washington
STATE LEARNING STANDARDS

kindergarten – 12th grade

Computer Science

Revised 2018

Foreward

The following Washington State Computer Science K-12 Learning Standards are modified to reflect the 2017 revisions to the Computer Science Teachers Association (CSTA) K–12 Computer Science Standards.

Substitute House Bill 1813 (2015 Legislative session) tasked the Office of the Superintendent of Public with the adoption of national computer science standards. To fulfill the legislative request, the CSTA Computer Science K–12 Interim Standards were adopted in December 2016 with the caveat that Washington would enact changes once CSTA revisions were completed. The CSTA final standards were published in August 2017.

The CSTA revisions aligned the standards with the K–12 Computer Science Framework; standards were reworked to be measurable and provide a clear progression from kindergarten through 12th grade in the concepts and subconcepts of the framework. Final CSTA standards are no longer copyrighted, but instead hold a Creative Common License. For more information: https://www.csteachers.org/page/StandardsProcess.

Based on support from OSPI's Curriculum Advisory and Review Committee, statewide computer science stakeholders and leadership, and unanimous public comment, revisions to the the Washington State Computer Science K–12 Learning Standards have been made. Those new standards are reflected in the following pages and will support computer science instruction statewide.

OSPI and its organizational partners will continue to provide professional learning opportunities to support implementation of the standards.

Shannon Thissen
Computer Science Program Supervisor

SUPERINTENDENT OF PUBLIC INSTRUCTION

Randy I. Dorn Old Capitol Building · PO BOX 47200 · Olympia, WA 98504-7200 · http://www.k12.wa.us

Computer Science K–12 Learning Standards
Adoption Statement

The ***2016 Computer Science K–12 Learning Standards*** were developed collaboratively with teachers, administrators, subject matter experts, state and national associations, and stakeholders in computer science. Teams of Washington teachers, technology integration specialists, and librarians reviewed national standards to determine needs for Washington students.

Since the first draft was made available in December 2015, the Computer Science K–12 Learning Standards have been reviewed by Washington educators, administrators, and family members. The standards underwent a Bias and Sensitivity Review and a Public Comment Period, providing those with a stake in computer science education an opportunity to inform the development and implementation of the standards and supporting documents.

Pursuant to Substitute House Bill (SHB) 1813 and based on support from educators, OSPI's Curriculum Advisory and Review Committee, and statewide computer science stakeholders, I hereby adopt the ***Computer Science K–12 Learning Standards***.

Adopted on this 8th day of December, 2016.

Randy Dorn

Randy I. Dorn
State Superintendent
of Public Instruction

Computer Science K–12 Learning Standards

Office of Superintendent of Public Instruction

Shannon Thissen, Program Supervisor
Computer Science

Kathe Taylor, Ph.D., Assistant Superintendent
Learning and Teaching

Revised 2018

License

Attribution

The CSTA K–12 Computer Science Standards are created and maintained by members of the Computer Science Teachers Association (CSTA).

The Association for Computing Machinery (ACM) founded CSTA as part of its commitment to K–12 computer science education. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Suggested citation: Computer Science Teachers Association (2017). CSTA K–12 Computer Science Standards, Revised 2017. Retrieved from http://www.csteachers.org/standards.

The K–12 Computer Science Framework, led by the Association for Computing Machinery, Code.org, Computer Science Teachers Association, Cyber Innovation Center, and National Math and Science Initiative in partnership with states and districts, informed the development of this work.

The CSTA Standards Revision Task Force crafted standards by combining concept statements and practices from the Framework. The Task Force also used descriptive material from the Framework when writing examples and clarifying statements to accompany the standards. The glossary referenced in the navigation header links directly to the Framework's glossary.

For more information about the Framework, please visit k12cs.org.

Table of Contents

Computer Science Is an Essential Academic Subject

The mission of the Office of Superintendent of Public Instruction (OSPI) is to "provide funding, resources, tools, data and technical assistance that enable educators to ensure students succeed in our public schools, are prepared to access post-secondary training and education, and are equipped to thrive in their careers and lives." Our vision is that "every student is ready for college, career, and life." Effective and relevant computer science education is essential to achieving these aims. While attention to computer science education has increased in recent years, a lack of awareness about its content and potential impact is widespread. The new Washington State Computer Science K–12 Learning Standards are designed to enhance teacher understanding and improve student learning so that students are better equipped for college, career, and life.

Washington is committed to implementing high-quality computer science instruction to:

- Increase the opportunity for all students to gain knowledge of computer science.
- Introduce the fundamental concepts and applications of computer science to all students, beginning at the elementary school level.
- Make computer science at the secondary level accessible, worthy of a computer science credit, and/or equivalent to math and science courses as a required graduation credit (see Level 3B of computer science standards).
- Offer additional secondary-level computer science instruction that allows interested students to study facets of computer science in depth and prepare them for entry into a career or college.

Washington State Learning Goals, Standards, and Outcomes

Learning standards are for all of us: students, principals, administrators, decision-makers, community partners, teachers, and families. They help define what is important for students to know and be able to do as they progress through school. Standards help ensure that students acquire the skills and knowledge they need to achieve personal and academic success. Standards also provide an avenue for promoting consistency in what is taught to students across our state—from district to district, school to school, and classroom to classroom.

These **four learning goals** are the foundation of all academic learning standards in Washington:

1. **Read** with comprehension, **write** effectively, and **communicate** successfully in a variety of ways and settings and with a variety of audiences.
2. **Know and apply the core concepts and principles** of mathematics; social, physical, and life sciences; civics and history, including different cultures and participation in representative government; geography; arts; *and health and fitness* [now named physical education].
3. **Think** analytically, logically, and creatively, and to integrate technology literacy and fluency as well as different experiences and knowledge to form reasoned judgments and solve problems.
4. **Understand** the importance of work and finance and how performance, effort, and decisions directly affect **future career and educational opportunities**.

