Computer Science Implementation Framework

Grade 6-8

2026

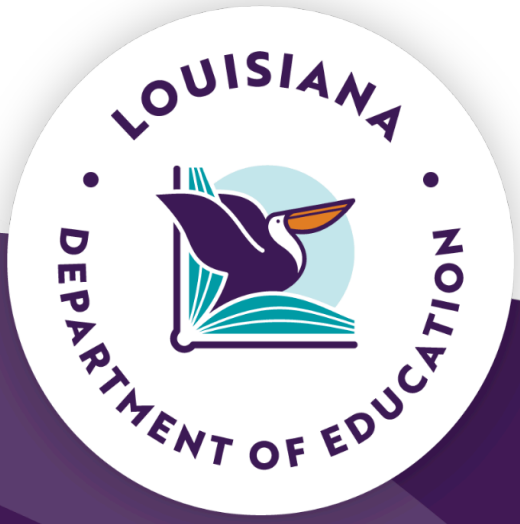

Table of Contents

Background

Louisiana Act 541 (2022), the Computer Science Education Act, aimed to establish a comprehensive and seamless statewide computer science education program across all educational levels to meet the demands of a dynamic economy. To achieve this goal, Act 541 created the Computer Science Education Advisory Commission (CSEAC). CSEAC was responsible for advising the State Board of Elementary and Secondary Education (BESE) through the state Department of Education (LDOE) to develop and implement a state action plan for delivering education in computer science in all public schools.

The first key action within the Louisiana K-12 Computer Science Education Plan calls for establishing content standards for computer science. From May to August 2024, the K-12 Computer Standards Writing Committee drafted student-centered and developmentally appropriate computer science standards within defined grade bands: elementary (K-5), middle school (6-8), and high school (9-12). These standards were carefully designed to reflect the increasing complexity and depth of understanding expected as students progress through each educational stage, with a focus on analytical and critical thinking skills, digital literacy, digital responsibility, and technology skills. In October 2024, BESE approved the K-12 Louisiana Student Standards for Computer Science. These standards now serve as a guiding framework to support high-quality computer science instruction in Louisiana.

Purpose of the Framework

Louisiana Act 211 (2024), known as the Computer Science Education Advancement Act, provides that each public school with students in grades six through eight shall provide instruction in exploratory computer science to its students.

The Louisiana Computer Science Implementation Framework for grades 6-8 supports Key Action 1 of the Louisiana K-12 Computer Science Education Plan by providing a structure for successfully implementing quality computer science instruction aligned to the K-12 Louisiana Student Standards for Computer Science. By utilizing the framework, teachers will equip students with essential analytical thinking, digital literacy, and technology skills needed for success in society and future career opportunities.

Overview of the Framework

The implementation of computer science instruction focuses on ensuring that all students have access to and engage with high-quality, standards-aligned learning experiences. This instruction is grounded in the K-12 Louisiana Student Standards for Computer Science, which serve as the foundational framework for curriculum design, lesson planning, and classroom delivery. These standards guide educators in introducing core computer science concepts, fostering computational thinking, and building problem-solving skills that prepare students for advanced study and real-world applications.

In middle grades, students build upon the foundational computer science skills acquired in elementary school by engaging with more complex applications. Students engage in more structured problem-solving, use programming to create functional solutions, and explore concepts such as networks, cybersecurity, and data analysis in greater depth. The emphasis shifts toward applying foundational skills to real-world scenarios and collaborative projects.

Core Concepts

Core concepts represent fundamental ideas in computer science that students explore throughout their education and enable students to develop a comprehensive knowledge of computer science. The standards are organized around these five core concepts.

Core Practices

Core practices encompass the skills and habits of mind that students develop while learning computer science. These seven practices are interconnected and reinforce one another, guiding students in analyzing problems, designing solutions, and communicating their understanding.

