: Digital banking and cryptocurrencies offer new financial tools and investment opportunities worldwide.

[7.IC.1b] Artificial intelligence and blockchain are examples of computing innovations that change how systems function across sectors. These technologies are used to automate decision-making, secure data transactions, and streamline digital interactions.
Emerging technologies raise considerations related to data privacy, algorithmic reliability, and compliance with regulatory frameworks. Their adoption requires evaluation of both functional capabilities and ethical implications.Consider the following examples:

- Artificial Intelligence: While AI can improve efficiency and decision-making, it also raises questions about job automation and data privacy.
- Blockchain: This technology provides secure transactions but requires significant energy, impacting environmental sustainability.

[7.IC.1c] The environmental impact of computing technologies includes energy consumption, electronic waste, and resource-intensive manufacturing processes. Increased demand contributes to higher power usage and the depletion of natural resources. Consider the following examples of environmental impact:

- Energy Consumption: Data centers require substantial energy to power and cool systems, impacting global energy resources.
- E-Waste: Disposing of outdated or damaged devices generates electronic waste, which can harm ecosystems if not managed properly. Consider the following example:The production of a single smartphone requires energy and materials, and improper disposal contributes to pollution.

Concepts and Connections

CONCEPTS

Computing technologies have transformed industries worldwide by improving efficiency, communication, and innovation across sectors such as healthcare, finance, and manufacturing. Emerging technologies bring new opportunities and challenges, influencing economies and societies on a national and global scale. Evaluating their environmental impact helps understand the sustainability and long-term consequences of technological advancement.

CONNECTIONS

Within this grade level: At this grade level, students assess the national and global impact of computing technologies. They discuss specific examples of how these technologies have influenced various industries and sectors, analyze the implications of emerging technologies, and evaluate the environmental impact of computing technologies. This comprehensive understanding enables students to critically evaluate the multifaceted effects of computing on society and the environment.
Vertical progression: In Grade 6, students examine the local impact of computing technologies. They research and analyze the implications of computing technologies. (6.IC.1) In Grade 8, students assess ethical implications of computing technologies and their social impacts. They research and critique the impact on the global economy and cultures. (8.IC.1).

Across Content Areas

History and Social Science

- **CE.13** The student will analyze the role of government in the United States economy, focusing on regulation and technological innovation.

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. Students manage their digital identities and reputations, including demonstrating an understanding of their digital footprints.
- **6-8.GC** Students use appropriate technologies, including assistive technologies, to broaden their perspectives and enrich their learning by collaborating with others and working effectively in teams locally and globally. Students use appropriate technologies to connect with others to develop a richer understanding of different perspectives, backgrounds, and cultures.

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 arguing for or against a specific technological advancement. They will support their position with relevant evidence, considering its societal, economic, and environmental impact while addressing opposing viewpoints.

History and Social Science

- Students research and analyze the impact of government regulations on technological advancements in various industries. They can create a timeline or infographic showing key policies that have shaped innovation, such as data privacy laws, environmental regulations on electronic waste, or policies influencing artificial intelligence development.

Mathematics

- Students use proportional reasoning to analyze data on the growth of computing technologies, such as the increase in internet usage, digital device production, or energy consumption in data centers. They can compare trends over time and create visual representations, such as graphs or tables, to illustrate proportional relationships.

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
3. Evaluate Resources and Recognize Contributions

7.IC.2 The student will describe and explain the impact of screen time on interactions with others.

a. Describe the positive and negative impact of social media on socialization.
b. Research the type of data collected on social media and online platforms that monitor social interactions.
c. Describe and explain the evolution of screen time and the impact it has had on social interactions.
d. Create a social media usage plan that demonstrates safe practices, meaningful use, and a balanced approach.

Understanding the Standard

Screen time refers to the amount of time spent interacting with devices that have digital screens, such as smartphones, tablets, computers, and TVs. This includes both passive activities (e.g., watching videos) and interactive ones (e.g., using social media or gaming). According to the National Institutes of Health (NIH), high screen time can begin early and may influence social development from a young age. Research shows that while screen time provides new ways to connect, excessive or unbalanced use can reduce quality time spent with family and friends, impacting social development (American College of Sports Medicine, 2017).