The **Washington State K–12 Learning Standards** are the required elements of instruction and are worded broadly enough to allow for local decision-making. Depending on school resources and community norms, instructional activities may vary. The 2018 Computer Science K–12 Learning Standards reflect OSPI's continuous commitment to supporting rigorous, inclusive, age-appropriate, accurate instruction to ensure that students are prepared to live productive and successful lives in a global society.

The computer science standards provide guidance to teach, reinforce, and apply the state's learning goals. They are aligned vertically to strengthen application of learning and depth of knowledge. If implemented effectively, these standards and outcomes will help students to understand and apply knowledge and skills necessary to thrive in a global economy and to be successful learners across other academic disciplines.

Learning Standards: Equity, Access, Inclusion, and Diversity

Computer science, among other STEM disciplines, can provide the knowledge and skills to empower individuals to create technologies with broad influence and impact. Women, underrepresented minorities, and people with disabilities are often missing in computer science classes, majors, and occupations. Limited access to technology due to geography or poverty can also restrict access and opportunities. A lack of diversity limits the scope of problems being addressed and the ability of new tools and technologies to reach multiple audiences.

One way to address this opportunity gap is by increasing access, inclusion, and opportunities for all students to learn computer science.

All students need to understand a world that is increasingly influenced by technology and to apply computing as a tool for learning and expression in a variety of disciplines and interests. Computer science and computational thinking, essential 21st Century Skills that increase a student's readiness for careers and college in any field, can be integrated into any discipline. The High School and Beyond Plan, required of all students to graduate, is the place to identify individual student goals for career, college, and life. Computer science offers a strong foundation for students to attain their goals.

Equity is embodied in the standards through both concepts and practices. For example, *Impacts of Computing* is a core concept aimed at promoting ideas about equity. *Fostering an Inclusive and Diverse Computing Culture* is an example of a core practice that promotes equity in K–12 computer science. Equity in computer science is not just about an equitable K–12 Computer Science Framework to implement the standards, but also about subsequent initiatives such as curriculum development, teacher preparation, access to tools and equipment, and integrated instruction.

Computer science courses and modules present a distinct opportunity to educate students about diversity, equity, and inclusion. As noted above, the standards and supporting framework provide explicit content about inclusive and diverse computing cultures. Students can engage in thoughtful interaction about the value of diversity while using computational thinking to develop computer artifacts to solve real-world problems. Educators and students can challenge implicit bias, stereotypes about computer science, and narrow perspectives while learning about core concepts like networks and security, data analysis, and impacts of computing because the cross-cutting themes of equity and inclusion are embedded in the framework.

Computer Science K–12 Learning Standards

The 2015 Washington State Legislature required the adoption of nationally-recognized computer science standards. The enactment of this law coincided with the development of a national computer science framework by K12CS.org. The Computer Science K–12 Learning Standards reflect the recommendations of the K–12 Computer Science Framework, led by the Association for Computing Machinery, Code.org, Computer Science Teachers Association, Cyber Innovation Center, and National Math and Science Initiative in partnership with states and districts. The K–12 Computer Science Framework is endorsed by leading industry and educational organizations as well as K–12, higher education, and research leaders in the field of computer science education.

The standards are meant to establish a baseline literacy in computer science for all students and provide guidance for designing curriculum, assessments, and teacher preparation programs. It consists of five core concepts and seven core practices, as listed:

Core Concepts

1. Computing Systems
2. Networks and the Internet
3. Data and Analysis
4. Algorithms and Programming
5. Impacts of Computing

Core Practices

1. Fostering an Inclusive and Diverse Computing Culture
2. Collaborating
3. Recognizing and Defining Computational Problems
4. Developing and Using Abstractions
5. Creating Computational Artifacts
6. Testing and Refining
7. Communicating

The Computer Science K–12 Learning Standards and connected framework represent a vision in which all students, from a young age, engage in the concepts and practices of computer science to understand a world that is increasingly influenced by technology and to apply computing as a tool for learning and expression in a variety of disciplines and interests. From kindergarten through 12th grade, students will develop new approaches to problem solving that harness the power of computational thinking, while not only becoming users, but creators of computing technology.

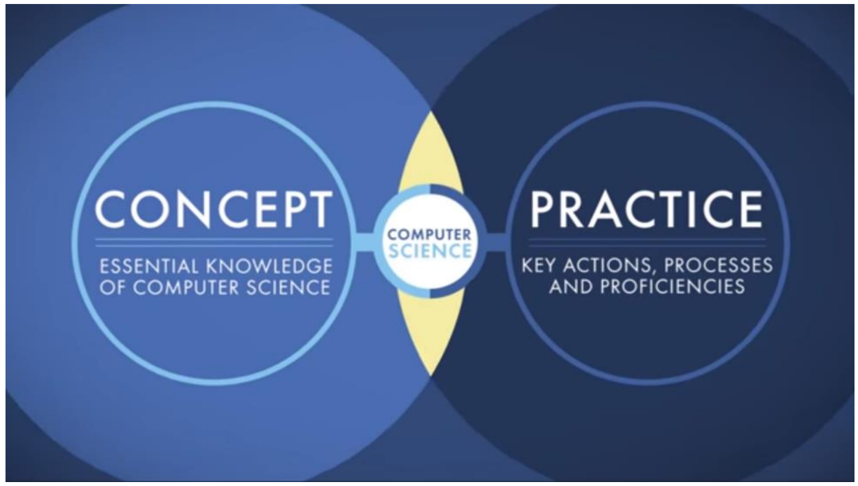

Figure 1: Relationship between Framework Concepts and Practices.
Graphic from K–12 Computer Science Framework video. YouTube: https://youtu.be/CD0EIGfr950

Computer science also has strong connections to other disciplines, and is becoming increasingly important in the workplace. Many problems in science, engineering, health care, business, and other areas can be solved effectively with computers, but finding a solution requires both computer science expertise and knowledge of the particular application domain. Thus, computer scientists need to understand and often become proficient in other subjects.