Core Concepts	Core Practices
1. Computing Systems	1. Fostering cyber responsibility
2. Networks and the Internet	2. Collaborating around Computing
3. Data and Analysis	3. Recognizing and defining computational problems
4. Algorithms and Programming	4. Developing and using abstractions
5. Impacts on Computing	5. Creating computational artifacts
	6. Testing and refining computational artifacts
	7. Communicating about computing

Core Concept 1: Computing Systems (CS)

Overview

Students interact with a wide variety of computers each day. Computers are devices that can collect, store, analyze, and act upon information in ways that can positively and negatively affect human capabilities. Humans (users) operate, program, and maintain the physical components (or hardware) and instructions (or software) that make up a computer. The hardware, software, and users are collectively called computing systems. Users leverage their understanding of hardware and software to resolve problems within computing systems in a process called troubleshooting.

Middle School Standards for Core Concept 1: Computing Systems

Subconcepts and Core Practices	Middle (6-8)
Subconcept 1: **Hardware and Software** *Core Practices:* *Collaborating Around Computing;* *Recognizing and Defining Computational Problems;* *Communicating about Computing*	M.CS.1A. Analyze the functions and interactions of core components within a computer system.
	M.CS.1B. Explain how hardware and software components work together to perform specific tasks.
Subconcept 2: **Troubleshooting** *Core Practices:* *Collaborating Around Computing;* *Recognizing and Defining Computational Problems;* *Testing and Refining Computational Artifacts;* *Communicating about Computing*	M.CS.2A. Evaluate possible solutions to a hardware or software problem.

Core Concept 2: Networks and the Internet (NI)

Overview

Computing systems typically do not operate in isolation. Networks connect computers to share information and resources and are integral to computer and data science. Networks enable critical communication for the computing systems that drive our economy and career sectors. The increased level of connectivity brought about by the internet provides fast and secure communication that facilitates innovation.

Middle School Standards for Core Concept 2: Networks and the Internet

Subconcepts and Core Practices	Middle (6-8)
Subconcept 1: **Hardware and Network Communication** *Core Practices: Fostering cyber responsibility; Collaborating around computing; Recognizing and defining computational problems; Developing and using abstractions; Creating computational artifacts; Testing and refining computational artifacts; Communicating about computing*	M.NI.1A. Analyze the various structures and functions of a network.
	M.NI.1B. Identify and differentiate the protocols utilized in data sharing across networks.
Subconcept 2: **Cybersecurity** *Core Practices: Fostering cyber responsibility; Collaborating around computing; Recognizing and defining computational problems; Developing and using abstractions; Creating computational artifacts; Testing and refining computational artifacts; Communicating about computing*	M.NI.2A. Analyze threats and vulnerabilities to information security for individuals and organizations.
	M.NI.2B. Explain how physical and digital security practices and measures proactively address threats to users, data, and devices within and across networks.

Core Concept 3: Data and Analysis (DA)

Overview

Computing systems function by processing and storing data. Data is abundant due to the growing number of connected devices worldwide. As the volume of data expands, so does the demand for accurate and efficient data analysis methods. Data science is the cross-disciplinary use of data to inform decision-making, test hypotheses, predict trends, and develop precise models that drive innovation across industries.

Middle School Standards for Core Concept 3: Data and Analysis

Subconcepts and Core Practices	Middle(6-8)
Subconcept 1: **Data Representation** *Core Practices: Collaborating around computing; Recognizing and defining computational problems; Creating computational artifacts*	M.DA.1A. Analyze and explain the connection between data sets and graphical representations.
	M.DA.1B. Evaluate the most efficient and effective ways to arrange, collect, and visually represent data to inform others.
Subconcept 2: **Data Collection** *Core Practices:* *Core Practices: Fostering cyber responsibility; Collaborating around computing; Recognizing and defining computational problems; Developing and using abstractions; Creating computational artifacts; Testing and refining computational artifacts; Communicating about computing*	M.DA.2A. Compare and contrast how data is collected using computational and non-computational tools and processes.
	M.DA.2B. Analyze scenarios and computing systems to determine the appropriate data entry format for specific tasks.
Subconcept 3: **Data Storage** *Core Practices: Fostering cyber responsibility; Collaborating around computing; Recognizing and defining computational problems; Developing and using abstractions; Creating computational artifacts; Testing and refining computational artifacts; Communicating about computing*	M.DA.3A. Propose methods to back up data safely and the appropriate practices for data risk management.
	M.DA.3B. Describe how different representations of real-world phenomena, such as letters, numbers, and images, are encoded as data.
Subconcept 4: **Visualizations and Transformations** *Core Practices: Creating computational artifacts; Communicating about computing*	M.DA.4A. Utilize tools and techniques to locate, collect, and create visualizations of large-scale data sets.
	M.DA.4B. Collect and transform data using computational tools to make functional and