[7.IC.2a] Social media platforms enable users to exchange messages, share content, and engage in virtual communities. These tools support the maintenance of existing relationships and the formation of new connections based on common interests or affiliations. Digital devices and platforms facilitate continuous access to communication across geographic boundaries.

Extended use of social media can affect attention, emotional well-being, and real-world engagement. Identifying both the functions and limitations of these platforms helps users understand their role in shaping communication and social behavior.

Positive Impacts of Social Media:

- Connection Across Distances: Social media allows people to maintain relationships across distances, promoting a sense of community and belonging.
- Support Communities: Digital platforms provide supportive spaces for those facing similar challenges or interests, fostering positive social interactions.

Negative Impacts of Social Media:

- Reduced Face-to-Face Interaction: Excessive screen time can limit opportunities for direct communication, which is crucial for developing social skills and empathy (National Institutes of Health, 2023).
- Mental Health Concerns: Studies, including those summarized by the American College of Sports Medicine, show that high social media use correlates with increased risks of anxiety and depression, particularly in teens (American College of Sports Medicine, 2017).

[7.IC.2b] Social media platforms collect personal, behavioral, and technical data from users. Personal information includes names, email addresses, and demographic details. Behavioral data includes user interactions, content preferences, search history, and social connections.

- Personal Information: User details like name, age, and location.
- User-generated Content: post, photographs, and messages.
- Behavioral Data: Interaction patterns like likes, shares, comments, and time spent on content, used to tailor user experiences.

Technical data includes device type, operating system, browser, IP address, and location. Collected data is used to support personalized content delivery, targeted advertising, and user engagement strategies. Platforms apply algorithms to analyze this data and determine what content and advertisements to display to individual users.

Data collected on social media platforms influences content recommendations, targeted advertising, and user interaction patterns. This data includes user preferences, browsing behavior, post engagement, and connections with other accounts. Understanding the scope and use of this data allows users to evaluate the impact of their online activity.

Recognizing what data is collected helps users make informed decisions about sharing personal information. Awareness of data tracking supports actions such as adjusting privacy settings, limiting disclosures, and managing platform permissions. These practices contribute to the responsible use of digital tools and increased control over personal data.

[7.IC.2c] Screen time has shifted from passive consumption, such as watching television, to interactive engagement through smartphones, apps, and social media. This transition has redefined communication, social interaction, and how individuals manage their time. Interactive screen use introduces continuous digital connectivity, which shapes habits, behaviors, and personal routines.

Evolution of Screen Time:

- Past: Early screen time was primarily passive, with limited impact on direct social interaction.
- Present: With smartphones and social media, screen time is highly interactive, often substituting direct communication with digital messaging (National Institutes of Health, 2023).

[7.IC.2d] Creating a social media usage plan establishes guidelines for managing screen time. The plan defines when and how platforms are used, promotes healthy digital habits, and encourages a balance between online activity and offline interaction. Establishing boundaries supports intentional use of technology and reduces the potential for overuse.

Components of a Social Media Usage Plan:

- Safe Practices: Set privacy settings, avoid oversharing, and be cautious with unknown contacts.
- Meaningful Use: Focus on positive, educational, or creative engagement, and maintain in-person connections.
- Balanced Approach: Establish screen time limits, designate screen-free periods, and prioritize face-to-face interaction.

Concepts and Connections

Concepts

Social media affects socialization by enhancing connectivity and communication while also posing challenges such as reduced face-to-face interactions and privacy concerns. Understanding the types of data collected on these platforms reveals how social interactions are monitored and analyzed. The evolution of screen time has significantly influenced social behaviors, highlighting the need for balanced and intentional usage. Creating a social media usage plan promotes safe, meaningful engagement while maintaining healthy boundaries.