Goal of the Standards

The Computer Science K–12 Learning Standards are based on the Computer Science Teachers Association's K–12 Computer Science Standards, and define a set of standards that are supported by the K–12 Framework. The framework suggests steps that will be needed to enable their wide implementation. The standards introduce the principles and methodologies of computer science to all students, whether they are college bound or career bound after high school. The standards outlined in this document address the entire K–12 range. They complement existing K–12 computer science and information technology curricula where they are already established, especially the advanced placement (AP) computer science curricula (AP, 2010). Additionally, the standards complement existing curricula in other disciplines.

> *… the office of the superintendent of public instruction shall adopt computer science learning standards developed by a nationally recognized computer science education organization.*
>
> *-SHB 1813 (2015)*

Concepts in the Standards

The core concepts are categories that represent major content areas in the field of computer science.

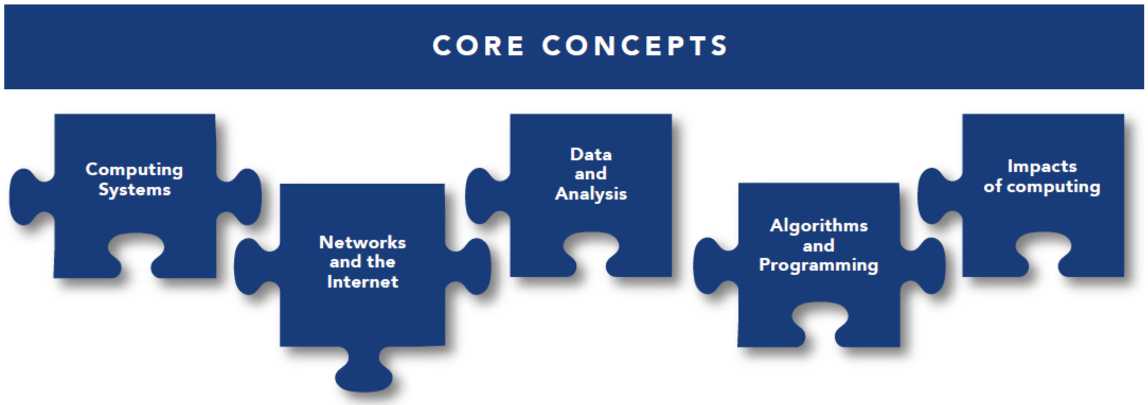

Figure 2: Concepts - K–12 Computer Science Framework. (2016). Retrieved from http://www.k12cs.org (CC BY NC SA 4.0).

Practices in the Standards

Practices are behaviors and ways of thinking that students will use as they learn and implement the various concepts described in the framework. For example, students will create computational artifacts to demonstrate and increase their knowledge of algorithms. Unlike the framework's concepts, the progressions of the practices are not delineated by grade bands.

Value of Concepts and Standards

The computer science concepts and practices will empower students to:

- Be informed citizens who can critically engage in public discussion on computer science related topics
- Develop as learners, users, and creators of computer science knowledge and artifacts
- Better understand the role of computing in the world around them
- Learn, perform, and express themselves in other subjects and interests
- Increase career and college readiness

The Computer Science Teachers Association collaborated with K12CS.org to align the development of Interim Computer Science Teachers Association K–12 Computer Science standards with the revision of the K–12 Computer Science Framework. This process intentionally paralleled the development of the Next Generation Science Standards, with a framework informing standards.

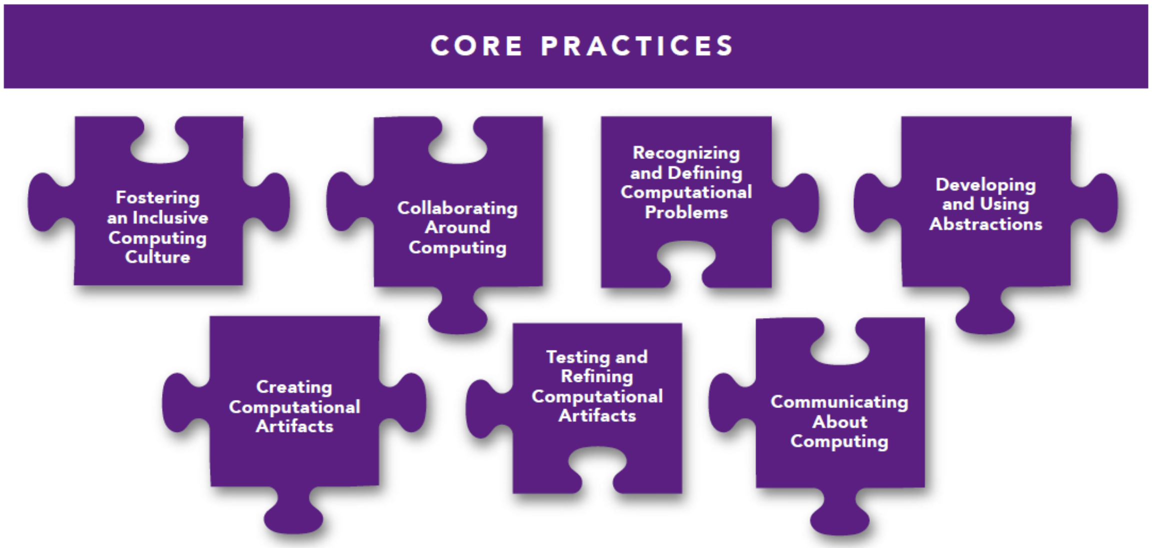

Figure 3: Practices- K–12 Computer Science Framework. (2016). Retrieved from http://www.k12cs.org (CC BY NC SA 4.0).

Computational Thinking

Computational Thinking, the human ability to formulate problems so that their solutions can be represented as computational steps or algorithms to be carried out by an information-processing agent (e.g., a computer), is central to the standards' practices and concepts.