Subconcepts and Core Practices	Middle(6-8)
	reliable data for use in hypothesis testing.
Subconcept 5: **Inference and Models** *Core Practices: Recognizing and defining computational problems; Developing and using abstractions; Creating computational artifacts; Testing and refining computational artifacts; Communicating about computing*	M.DA.5A. Refine computational models based on data generated by the models.
	M.DA.5B. Describe and evaluate the accuracy of a modeled system by comparing the generated results with observed data from the system the data represents.

Core Concept 4: Algorithms and Programming (AP)

Overview

An algorithm is a sequence of steps designed to accomplish a specific task. Algorithms are translated into programs, or code, to provide instructions for computing systems. Algorithms and programs control all computing systems, empowering people to communicate with the world in novel ways and solve compelling problems. The development process to create meaningful and efficient programs involves choosing which information to use, how to process the data, and how to store the information. The decomposition of more significant problems into simpler ones, combined with recombining existing solutions and analyzing various solutions, helps determine the most appropriate solution to a problem.

Middle School Standards for Core Concept 4: Algorithms and Programming

Subconcepts and Core Practices	Middle (6-8)
Subconcept 1: **Variables and Algorithms** *Core Practices: Collaborating around computing; Recognizing and defining computational problems; Developing and using abstractions; Creating computational artifacts; Testing and refining computational artifacts; Communicating about computing*	M.AP.1A. Evaluate and use naming conventions for variables to accurately communicate the variables' meaning to other users and programmers.
	M.AP.1B. Evaluate algorithms in terms of efficiency, correctness, and clarity.
	M.AP.1C. Compare and contrast data constants and variables.
Subconcept 2: **Control Structures** *Core Practices: Collaborating around computing; Recognizing and defining computational problems; Developing and using abstractions; Creating computational artifacts; Testing and refining computational artifacts; Communicating about computing*	M.AP.2A. Explain the functions of various control structures. Compare and contrast examples of control structure types.
	M.AP.2B. Design and iteratively develop programs that combine control structures into advanced control structures.
Subconcept 3: **Modularity** *Core Practices: Collaborating around computing; Recognizing and defining computational problems; Developing and using abstractions; Creating computational artifacts; Testing and refining computational artifacts; Communicating about computing*	M.AP.3A. Decompose problems to facilitate program design, implementation, and review.
	M.AP.3B. Create procedures with parameters to organize code and promote reusability.

Subconcepts and Core Practices	Middle (6-8)
Subconcept 4: **Program Development** *Core Practices: Fostering cyber responsibility; Collaborating around computing; Recognizing and defining computational problems; Developing and using abstractions; Creating computational artifacts; Testing and refining computational artifacts; Communicating about computing*	M.AP.4A. Seek and incorporate feedback from peers to employ user-centered design solutions.
	M.AP.4B. Use applicable industry practices to test, debug, document, and peer review code.
	M.AP.4C. Develop computational artifacts by working as a team, distributing tasks, and maintaining an iterative project timeline.
	M.AP.4D. Incorporate existing resources into original programs and give the proper attributions.
	M.AP.4E. Systematically test, document outcomes, and refine programs using a range of test cases.