Connections

Within this grade level: At this grade level, students examine the impact of screen time on academic performance by comparing and contrasting how screen time benefits and hinders academic performance. Students will also examine the impacts of social media and cyberbullying on their mental health. Students recognize that excessive screen time, especially with prolonged technology use and social media, can negatively affect both physical and mental health. To mitigate these effects, adopting healthy habits such as regular physical activity, limited screen time, and mindfulness techniques is important, along with being mindful of online content and reporting cyberbullying incidents.
Vertical progression: In Grade 6, students take an in-depth look at the impact of screen time on their physical and mental health. Students also examine social media use and cyberbullying. They will be able to define social media and cyberbullying along with their impact on their mental health.(6.IC.2) In Grade 8, students analyze the impact of screen time and social media on health and justify the argument that excessive screen time can have significant consequences on physical, emotional and cognitive development. (8.IC.2)

Across Content Areas

English

- **6.C.3** Integrating Multimodal Literacies Use media and visual literacy skills to select, organize, and create multimodal content that articulates the purpose of the presentation, using two or more communication modes to make meaning (e.g., still or moving images, gestures, spoken language, and written language). Craft and publish audience-specific media messages that present claims and findings in a logical sequence.

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 research the effects of screen time on adolescent academic performance, synthesizing information from credible sources, including scholarly articles, surveys, and digital media. Using this evidence, students create a persuasive media presentation that clearly articulates a claim, supports it with relevant data and reasoning, and addresses opposing viewpoints.

Mathematics

- Students collect authentic data on their personal screen time usage and daily activities over a set period. Using the data cycle students will apply statistical measures such as mean, median, mode, and range to examine trends in both individual and class-level datasets.

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

7.IC.3 The student will identify individual preferences, skillset, and experiences and determine how these relate to a chosen computer science career field.

a. Use a career interest assessment to identify and categorize preferences, skillsets, and experiences.
b. Evaluate and connect personal skillsets, interests, talents, and values to a computer science career.

Understanding the Standard

[7.IC.3a] A career interest assessment provides valuable insights into individual preferences, skills, and values to help students explore computer science careers that align with their unique strengths. By identifying interests and categorizing skills, students can better understand which computer science careers may suit them best. These assessments support informed decision-making by connecting students' self-awareness to career pathways within the computer science field. Incorporating career assessments into the learning process helps students set academic goals, select relevant coursework and extracurricular opportunities.

[7.IC.3b] Connecting personal skills to computer science careers helps students see how their strengths such as problem-solving, creativity, attention to detail, or logical reasoning can translate into various careers. By linking these skills to specific career paths like programming, data analysis, or user experience design, students can make more intentional choices about their education and future goals. This connection reinforces the relevance of computer science in diverse career fields and encourages deeper engagement with learning.

Evaluate Skillsets: Students can compare their assessment results with key skills required in computer science, such as problem-solving, critical thinking, creative design, and technical skills related to computer science fields: Consider the following example:

- A student who prefers creativity and design may see alignment with fields like web development or UI/UX design.
- A student who enjoys logical thinking and mathematics may find data science or cybersecurity appealing.

Concepts and Connections

CONCEPTS

Career interest assessments help individuals identify and organize their preferences, skills and experiences to better understand potential career paths. Evaluating personal strengths, interests, talents, and values provides students with the opportunity to make connections to computer science careers aligned with their goals and aspirations.

CONNECTIONS

Within the grade level/course: At this grade level, students investigate potential career paths for high demand careers that utilize computer science concepts and skills. Students discover that computer science skills and concepts extend beyond technology-focused careers and are applicable in fields like business, healthcare, and research. These careers involve data analytics, automation, and problem-solving, and it's

important for individuals to explore various career aspects, such as work expectations, pay rates, and required education, when considering future career paths.

Vertical Progression: In Grade 6, students narrow their focus to exploring a computing technology career of their choice and they examine how computational thinking practices are used in the chosen career. (6.IC.3) In Grade 8, students develop short- and long-term goals for a chosen career and research emerging trends in a chosen path by identifying training plans for computer science fields (8.IC.3).