Computational thinking is called out as an overarching practice reflected in a number of the core computer science practices. With its focus on abstraction, automation, and analysis, computational thinking is a core element of the broader discipline of computer science and for that reason it is interwoven through these computer science standards at all levels of K–12 learning.

Implementation of Grade-Level Bands

The grade-level bands associated with each learning standard are intended to provide teachers with the confidence to provide age-appropriate and accurate information and instruction that progresses in complexity from grade level to grade level. Competency for one grade level serves as a foundation for attaining competency of the bands for the next grade level. Teachers can use the grade-level bands as starting points for instruction and as checkpoints to ensure that the learning standards are taught and applied to the student's ability.

Teachers can use grade-level bands to:

- Develop lesson plans
- Establish specific and intentional learning objectives to guide teaching and learning
- Conduct ongoing formative and summative assessments to check student understanding and efficacy of instruction
- Integrate computational thinking into their curriculum
- Create an equitable environment

All curriculum in Washington is decided locally, within each district. Districts will determine how to incorporate the Computer Science K–12 Learning Standards into each grade level and integrate them into relevant high school courses leading to graduation.

An understanding of the fundamentals of computer science and its underlying problem-solving methodology of computational thinking is a valuable skill in our global economy. Not every student should become a computer scientist, but all students should have the opportunity to explore and create with computing. Learning standards are the foundation for what students should know and be able to do. How this learning occurs is up to districts to develop and teachers to impart every day in every classroom.

Organizations and Key Documents Referenced

A Framework for K–12 Computer Science Education, https://k12cs.org.

College Board's Computer Science A: https://apstudent.collegeboard.org/apcourse/ap-computer-science-a .

College Board's Computer Science Principles, https://apstudent.collegeboard.org/apcourse/ap-computer-science-principles.

Computer Science Teachers Association K–12 Computer Science Standards (2017), https://www.csteachers.org/page/standards.

Employment Projections (Washington Employment Security Department) https://fortress.wa.gov/esd/employmentdata/reports-publications/industry-reports/employment-projections.

Guzdial, M. (2015) Learner-centered design of computing education: Research on computing for everyone. Synthesis Lectures on Human-Centered Informatics, 8(6):1–165.

ISTE Standards, http://www.iste.org/standards.

K–12 Computer Science Framework (Video), https://www.youtube.com/watch?v=CD0EIGfr950.

Krauss, J., & Prottsman, K. (2016). Computational thinking and coding for every student: The teacher's getting-started guide. Thousand Oaks, CA: Corwin Press.

Legend for Identifiers

Unique Numbering System for the Washington Computer Science K–12 Learning Standards

To help organize and track each individual standard, a unique identifier was developed. An example appears below:

Level	Framework Concept	Number	Computer Science K–12 Learning Standard
Grades 6-8	Algorithms and Programming	17	Systematically test and refine programs using a range of test cases.
2	**AP**	**17**	**Identifier: 2-AP-17**

Use the following legend to interpret the unique identifier for each Computer Science K–12 Learning Standard:

The identifier code corresponds to: **Level – Concept – Number**		
Identifier Code		**Key**
Levels	1A	Grades K–2
	1B	Grades 3–5
	2	Grades 6–8
	3A	Grades 9–10
	3B	Grades 11–12
Concepts	CS	Computing Systems
	NI	Networks and the Internet
	DA	Data and Analysis
	AP	Algorithms and Programming
	IC	Impacts of Computing

Integrated into classroom activities through practices:

Practices	1	Fostering an Inclusive Computing Culture
	2	Collaborating
	3	Recognizing and Defining Computational Problems
	4	Developing and Using Abstractions
	5	Creating Computational Artifacts
	6	Testing and Refining
	7	Communicating about Computing

Figure 4: Standards Identifier Code –
Computer Science Teachers Association K–12 Computer Science Standards (2017)
Retrieved from http://www.csteachers.org

Standards

Identifier	Level 1A: K–2
1A-CS-01	Select and operate appropriate software to perform a variety of tasks, and recognize that users have different needs and preferences for the technology they use. (P 1.1)
1A-CS-02	Use appropriate terminology in identifying and describing the function of common physical components of computing systems (hardware). (P 7.2)
1A-CS-03	Describe basic hardware and software problems using accurate terminology. (P 6.2, P 7.2)
1A-NI-04	Explain what passwords are and why we use them, and use strong passwords to protect devices and information from unauthorized access. (P 7.3)
1A-DA-05	Store, copy, search, retrieve, modify, and delete information using a computing device and define the information stored as data. (P 4.2)
1A-DA-06	Collect and present the same data in various visual formats. (P 7.1, P 4.4)
1A-DA-07	Identify and describe patterns in data visualizations, such as charts or graphs, to make predictions. (P 4.1)
1A-AP-08	Model daily processes by creating and following algorithms (sets of step-by-step instructions) to complete tasks. (P. 4.4)
1A-AP-09	Model the way programs store and manipulate data by using numbers or other symbols to represent information. (P 4.4)
1A-AP-10	Develop programs with sequences and simple loops, to express ideas or address a problem. (P. 5.2)
1A-AP-11	Decompose (break down) the steps needed to solve a problem into a precise sequence of instructions. (P. 3.2)
1A-AP-12	Develop plans that describe a program's sequence of events, goals, and expected outcomes. (P. 5.1, P. 7.2)
1A-AP-13	Give attribution when using the ideas and creations of others while developing programs. (P. 7.3)
1A-AP-14	Debug (identify and fix) errors in an algorithm or program that includes sequences and simple loops. (P. 6.2)
1A-AP-15	Using correct terminology, describe steps taken and choices made during the iterative process of program development. (P. 7.2)
1A-IC-16	Compare how people live and work before and after the implementation or adoption of new computing technology. (P. 7)
1A-IC-17	Work respectfully and responsibly with others online. (P. 2.1)
1A-IC-18	Keep login information private, and log off of devices appropriately. (P. 7.3)