Core Concept 5: Impacts of Computing (IC)

Overview

The impacts of computing can be positive and negative, enabling innovation, communication, and access to information, while also raising concerns around ethics, privacy, and fairness. Individuals and communities not only shape computing systems through interactions, behaviors, industry practices, and laws, but are also shaped by interactions with computing systems. Computer and data science create new means of communication by accelerating information exchange and establishing cyberspace as a dynamic workspace for individuals.

Middle School Standards for Core Concept 5: Impacts of Computing

Subconcepts and Core Practices	Middle (6-8)
Subconcept 1: **Intellectual Achievements** *Core Practices: Fostering cyber responsibility; Collaborating around computing; Testing and refining computational artifacts; Communicating about computing*	M.IC.1A. Identify foundational computational advancements through the use of the technology innovation cycle.
	M.IC.1B. Plan and devise new ideas and solutions for problems with inspiration from previous discoveries in computational knowledge.
Subconcept 2: **Social Interaction** *Core Practices: Fostering cyber responsibility; Collaborating around computing; Testing and refining computational artifacts; Communicating about computing*	M.IC.2A. Develop and propose norms for informal versus formal online communications.
	M.IC.2B. Analyze communication technologies and then describe how the technology influences individuals and society.
	M.IC.2C. Generate designs that increase the accessibility and usability of technology for various groups of users.
Subconcept 3: **Laws, Safety, and Industry Practices** *Core Practices: Fostering cyber responsibility; Collaborating around computing; Testing and refining computational artifacts; Communicating about computing*	M.IC.3A. Identify applicable laws that impact personal, industry, or business computing practices.
	M.IC.3B. Recommend and propose computing-use guidelines to maintain a user's personal safety, privacy, and well-being.
	M.IC.3C. Describe and categorize factors that affect users' access to computing resources locally, nationally, and globally.

Implementing the Core Practices in Computer Science Instruction

The middle school standards are not only about learning content and correct terminology, but also about mastering the seven core practices. The core practices are essential skills and professional mindsets used by those in computer science careers. By intentionally weaving practices such as Collaborating Around Computing and Testing and Refining Artifacts into computer science instruction, teachers can support the transformation of students from passive consumers of technology into capable, critical-thinking creators and responsible digital users. This framework breaks down the core practices into actionable instructional strategies to ensure deep learning and successful preparation for high school computer science.

Fostering Cyber Responsibility

As students engage with digital tools and online spaces, they must develop an understanding of what it means to exhibit and practice cyber responsibility. Cyber responsibility involves navigating digital spaces thoughtfully, fairly, and safely. Other important aspects of cyber responsibility include safe behavior, respect for others, and protecting personal information. This practice also involves navigating complex issues, such as digital footprints, ethical use of technology, cyberbullying, and the impact of online actions.

Core Practice 1: Fostering Cyber Responsibility

Best Practices	Classroom Applications	Additional Considerations
Integrate ethics, safety, and privacy discussions directly into content lessons and project development.	Units and projects include explicit moments where students analyze the impacts of the technology being used or created (Core Concept 5). This includes: Analyzing data collection practices when studying (Core Concept 3), and applying network safety protocols (Core Concept 2).	Cyber responsibility should not be treated as a single, isolated unit at the beginning or end of the semester. It is not sufficient to simply provide a checklist of "don'ts" without connecting it to the underlying technical reasons (e.g., "Don't share data" without explaining how data leakage occurs).
Encourage consistent, ethical attribution and licensing analysis for all incorporated resources.	Students develop an understanding of intellectual property. Every time a student uses an external resource (an image, sound, Application Programming Interface API data, or external code), legal licenses should be reviewed and proper citations included.	Use of copyrighted images or code without any regard for licensing or attribution should be avoided. A crucial element of cyber responsibility is the attribution of all intellectual property used to create a computational artifact. This includes code, algorithms, software libraries, and any digital assets such as images and models.