ACROSS CONTENT AREAS

History and Social Science

- **CE.1** The student will demonstrate skills for historical thinking, geographical analysis, economic decision making, and responsible citizenship by a) analyzing and interpreting evidence from primary and secondary sources, including charts, graphs, and political cartoons; b) analyzing how political and economic trends influence public policy, using demographic information and other data sources; c) analyzing information to create diagrams, tables, charts, graphs, and spreadsheets; d) determining the accuracy and validity of information by separating fact and opinion and recognizing bias; e) constructing informed, evidence-based arguments from multiple sources; f) determining multiple cause-and-effect relationships that impact political and economic events; g) taking informed action to address school, community, local, state, national, and global issues; h) using a decision-making model to analyze and explain the costs and benefits of a specific choice; i) applying civic virtue and democratic principles to make collaborative decisions; and j) defending conclusions orally and in writing to a wide range of audiences, using evidence from sources.

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 conduct a research project on careers in data collection and analysis, applying strategies to locate, evaluate, and synthesize credible sources. Students organize their findings to develop a clear and well-supported explanation of how these careers contribute to decision-making in fields such as business, healthcare, or public policy.

History and Social Science

- Students investigate careers related to the development and impact of the Internet by exploring how technological innovation has influenced global economies, labor markets, and communication. Students analyze the skills and interests aligned with these careers

and examine how the growth of Internet-based careers reflects broader economic trends and globalization in the 20th and 21st centuries.

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
3. Create and Accept Feedback

B. Fostering Computational Thinking Practices:

1. Decompose Relevant Problems
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
3. Evaluate Resources and Recognize Contributions

7.IC.4 The student will identify and apply strategies to prevent personal and public works from being pirated and plagiarized.

a. Discuss and describe intellectual property protections.
b. Research and list safeguards used to prevent intellectual property infringement.

Understanding the Standard

[7.IC.4a] Protecting intellectual and creative works involves adhering to legal and ethical standards that govern ownership and distribution. Piracy refers to the unauthorized copying, distribution, or use of protected content such as software, music, videos, or written material. Plagiarism involves presenting someone else's work or ideas as one's own without proper attribution. These violations compromise the integrity of digital environments and infringe on intellectual property rights. Digital content must be used lawfully, with clear citation and authorization to preserve the value of original contributions and maintain professional standards.

- Consider the following example: Downloading a movie or software from an unauthorized site is considered piracy, as it bypasses the legal purchase or licensing process.

Plagiarism is the act of using another person's words, ideas, or creations without providing proper attribution. It involves presenting someone else's intellectual work as original content. This includes copying text, code, images, or other media without citing the source. Plagiarism violates academic and professional standards and undermines the integrity of digital and written work.

- Consider the following example:Copying text from a website into an essay without citing the source is plagiarism, as it fails to acknowledge the original author.

Intellectual property protections are legal rights that apply to original creations. These rights allow creators to control the use, distribution, and monetization of their work. Protected forms of intellectual property include inventions, written and artistic works, software, designs, and brand identifiers. Categories of intellectual property include copyrights, patents, trademarks, and trade secrets. These protections establish ownership and regulate how creative and technical content may be used in digital and physical environments..Consider the following example of intellectual property.

- Copyright: Protects creative works like books, movies, and songs, allowing the creator to control reproduction and distribution.
- Trademark: Protects logos, brand names, and symbols associated with goods or services, like the Nike "swoosh."

[7.IC.4b] Safeguards against intellectual property infringement include technical controls and legal protections. Digital rights management, watermarking, and access restrictions help prevent unauthorized copying, sharing, or alteration of content. Legal mechanisms such as

copyright registration, licensing agreements, and terms of use define ownership rights and usage permissions. Instruction on proper attribution, source citation, and ethical content use reinforces respect for intellectual property.

- Digital Watermarks: Creators can embed invisible watermarks in digital files (e.g., images or PDFs) to trace unauthorized use.
- Licensing Agreements: Establishes terms for legal use, often involving fees or usage limits, to prevent unauthorized distribution.
- Citing Sources: Encourages academic honesty by crediting authors and sources, showing respect for others' ideas and efforts.

Concepts and Connections

Concepts

Intellectual property protections safeguard creators' rights by legally securing their original works, inventions, and ideas. Various safeguards, such as copyrights, patents, trademarks, and digital rights management, help prevent unauthorized use and infringement of intellectual property.