Identifier	Level 1B: 3–5
1B-CS-01	Describe how internal and external parts of computing devices function to form a system. (P. 7.2)
1B-CS-02	Model how computer hardware and software work together as a system to accomplish tasks. (P. 4.4)
1B-CS-03	Determine potential solutions to solve simple hardware and software problems using common troubleshooting strategies. (P. 6.2)
1B-NI-04	Model how information is broken down into smaller pieces, transmitted as packets through multiple devices over networks and the Internet, and reassembled at the destination. (P. 4.4)
1B-NI-05	Discuss real-world cybersecurity problems and how personal information can be protected. (P. 3.1)
1B-DA-06	Organize and present collected data visually to highlight relationships and support a claim. (P. 7.1)
1B-DA-07	Use data to highlight or propose cause-and-effect relationships, predict outcomes, or communicate an idea. (P. 7.1)
1B-AP-08	Compare and refine multiple algorithms for the same task and determine which is the most appropriate. (P. 6.3, P. 3.3)
1B-AP-09	Create programs that use variables to store and modify data. Variables are used to store and modify data. (P. 5.2)
1B-AP-10	Create programs that include sequences, events, loops, and conditionals. (P. 5.2)
1B-AP-11	Decompose (break down) problems into smaller, manageable subproblems to facilitate the program development process. (P. 3.2)
1B-AP-12	Modify, remix, or incorporate portions of an existing program into one's own work, to develop something new or add more advanced features. (P. 5.3)
1B-AP-13	Use an iterative process to plan the development of a program by including others' perspectives and considering user preferences. (P. 1.1, P. 5.1)
1B-AP-14	Observe intellectual property rights and give appropriate attribution when creating or remixing programs. (P. 5.2, P. 7.3)
1B-AP-15	Test and debug (identify and fix errors) a program or algorithm to ensure it runs as intended. (P. 6.1, P. 6.2)
1B-AP-16	Take on varying roles, with teacher guidance, when collaborating with peers during the design, implementation, and review stages of program development. (P. 2.2)
1B-AP-17	Describe choices made during program development using code comments, presentations, and demonstrations. (P. 7.2)
1B-IC-18	Discuss computing technologies that have changed the world, and express how those technologies influence, and are influenced by, cultural practices. (P. 3.1)
1B-IC-19	Brainstorm ways to improve the accessibility and usability of technology products for the diverse needs and wants of users. (P. 1.2)
1B-IC-20	Seek diverse perspectives for the purpose of improving computational artifacts. (P. 1.1)
1B-IC-21	Use public domain or creative commons media, and refrain from copying or using material created by others without permission. (P. 7.3)

Identifier	Level 2: 6–8
2-CS-01	Recommend improvements to the design of computing devices, based on an analysis of how users interact with the devices. (P. 3.3)
2-CS-02	Design projects that combine hardware and software components to collect and exchange data. (P. 5.1)
2-CS-03	Systematically identify and fix problems with computing devices and their components. (P. 6.2)
2-NI-04	Model the role of protocols in transmitting data across networks and the Internet. (P. 4.4)
2-NI-05	Explain how physical and digital security measures protect electronic information. (P. 7.2)
2-NI-06	Apply multiple methods of encryption to model the secure transmission of information. (P. 4.4)
2-DA-07	Represent data using multiple encoding schemes. (P. 4)
2-DA-08	Collect data using computational tools and transform the data to make it more useful and reliable. (P. 6.3)
2-DA-09	Refine computational models based on the data they have generated. (P. 5.3, P. 4.4)
2-AP-10	Use flowcharts and/or pseudocode to address complex problems as algorithms. (P. 4.4, 4.1)
2-AP-11	Create clearly named variables that represent different data types and perform operations on their values. (P. 5.1, P. 5.2)
2-AP-12	Design and iteratively develop programs that combine control structures, including nested loops and compound conditionals. (P. 5.1, P. 5.2)
2-AP-13	Decompose problems and subproblems into parts to facilitate the design, implementation, and review of programs. (P. 3.2)
2-AP-14	Create procedures with parameters to organize code and make it easier to reuse. (P. 4.1, P. 4.3)
2-AP-15	Seek and incorporate feedback from team members and users to refine a solution that meets user needs. (P. 2.3, P. 1.1)
2-AP-16	Incorporate existing code, media, and libraries into original programs, and give attribution. (P. 4.2, P. 5.2, P. 7.3)
2-AP-17	Systematically test and refine programs using a range of test cases. (P. 6.1)
2-AP-18	Distribute tasks and maintain a project timeline when collaboratively developing computational artifacts. (P. 2.2)
2-AP-19	Document programs in order to make them easier to follow, test, and debug. (7.2)
2-IC-20	Compare tradeoffs associated with computing technologies that affect people's everyday activities and career options. (P. 7.2)
2-IC-21	Discuss issues of bias and accessibility in the design of existing technologies. (1.2)
2-IC-22	Collaborate with many contributors through strategies such as crowdsourcing or surveys when creating a computational artifact. (P. 2.4, P. 5.2)
2-IC-23	Describe tradeoffs between allowing information to be public and keeping information private and secure. (P. 7.2)