Best Practices	Classroom Applications	Additional Considerations
Guide students to develop and adhere to explicit norms for online communication and collaboration tools.	Students distinguish between casual, informal digital dialogue (e.g., social media) and the formal, respectful language required for peer feedback, shared code documentation, or professional communication. This means establishing and enforcing "Netiquette" guidelines for all digital project interactions.	Peer communication and interactions do not have to be formal (like a business letter), but should be respectful and meet expectations for digital responsibility. Effective communication and constructive criticism should be modeled.
Focus on proactive, preventative digital security skills over fear-based warnings.	Instruction should be centered on teaching students *why* specific security measures work, connecting to the concepts of threats and vulnerabilities (Core Concept 2). This includes teaching the mechanics and importance of multi-factor authentication (MFA) and how to responsibly identify and react to common threats like phishing.	Fear-based tactics and horror stories or outdated security protocols should be avoided. The goal is to empower students with knowledge of digital self-defense, such as strong passwords, authentication methods, security hygiene, and threat awareness.

Collaborating around Computing

Students develop collaborative computing skills through group work to solve problems and create digital products. Through work with others, shared responsibilities, peer feedback, and communication, students develop stronger outcomes than those achieved by working alone. Collaboration involves navigating various viewpoints, resolving disagreements, and contributing unique strengths. Collaborative computing relies on using digital tools that support coordination, co-creation, and project management. These skills enable teams of students to build more effective and innovative solutions together.

Core Practice 2: Collaborating around Computing

Best Practices	Classroom Applications	Additional Considerations
Implement structured group work models for problem-solving and creating computational artifacts.	Collaboration is intentional and structured. This requires using methods like pair programming, where students rotate between defined roles, and assigning clear, distributed tasks when working in larger groups.	Projects should not be assigned with the expectation that students will divide the work evenly. Project-specific roles may need to be created and assigned to students. Documentation and digital footprints should confirm that input, workload, and intellectual contribution are shared.
Integrate peer review, testing, and feedback loops as part of the grading process.	Collaboration extends beyond initial creation to include review and refinement (Core Practice: Testing and Refining). Students use commenting to provide specific, actionable feedback on their peers' designs and code logic, following established netiquette norms (M.AP.4A). Teachers can monitor the live document to assess student collaboration.	Grades should not be based solely on the final product. The quality of the collaborative process and peer feedback should be assessed as well. Require students to follow a formal review process and receive documented consent before editing peers' code.
Model and require the use of collaborative applications for managing projects, creating artifacts, and coding.	Students need to be proficient in tools for shared code editing, version tracking, and synchronous documentation. This ensures that when tasks are distributed among a team, work is merged seamlessly and changes are tracked.	Artifacts are not limited to code. Students should collaborate and have shared access when developing the project plan, providing documentation, and code.

Best Practices	Classroom Applications	Additional Considerations
Establish roles and expectations when structuring projects for effective team collaboration and conflict resolution.	When working in teams, students learn to distribute tasks (M.AP.4C) and manage an iterative project timeline. Through decomposition, tasks can be divided into modules or subsystems that allow team members to work simultaneously on different parts. Teachers guide students in setting deadlines, defining project scope, and developing strategies for respectfully resolving disagreements.	Conflict resolution skills should not be assumed. With student input, create a framework for conflict resolution. When planning, ensure projects are complex enough to require teamwork and task distribution.

Recognizing and Defining Computational Problems

Computational thinking requires students to clearly define a problem, break it into manageable parts, and evaluate whether computing can be used to solve the computational problem effectively. This practice includes pattern recognition, designing logical steps, and identifying opportunities where technology can offer meaningful solutions. Recognizing and defining computational problems requires analytical thinking and creative problem-solving to determine when and how computing methods can be used in real-world contexts.