Connections

Within the grade level/course: At this grade level, students examine types of software licenses and how each influences the use, distribution, and modification of software. Students evaluate the benefits and limitations of each, while also understanding the role of intellectual property rights in protecting developers and users. Through research and analysis, students learn to synthesize information and make informed decisions about software usage and licensing.
Vertical Progression: In Grade 6,Students identify the role of software licenses and why they are used. They compare the positives and negatives of different licenses. (6.IC.4) In Grade 8, students will incorporate work from others into programs and projects. (8.AP.4)

Across Content Areas

English

- **7.R.1A** Formulate questions about a research topic, broadening or narrowing the inquiry as necessary.
- **7.R.1D** Quote, summarize, and paraphrase research findings from primary and secondary sources, avoiding plagiarism by using own words and following ethical and legal guidelines.

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.

History and Social Science

- Students investigate the role of copyright and software licensing in protecting digital content, situating these concepts within the historical development of intellectual property rights and their influence on the global economy. Through case studies and international examples, students analyze how legal protections for digital assets support innovation, shape economic policy, and affect global trade relations.

Science

- Students investigate the role of open-source software in advancing scientific research by analyzing how these tools support data analysis, collaboration, reproducibility, and innovation within the scientific community. Through a structured report or digital presentation, students evaluate the technical and functional benefits of open-source platforms.

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:

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

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

7.IC.5 The student will evaluate the effect of Artificial Intelligence (AI) in various professions.

a. Research AI integration in various professions and evaluate its impact on the job market and society.
b. Examine and analyze the impact on job creation and changes in employment needs based on the use of AI.
c. Evaluate and explain the benefits and drawbacks of the implementation of AI technologies in various professions.

Understanding the Standard

[7.IC.5 a] Artificial intelligence (AI) is a branch of computer science that aims to create intelligent agents, which are systems that can reason, learn, and act autonomously. An intelligent agent is software that has been designed to use artificial intelligence and machine learning to perform tasks without human intervention. AI systems are designed to mimic human intelligence, enabling them to perform tasks that would typically require human cognitive abilities.

Artificial intelligence is increasingly integrated across a wide range of professional fields through the use of data-driven algorithms, predictive analytics, and intelligent automation. In healthcare, AI supports diagnostic accuracy through image recognition models, predicts patient outcomes with supervised learning algorithms, and assists in drug discovery using natural language processing and data mining. In finance, AI systems are used for algorithmic trading, fraud detection through anomaly detection models, and personalized financial advising via recommender systems.

In manufacturing, AI enables predictive maintenance through sensor data analysis, optimizes supply chain operations with real-time data modeling, and supports quality control using computer vision. In education, AI applications include adaptive learning platforms powered by machine learning algorithms that personalize content delivery based on student performance data. In transportation, AI supports autonomous vehicle navigation using deep learning, computer vision, and reinforcement learning to interpret sensor data and make real-time driving decisions.

AI integration across domains requires attention to data quality, model training, interpretability, and ethical considerations. Data must be accurate, relevant, and representative to support effective model development. Model training must reflect the intended task and context. Interpretability ensures transparency in how decisions are made. Ethical considerations include accountability and responsible use.

Consider the following example of AI in professions:

- Healthcare: AI aids in diagnosing medical conditions, predicting patient outcomes, and even performing certain surgical procedures with precision.
- Finance: AI is used for fraud detection, algorithmic trading, and customer service through chatbots, enhancing efficiency and accuracy.

- Transportation: Self-driving vehicles and traffic management systems use AI to improve safety and reduce travel time.

[7.IC.5b] AI integration affects the job market by automating tasks and creating demand for new roles. Emerging opportunities require skills in digital literacy, data fluency, algorithmic thinking, model evaluation, and responsible data use. The shift emphasizes the need for AI-related competencies across industries.

Consider the following examples of the impact of AI on the job market:

- Job Displacement: AI can replace roles that involve repetitive, predictable tasks, such as data entry or assembly line work, posing a risk to jobs in manufacturing and administrative fields.
- Job Creation: The rise of AI creates new opportunities in areas like data science, machine learning, and AI ethics, which require specialized skills.
- Regional Differences: The impact varies across sectors and regions, with some areas experiencing shortages in AI talent and others seeing job reductions.