Identifier	Level 3A: 9–10
3A-CS-01	Explain how abstractions hide the underlying implementation details of computing systems embedded in everyday objects. (P. 4.1)
3A-CS-02	Compare levels of abstraction and interactions between application software, system software, and hardware layers. (P. 4.1)
3A-CS-03	Develop guidelines that convey systematic troubleshooting strategies that others can use to identify and fix errors. (P. 6.2)
3A-NI-04	Evaluate the scalability and reliability of networks, by describing the relationship between routers, switches, servers, topology, and addressing. (P. 4.1)
3A-NI-05	Give examples to illustrate how sensitive data can be affected by malware and other attacks. (P. 7.2)
3A-NI-06	Recommend security measures to address various scenarios based on factors such as efficiency, feasibility, and ethical impacts. (P. 3.3)
3A-NI-07	Compare various security measures, considering tradeoffs between the usability and security of a computing system. (6.3)
3A-NI-08	Explain tradeoffs when selecting and implementing cybersecurity recommendations. (P. 7.2)
3A-DA-09	Translate between different bit representations of real-world phenomena, such as characters, numbers, and images. (P. 4.1)
3A-DA-10	Evaluate the tradeoffs in how data elements are organized and where data is stored. (P. 3.3)
3A-DA-11	Create interactive data visualizations using software tools to help others better understand real-world phenomena. (P. 4.4)
3A-DA-12	Create computational models that represent the relationships among different elements of data collected from a phenomenon or process. (P. 4.4)
3A-AP-13	Create prototypes that use algorithms to solve computational problems by leveraging prior student knowledge and personal interests. (P. 5.2)
3A-AP-14	Use lists to simplify solutions, generalizing computational problems instead of repeatedly using simple variables. (P. 4.1)
3A-AP-15	Justify the selection of specific control structures when tradeoffs involve implementation, readability, and program performance, and explain the benefits and drawbacks of choices made. (P. 5.2)
3A-AP-16	Design and iteratively develop computational artifacts for practical intent, personal expression, or to address a societal issue by using events to initiate instructions. (P. 5.2)
3A-AP-17	Decompose problems into smaller components through systematic analysis, using constructs such as procedures, modules, and/or objects. (P. 3.2)
3A-AP-18	Create artifacts by using procedures within a program, combinations of data and procedures, or independent but interrelated programs. (P. 5.2)
3A-AP-19	Systematically design and develop programs for broad audiences by incorporating feedback from users. (P. 5.1)
3A-AP-20	Evaluate licenses that limit or restrict use of computational artifacts when using resources such as libraries. (P. 7.3)
3A-AP-21	Evaluate and refine computational artifacts to make them more usable and accessible. (P. 6.3)

Identifier	Level 3A: 9–10
3A-AP-22	Design and develop computational artifacts working in team roles using collaborative tools. (P. 2.4)
3A-AP-23	Document –esign decisions using text, graphics, presentations, and/or demonstrations in the development of complex programs. (P. 7.2)
3A-IC-24	Evaluate the ways computing impacts personal, ethical, social, economic, and cultural practices. (P. 1.2)
3A-IC-25	Test and refine computational artifacts to reduce bias and equity deficits. (P. 1.2)
3A-IC-26	Demonstrate ways a given algorithm applies to problems across disciplines. (P. 3.1)
3A-IC-27	Use tools and methods for collaboration on a project to increase connectivity of people in different cultures and career fields. (P. 2.4)
3A-IC-28	Explain the beneficial and harmful effects that intellectual property laws can have on innovation. (P. 7.3)
3A-IC-29	Explain the privacy concerns related to the collection and generation of data through automated processes that may not be evident to users. (P. 7.2)
3A-IC-30	Evaluate the social and economic implications of privacy in the context of safety, law, or ethics. (P. 7.3)

Identifier	Level 3B: 11–12
3B-CS-01	Categorize the roles of operating system software. (P. 7.2)
3B-CS-02	Illustrate ways computing systems implement logic, input, and output through hardware components. (P. 7.2)
3B-NI-03	Describe the issues that impact network functionality (e.g., bandwidth, load, delay, topology). (P. 7.2)
3B-NI-04	Compare ways software developers protect devices and information from unauthorized access. (P. 7.2)
3B-DA-05	Use data analysis tools and techniques to identify patterns in data representing complex systems. (P. 4.1)
3B-DA-06	Select data collection tools and techniques to generate data sets that support a claim or communicate information. (P. 7.2)
3B-DA-07	Evaluate the ability of models and simulations to test and support the refinement of hypotheses. (P. 4.4)
3B-AP-08	Describe how artificial intelligence drives many software and physical systems. (P. 7.2)
3B-AP-09	Implement an artificial intelligence algorithm to play a game against a human opponent or solve a problem. (P. 5.3)
3B-AP-10	Use and adapt classic algorithms to solve computational problems. (P. 4.2)
3B-AP-11	Evaluate algorithms in terms of their efficiency, correctness, and clarity. (P. 4.2)
3B-AP-12	Compare and contrast fundamental data structures and their uses. (P. 4.2)
3B-AP-13	Illustrate the flow of execution of a recursive algorithm. (P. 3.2)
3B-AP-14	Construct solutions to problems using student-created components, such as procedures, modules and/or objects. (P. 5.2)
3B-AP-15	Analyze a large-scale computational problem and identify generalizable patterns that can be applied to a solution. (P. 4.1)
3B-AP-16	Demonstrate code reuse by creating programming solutions using libraries and APIs. (P. 5.3)
3B-AP-17	Plan and develop programs for broad audiences using a software lifecycle process. (P. 5.1)
3B-AP-18	Explain security issues that might lead to compromised computer programs. (P. 7.2)
3B-AP-19	Develop programs for multiple computing platforms. (P. 5.2)
3B-AP-20	Use version control systems, integrated development environments (IDEs), and collaborative tools and practices (code documentation) in a group software project. (P. 2.4)

Identifier	Level 3B: 11–12
3B-AP-21	Develop and use a series of test cases to verify that a program performs according to its design specifications. (P. 6.1)
3B-AP-22	Modify an existing program to add additional functionality and discuss intended and unintended implications (e.g., breaking other functionality). (P. 5.3)
3B-AP-23	Evaluate key qualities of a program through a process such as a code review. (P. 6.3)
3B-AP-24	Compare multiple programming languages and discuss how their features make them suitable for solving different types of problems. (P. 7.2)
3B-IC-25	Evaluate computational artifacts to maximize their beneficial effects and minimize harmful effects on society. (P. 6.1, P. 1.2)
3B-IC-26	Evaluate the impact of equity, access, and influence on the distribution of computing resources in a global society. (P. 1.2)
3B-IC-27	Predict how computational innovations that have revolutionized aspects of our culture might evolve. (P. 7.2)
3B-IC-28	Debate laws and regulations that impact the development and use of software. (P. 3.3, P. 7.3)