Core Practice 3: Recognizing and Defining Computational Problems

Best Practices	Classroom Applications	Additional Considerations
Start projects with open-ended prompts and challenges.	Start with a real-world scenario or an ambiguous task instead of a perfectly formed prompt. Students should begin by defining the specific inputs, outputs, processes, and constraints of the problem before proposing a solution. Emphasize the importance of analyzing the user's needs (user-centered design) before proposing a technical solution.	Foster independent problem-solving by using open-ended scenarios. Encourage students to lead the discovery process, mapping out their strategies and intentions as a foundation for their technical work.
Model decomposing big problems into smaller, manageable tasks.	Decomposition is a fundamental problem-solving skill. Students practice breaking a large problem into smaller, solvable components (M.AP.3A). Beyond modeling, teachers can provide planning templates for students to identify the main goals and define functions or procedures before coding.	While a large, single block of code might technically solve a complex problem, it does not provide the career-ready skills that students need. Moving beyond 'code that just works' to structured, manageable components prepares students for the complexity and rigor of real-world software development.
Emphasize distinguishing between computational and non-computational parts of a solution.	Students can determine which parts of a problem require a computer program (e.g., calculations, data filtering, complex logic) and which parts are best solved by humans or existing tools (e.g., gathering initial input, interpreting final results).	Code is not necessary to solve every aspect of a problem. A non-computational tool could be more efficient for some tasks. For example, tasks such as brainstorming ideas or using mental math are better handled by humans.

Best Practices	Classroom Applications	Additional Considerations
Focus on finding and defining the necessary data to solve the problem.	Before solving, students identify what data is needed, its source, its required format, and how it will be stored and manipulated (M.DA.2B).	High-quality data collection requires planning the requirements, structure, and entry format. Define how to handle missing, messy, or unreliable data as part of the problem definition to ensure any results are based on verified and reliable information.

Developing and Using Abstractions

Abstraction is the process of managing complexity by identifying patterns and creating generalizations. Users recognize similarities across different examples and use these insights to develop reusable solutions. By simplifying problems and focusing on essential details, abstraction streamlines the problem-solving process.

Core Practice 4: Developing and Using Abstractions

Best Practices	Classroom Applications	Additional Considerations
Encourage the use of procedures/functions (modularity) with parameters in all programs.	Students learn to group related instructions into named blocks of code (procedures/methods/functions) to solve smaller sub-problems (M.AP.3B). The ability to reuse these defined blocks allows students to move from specific, one-time instructions to a broader generalized logic, demonstrating how abstracting a process makes code more efficient and adaptable.	Focus on building efficient code by organizing repeating steps into reusable sections. Functions should be utilized for any sequence of steps that is repeated, not only for simple or minor tasks.
Utilize models, diagrams, and pseudocode to represent complex systems or data.	Abstraction is a thinking tool that extends beyond coding. Students can use flowcharts, block diagrams, or pseudo-code to represent the *logic* of a program or the structure of a system (M.AP.3A). This approach allows students to concentrate on the step-by-step reasoning (the algorithm) by concealing the syntax-specific details.	Designing the overall strategy for addressing the problem is a necessary first step before writing code. Modeling is a core part of computational design. Prioritizing this stage helps students avoid logical errors and build stronger computational thinking skills.
Teach students to generalize specific solutions to broader problems.	The goal is to move beyond solving a singular instance of a problem (e.g., calculating the cost of 5 apples) to creating a general solution (e.g., a procedure that calculates the cost for *any* item and *any* quantity). This requires defining inputs and outputs via parameters (M.AP.3B).	In addition to solving new, complex problems, students should also be taught to adapt a pre-existing, abstract solution to a different context. Design solutions that accommodate various inputs and datasets to ensure programs remain flexible and avoid single-set or hard-coded configurations.

Best Practices	Classroom Applications	Additional Considerations
Introduce data types and variable naming conventions as forms of abstraction.	Variable names (M.AP.1A) are a fundamental abstraction that represent a complex piece of data with a simple, human-readable name. Teaching data types (M.AP.1C) addresses how the computer encodes different real-world data (text, numbers, images) into a common format (binary).	Encourage the use of descriptive variable names to improve code readability and logic, avoiding the use of unclear or single-letter variable names (e.g., x, y, temp). Shift the focus of binary instruction from manual calculation to the broader concept of data abstraction and how the computer simplifies and categorizes complex information.