[7.IC.5c] AI implementation offers benefits such as increased efficiency, accuracy, and scalability across sectors including finance, healthcare, and logistics. It enables automation of tasks like data analysis and natural language processing. Drawbacks include algorithmic bias, data privacy concerns, and workforce displacement. These trade-offs require design considerations and oversight. Consider the following examples of benefits and drawbacks of AI in professions:

Benefits:

- Increased Efficiency: AI can process large amounts of data quickly, making it valuable in sectors that require speed and accuracy, like finance and healthcare.
- Enhanced Accuracy: In fields like diagnostics and fraud detection, AI reduces human error, improving outcomes.
- Support for Skilled Work: AI can assist professionals by handling routine tasks, allowing them to focus on complex, strategic work.

Drawbacks:

- Job Displacement: Routine roles are at risk of automation, creating concerns about workforce reduction in certain sectors.
- Skill Demands: As AI transforms industries, workers need advanced skills like critical thinking and problem-solving to stay competitive.
- Ethical and Privacy Concerns: AI raises questions about data privacy, algorithmic bias, and the ethical use of technology, particularly in sensitive fields like law enforcement and healthcare.

Concepts and Connections

Concepts

AI integration across professions is reshaping the job market and society by automating tasks, enhancing productivity, and creating new opportunities. Examining changes in employment needs reveals shifts in job creation, skill requirements, and workforce dynamics. Evaluating the benefits and drawbacks of AI implementation highlights its potential to improve efficiency while also raising concerns about job displacement and ethical considerations.

Connections

Within the grade level/course: At this grade level, students research and examine how artificial intelligence (AI) is transforming the workforce across a range of industries, such as healthcare, finance, and manufacturing. Students explore how AI can enhance productivity and drive innovation, while also considering potential challenges, including job displacement and ethical concerns.

Vertical Progression: In Grade 6, students investigate and analyze the impact of the progression and advancement of AI technologies on industries (6.IC.6).

Across Content Areas

History and Social Science

- **CE.1** The student will demonstrate skills for historical thinking, geographical analysis, economic decision making, and responsible citizenship by b) analyzing how political and economic trends influence public policy, using demographic information and other data sources.

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 conduct research projects focused on careers in artificial intelligence, machine learning, or quantum computing. They will locate, evaluate, and compare multiple credible sources, analyzing aspects such as author intent and reliability. As part of their research process, students will create an annotated bibliography and maintain a glossary of domain-specific vocabulary (e.g., algorithm, neural network, quantum bit), using context clues and technical definitions to support understanding. Students will then synthesize their findings into a written or digital product that explains the nature of the career, required skills, and the broader impact of these technologies on society.

History and Social Science

- Students research how artificial intelligence is transforming careers across various industries and analyze the resulting effects on the U.S. economy. Using primary and secondary sources, students will examine trends such as job displacement, the creation of new roles, and shifts in workforce demands. Students will then present findings that evaluate how AI-driven economic change connects to broader concepts such as labor specialization, productivity, global trade, and economic interdependence.

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

3. Evaluate Resources and Recognize Contributions

Networks and the Internet (NI)

7.NI.1 The student will describe and explain why protocols are essential in data transmission.

a. Define packet, router, and protocol.
b. Describe the process of sending a file through a network.
c. Explain the role of Internet Protocol (IP) addresses in transmitting information.
d. Explain how packets ensure reliable communication among computing devices.
e. Model how data is transmitted over networks and the Internet.

Understanding the Standard

Protocols are standardized rules that govern how data is packaged, transmitted, received, and interpreted across a network. They enable communication and interoperability between devices, systems, and technologies.

[7.NI.1a] Protocols are standardized rules that define how devices communicate and exchange data over a network. They ensure that data is transmitted, received, and interpreted consistently across different systems. Protocols enable interoperability between devices by establishing uniform methods for packaging, addressing, and delivering information. This standardization is crucial for enabling seamless interactions between devices and networks.