Computer Science Glossary

The following glossary includes definitions of terms used in the statements in the Washington Computer Science K–12 Learning Standards. These terms are intended to increase teacher understanding and decrease biased language.

abstraction *(process):* The process of reducing complexity by focusing on the main idea. By hiding details irrelevant to the question at hand and bringing together related and useful details, abstraction reduces complexity and allows one to focus on the problem. In elementary classrooms, abstraction is hiding unnecessary details to make it easier to think about a problem.

(product): A new representation of a thing, a system, or a problem that helpfully reframes a problem by hiding details irrelevant to the question at hand. [MA-DLCS]

(Code.org K–5) Pulling out specific differences to make one solution work for multiple problems.

algorithm: A step-by-step process to complete a task.

A list of steps to finish a task. A set of instructions that can be performed with or without a computer. For example, the collection of steps to make a peanut butter and jelly sandwich is an algorithm. (Code.org K–5)

app: A type of application software designed to run on a mobile device, such as a smartphone or tablet computer (also known as a mobile application). [Techopedia]

artifact: Anything created by a human. *See "computational artifact" for the computer science-specific definition.*

ASCII: (American Standard Code for Information Interchange) is the most common format for text files in computers and on the Internet. In an ASCII file, each alphabetic, numeric, or special character is represented with a 7-bit binary number (a string of seven 0s or 1s). 128 possible characters are defined.

automation: To link disparate systems and software in such a way that they become self-acting or self-regulating.

backup: The process of making copies of data or data files to use in the event the original data or data files are lost or destroyed. [Techopedia]

binary: A method of encoding data using two symbols (usually 1 and 0). To illustrate binary encoding, we can use any two symbols. [MA-DLCS]

A way of representing information using only two options. (Code.org K–5)

Block-based programming language: (Code.org K–5) Any programming language that lets users create programs by manipulating "blocks" or graphical programing elements, rather than writing code using text. Examples include Code Studio, Scratch, and Swift. (Sometimes called visual coding, drag and drop programming, or graphical programming blocks)

bug: An error in a software program. It may cause a program to unexpectedly quit or behave in an unintended manner. [TechTerms] The process of removing errors (bugs) is called debugging.

An error in a program that prevents the program from running as expected. (Code.org K–5)

cloud: Remote servers that store data and are accessed from the Internet. [Techopedia]

code: Any set of instructions expressed in a programming language. [MA-DLCS] One or more commands or algorithm(s) designed to be carried out by a computer. (Code.org K–5) See also: program

command: An instruction for the computer. Many commands put together make up algorithms and computer programs. (Code.org K–5)

computational artifact: Anything created by a human using a computational thinking process and a computing device. A computational artifact can be, but is not limited to, a program, image, audio, video, presentation, or web page file.

computational thinking: Mental processes and strategies that include: decomposition, pattern matching, abstraction, algorithms (decomposing problems into smaller, more manageable problems, finding repeating patterns, abstracting specific differences to make one solution work for multiple problems, and creating step-by-step algorithms). (Code.org K–5)

computer science: Using the power of computers to solve problems. (Code.org K–5)

conditionals: Statements that only run under certain conditions or situations. (Code.org K–5)

data: Information. Often, quantities, characters, or symbols that are the inputs and outputs of computer programs. (Code.org K–5)

debugging: Finding and fixing errors in programs. (Code.org K–5)

decompose: Break a problem down into smaller pieces. (Code.org K–5)

decryption: The process of taking encoded or encrypted text or other data and converting it back into text that you or the computer can read and understand.

Digital divide: the gulf between those who have ready access to computers and the Internet, and those who do not.

encryption: The process of encoding messages or information in such a way that only authorized parties can read it.

event: An action that causes something to happen. (Code.org K–5)

execution: The process of executing an instruction or instruction set.

for loop: A loop with a predetermined beginning, end, and increment (step interval) (Code.org K–5)

function: A type of procedure or routine. Some programming languages make a distinction between a function, which returns a value, and a procedure, which performs some operation, but does not return a value. [MA-DLCS] *Note: This definition differs from that used in math.* A piece of code that you can easily call over and over again. Functions are sometimes called 'procedures.' (Code.org K–5)

GPS: Abbreviation for "Global Positioning System." GPS is a satellite navigation system used to determine the ground position of an object. [TechTerms]

hacking: Appropriately applying ingenuity (from "The Meaning of Hack"), cleverly solving a programming problem (the New Hacker's Dictionary), and using a computer to gain unauthorized access to data within a system. [MA-DLCS]

hardware: The physical components that make up a computing system, computer, or computing device. [MA-DLCS]

hierarchy: An organizational structure in which items are ranked according to levels of importance. [TechTarget]

HTTP: (Hypertext Transfer Protocol) is the set of rules for transferring files (text, graphic images, sound, video, and other multimedia files) on the World Wide Web.