Creating Computational Artifacts

Computational artifacts can take many forms, such as programs, simulations, animations, visualizations, apps, or robotic systems, and serve as tools for innovation and problem-solving. Creating computational artifacts allows students to explore ideas, express creativity, and develop solutions to meaningful problems. Artifacts can be created by remixing existing components or designing entirely new ones. By designing and remixing existing components or creating entirely new solutions, students use computational artifacts to bring their ideas to life and address challenges with meaningful solutions.

Core Practice 5: Creating Computational Artifacts

Best Practices	Classroom Applications	Additional Considerations
Create projects that result in a tangible, demonstrable digital output.	Coursework should include assignments that result in computational artifacts. This could be culminating tasks or projects in which students create a program, app, or model that serves a clear purpose (M.AP.4C). The shift from theoretical understanding to practical skill implementation allows students to gain more complete mastery of computational concepts.	Emphasize authentic assessment strategies that value the process of creation and problem-solving. While traditional quizzes can support quick checks for understanding, the core of evaluation should rely on the student's ability to apply their knowledge in meaningful, hands-on contexts.
Provide clear goals, constraints, and rubrics focused on functionality and design.	Artifact types may vary across assignments, but clear parameters should be provided. Rubrics should value coding logic, efficient algorithms, user-centered design, functionality, and effective problem solving (M.AP.4A).	Successful computational design requires well-defined objectives and project scope. Encourage students to design with the end-user in mind, ensuring that their completed programs are documented and organized well enough to be used by others.

Testing and Refining Computational Artifacts

Testing and refinement are essential parts of developing high-quality computational artifacts. This iterative process involves identifying and correcting errors, comparing results to intended outcomes, and making adjustments to improve performance. The practice also includes consideration of user needs and feedback, and refining artifacts to enhance reliability, usability, and accessibility.

Core Practice 6: Testing and Refining Computational Artifacts

Best Practices	Classroom Applications	Additional Considerations
Support systematically testing programs using a defined range of test cases.	Students develop a structured approach to testing. Before declaring a project complete, students must define a set of specific inputs (test cases) that target different parts of the code, especially boundary conditions and potential error states. Expected outcome versus the actual outcome for each test should be documented. (M.AP.4E).	Code testing is a valuable and rigorous process, requiring systematic effort to ensure quality. An artifact should be accepted only after the student or group has demonstrated that all intended features function reliably and effectively across a variety of relevant conditions.
Explicitly teach and model effective debugging strategies.	Debugging is an important skill that drives innovation and continuous improvement. Teachers should model strategies such as step-through execution and isolation of problematic code sections. Console logs or print statements can also be used to trace variables.	Provide detailed feedback that helps students diagnose and clearly explain the cause of bugs. This supports a deeper learning journey rather than just deleting code until errors disappear.
Model and incorporate peer/user feedback loops for refinement.	Feedback should be used to refine artifacts, focusing on usability, design, and clarity (M.AP.4A). Dedicated, structured time should be provided for students to solicit feedback from end-users and apply it in the subsequent development iteration.	Constructive feedback is valuable, and initial designs are opportunities for improvement. An artifact with known user experience concerns should be refined to ensure a better interaction, even if the code functions correctly.

Best Practices	Classroom Applications	Additional Considerations
Model professional documentation of known issues and limitations (bugs) as part of the maintenance process.	To cultivate professional accountability, students must systematically identify and document all issues, known bugs, and limitations in their work. This documentation requires students to explain the root cause of each issue.	Assessment of this practice should focus on rewarding the process of identifying and attempting to correct errors, not solely on a flawless final product. This encourages students to embrace mistakes as a natural and valuable part of the learning and problem-solving process.

Communicating about Computing

Communication is a key part of computer science, facilitating both personal expression and collaboration. Sharing ideas clearly with a variety of audiences, explaining how and why computation is used, and reflecting on the impact better equips stakeholders for collaborative projects. Effective communication also entails writing meaningful comments, documenting work, and presenting thinking through different forms of media. Emphasis is placed on using precise language and tailoring communication to meet the needs of the intended audience.