- A protocol is the agreed-upon set of rules that governs how data is formatted, transmitted, received, and acknowledged between devices on a network.

Internet protocols manage data transmission by dividing information into packets and routing them through a network. Each element, such as packets and IP addresses, contributes to accurate delivery. Protocols support error detection and correction to maintain data integrity. They regulate timing and flow control to prevent congestion and enable efficient communication.

- A packet is a small unit of data that is a segment a larger message for transmission over a network, allowing the information to be sent efficiently and reassembled at the destination.
- A router is a networking device that directs packets along the most efficient path to their destination based on IP addresses.

[7.NI.1b] Sending a file through a network uses communication protocols such as TCP/IP. The file is divided into packets, each containing data and headers with source and destination IP addresses, sequence numbers, and error-checking codes. Routers direct the packets across the network based on current traffic conditions. At the destination, the system reassembles the packets, checks for errors, and reconstructs the file.

Process of Sending a File Over a Network:

- Step 1: The file is divided into packets.
- Step 2: Each packet is labeled with the sender and receiver's IP addresses.
- Step 3: Routers direct the packets across the network to their destination.
- Step 4: The packets are reassembled into the original file once they reach the recipient's device.

[7.NI.1c] IP addresses are numerical identifiers assigned to devices on a network. They enable accurate delivery of data packets to the correct destination. IP addresses follow IPv4 or IPv6 formats, providing different address capacities. Most devices receive dynamic IP addresses through network services, while static IP addresses are used for consistent access. Networking hardware uses IP addresses to determine routing paths for data transmission.

[7.NI.1d] Packets enable reliable communication by dividing data into smaller units for transmission. Each packet includes headers with routing and sequencing information. These headers support correct reassembly and identification of transmission order. Protocols such as TCP manage packet delivery using acknowledgments, error checking, retransmission, and flow control. This process ensures data integrity across complex networks.

- Error Checking: Each packet includes error-checking data, allowing the recipient device to verify its integrity.
- Retransmission: If a packet is lost, the system requests it again, ensuring all parts of the file arrive intact.

Concepts and Connections

CONCEPTS

Packets are small units of data transmitted over a network, routers direct these packets between devices, and protocols are the rules governing data exchange. Sending a file involves breaking it into packets, which travel through routers using IP addresses to reach the correct destination. Packets include information that ensures data is transmitted reliably, allowing devices to communicate accurately. Modeling data transmission demonstrates how networks and the Internet manage and deliver information efficiently.

CONNECTIONS

Within the grade level/course: At this grade level, students explore the essential role of protocols in data transmission, understanding how standardized rules facilitate reliable and efficient communication across networks. They learn about concepts such as packets, routers, and Internet Protocol (IP) addresses, and how these elements work together to ensure data integrity and security. This foundational knowledge not

only deepens their comprehension of computer science principles but also enhances their analytical and problem-solving skills across various disciplines.

Vertical Progression: In Grade 6, students use computation thinking skills to take what they already know about cloud computing and make decisions using that information (6.NI.1). In Grade 8, model and describe the role of computing devices in transmitting data in and on computing networks and the Internet (8.NI.1).

ACROSS CONTENT AREAS

English

- **7.W.2.A.iii** The student will defend conclusions or positions with reasons and precise, relevant evidence (e.g., facts, definitions, details, quotations, and examples).

History

- **CE.9.e** The student will evaluate multiple sources describing the same event and reflect on reasons for discrepancies.

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 an argumentative response on the importance of network security and reliable data transmission, using precise evidence to support their claims. They can research real-world examples of cybersecurity breaches or misinformation spread through unreliable data transmission, connecting their findings to the role of protocols in ensuring accuracy and integrity in digital communication.

History and Social Science

- Students can investigate how historical events have been communicated differently based on available technology, such as how telegrams, radio broadcasts, or the internet have influenced public understanding. They can then model how network protocols ensure data integrity by simulating the transmission of historical information in packets, demonstrating how errors, delays, or missing data can impact the accuracy of messages over time.

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 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
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 7 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.
World Wide Web (WWW)	A system of interconnected web pages that users can access through the internet.