Core Practice 7: Communicating about Computing

Best Practices	Classroom Applications	Additional Considerations
Implement professional-level internal and external documentation for all code.	Students provide clear, concise comments within their code to explain logic, variable usage, and complex steps. Externally, create documentation (e.g., README files) justifying design choices by explaining how the program works, runs, and primary features (M.AP.4B).	Code should be accepted when it is functional and includes clear, helpful comments and documentation. This documentation should be concise and provide useful explanations, rather than simply restating the obvious or being overly lengthy.
Encourage oral or written analyses and reports that explain design rationale.	Students practice defending their solutions. This includes writing reports or delivering presentations that analyze their project's effectiveness, detailing why specific algorithms, control structures, or data formats were chosen, and how principles of abstraction were applied.	Acceptable project submissions should include thorough documentation. This documentation should not only detail the project's features but also provide an analysis of the computational principles that support the design.
Communicate with students at different academic levels to peer review.	Demonstration of proficiency includes the ability to teach others. This can be accomplished by creating resources like simple video tutorials, detailed help guides, or comprehensive walkthrough documents for peers who need assistance understanding complex code modules developed by other students.	Communication should extend beyond the team. This practice is essential for understanding how projects are perceived by external audiences. Sharing and receiving feedback is an essential skill for successful collaboration.

Best Practices	Classroom Applications	Additional Considerations
Use precise, technical vocabulary in all written and oral discussions.	Promote and evaluate students' correct application of computing terminology (e.g., using "iteration" instead of "looping," "protocol" instead of "rules," "variable" instead of "box for stuff"). This practice demonstrates a grasp of the relevant specialized vocabulary.	Emphasis goes beyond simple memorization of vocabulary. Students must be able to demonstrate their understanding by applying concepts.

Considerations for Computer Science Implementation

Implementing computer science in grades 6-8 requires strategic planning to guarantee long-term success. Strategic planning ensures a learning progression that aligns middle school foundations with high school expectations and provides access for all students. When determining the best implementation model to fit the needs of schools and students, systems should consider the following factors:

- Cost and Funding
    - What is the total cost for teachers to prepare for and deliver quality computer science experiences? Costs may include:
        - Professional learning or development in a curriculum or towards certification
            - Teacher access licenses
            - Training registration costs
            - Teacher expenses and stipends
    - Consider the overall impact of the investment. Is the program designed to bridge across multiple grade levels (K-5 to 9-12) to ensure progression, or is it grade-level specific?
    - What is the total cost of materials needed for the number of students participating in the quality CS experiences? Materials may include:
        - Hands-on student materials
            - Support items required to connect with digital programs (e.g., calculator or micro: bits, using some form of hands-on device that connects with the program)
            - Consumables and non-consumables
        - Digital devices or other technology
            - Student access codes
            - Platform usage
    - How much money is available to purchase high-quality materials for both students and the teacher? Is there a plan to maintain and expand the program after the purchase of initial materials is complete? Funding sources may include:
        - Federal or grant funding
        - Classroom/school donations
        - Partnerships with local industry and the community
- Data Privacy
    - Does the program meet federal/state/district laws to protect student privacy? Applicable laws can include:
        - FERPA (Family Educational Rights and Privacy Act)
        - COPPA (Children's Online Privacy Protection Act)
        - ACT 837
        - Parental Agreements/Consents

    - What is necessary to ensure all work with vendors complies with state and system policies? This may include:
        - Vendor Agreements or data protection contracts
        - LDOE Data Sharing Agreements
    - Does the vendor provide additional enhanced data security measures? Data security measures may include:
        - MultiFactor Authentications (MFA)
        - Breach response plans
- Additional Resource Considerations
    - What resources are available to set up these quality computer science experiences for students?
        - Human resources - computer science teacher, teacher/staff members, parent volunteers
        - Local resources - industry and business partners through CTE/Career Opportunities/District Technology Centers
        - Teachers must ensure that local school system networks permit access to the resources and create website-required teacher accounts.