HTTPS: encrypts and decrypts user page requests as well as the pages that are returned by the Web server. The use of HTTPS protects against eavesdropping and man-in-the-middle attacks.

input: The signals or instructions sent to a computer. [Techopedia]

Internet: The global collection of computer networks and their connections, all using shared protocols to communicate [CAS-Prim] A group of computers and servers that are connected to each other. (Code.org K–5)

iterative: Involving the repeating of a process with the aim of approaching a desired goal, target, or result. [MA-DLCS]

logic (Boolean): Boolean logic deals with the basic operations of truth values: AND, OR, NOT and combinations thereof. [FOLDOC]

loop: A programming structure that repeats a sequence of instructions as long as a specific condition is true. [TechTerms]

looping: Repetition, using a loop. The action of doing something over and over again. (Code.org K–5)

lossless: data compression without loss of information.

lossy: data compression in which unnecessary information is discarded.

memory: Temporary storage used by computing devices. [MA-DLCS]

model: A representation of (some part of) a problem or a system. (Modeling (v): the act of creating a model) [MA-DLCS] *Note: This definition differs from that used in science.*

network: A group of computing devices (personal computers, phones, servers, switches, routers, and so on) connected by cables or wireless media for the exchange of information and resources.

nested loop: A loop within a loop, an inner loop within the body of an outer one.

operating system: Software that communicates with the hardware and allows other programs to run. An operating system (or "OS") is comprised of system software, or the fundamental files a computer needs to boot up and function. Every desktop computer, tablet, and smartphone includes an operating system that provides basic functionality for the device. [TechTerms]

operation: An action, resulting from a single instruction, that changes the state of data. [Dictionary.com]

packets: Small chunks of information that have been carefully formed from larger chunks of information.

pair programming: A technique in which two developers (or students) team together and work on one computer. [TechTarget] The terms "driver" and "navigator" are often used for the two roles. In a classroom setting, teachers often specify that students switch roles frequently (or within a specific period of time).

paradigm (programming): A theory or a group of ideas about how something should be done, made, or thought about. A philosophical or theoretical framework of any kind. [Merriam-Webster] Common programming paradigms are object-oriented, functional, imperative, declarative, procedural, logic, and symbolic. [DC, Wikipedia]

parallelism: The simultaneous execution on multiple processors of different parts of a program.

parameter: A special kind of variable used in a procedure to refer to one of the pieces of data provided as input to the procedure. These pieces of data are called arguments. An ordered list of parameters is usually included in the definition of a subroutine so each time the subroutine is called, its arguments for that call can be assigned to the corresponding parameters. [MA-DLCS]

An extra piece of information that you pass to the function to customize it for a specific need. (Code.org)

pattern matching: Finding similarities between things. (Code.org K–5)

persistence: Trying again and again, even when something is very hard. (Code.org K–5)

piracy: The illegal copying, distribution, or use of software. [TechTarget]

procedure: An independent code module that fulfills some concrete task and is referenced within a larger body of source code. This kind of code item can also be called a function or a subroutine. The fundamental role of a procedure is to offer a single point of reference for some small goal or task that the developer or programmer can trigger by invoking the procedure itself. A procedure may also be referred to as a function, subroutine, routine, method or subprogram. [Techopedia]

processor: The hardware within a computer or device that executes a program. The CPU (central processing unit) is often referred to as the brain of a computer.

program; programming *(n)*: A set of instructions that the computer executes in order to achieve a particular objective. [MA-DLCS] **program** *(v)*: To produce a program by programming. An algorithm that has been coded into something that can be run by a machine. (Code.org K–5)

programming: The craft of analyzing problems and designing, writing, testing, and maintaining programs to solve them. [MA-DLCS] The art of creating a program. (Code.org K–5)

protocol: The special set of rules that end points in a telecommunication connection use when they communicate. Protocols specify interactions between the communicating entities. [TechTarget]

prototype; prototype: An early approximation of a final product or information system, often built for demonstration purposes. [TechTarget, Techopedia]

pseudocode: A detailed yet readable description of what a computer program or algorithm must do, expressed in a formally-styled natural language rather than in a programming language. [TechTarget]

RGB: (red, green, and blue) Refers to a system for representing the colors to be used on a computer display. Red, green, and blue can be combined in various proportions to obtain any color in the visible spectrum.

routing; router; routing: Establishing the path that data packets traverse from source to destination. A device or software that determines the routing for a data packet. [TechTarget]

run program: Cause the computer to execute the commands you've written in your program. (Code.org K–5)

security: The protection against access to, or alteration of, computing resources, through the use of technology, processes, and training. [TechTarget]

servers: Computers that exist only to provide things to others. (Code.org K–5)

simulate: to imitate the operation of a real world process or system over time.

simulation: Imitation of the operation of a real world process or system over time. [MA-DLCS]

software: Programs that run on a computer system, computer, or other computing device.

SMTP: the standard protocol for sending emails across the Internet. The communication between mail servers uses port 25.

IMAP: a mail protocol used for accessing email on a remote web server from a local client.

storage: A place (usually a device) into which data can be entered, in which it can be held, and from which it can be retrieved at a later time. [FOLDOC] A process through which digital data is saved within a data storage device by means of computing technology. Storage is a mechanism that enables a computer to retain data, either temporarily or permanently. [Techopedia]

string: A sequence of letters, numbers, and/or other symbols. A string might represent a name, address, or song title. Some functions commonly associated with strings are length, concatenation, and substring. [TechTarget]

structure: A general term used in the framework to discuss the concept of encapsulation without specifying a particular paradigm.

subroutine: A callable unit of code, a type of procedure.

switch: A high-speed device that receives incoming data packets and redirects them to their destination on a local area network (LAN). [Techopedia]

system: A collection of elements or components that work together for a common purpose. [TechTarget] A collection of computing hardware and software integrated for the purpose of accomplishing shared tasks.

topology: The physical and logical configuration of a network; the arrangement of a network, including its nodes and connecting links. A logical topology is how devices appear connected to the user. A physical topology is how they are actually interconnected with wires and cables. [PC Magazine]

troubleshooting: A systematic approach to problem solving that is often used to find and resolve a problem, error, or fault within software or a computer system. [Techopedia, TechTarget]

user: A person for whom a hardware or software product is designed (as distinguished from the developers). [TechTarget]

variable: A symbolic name that is used to keep track of a value that can change while a program is running. Variables are not just used for numbers. They can also hold text, including whole sentences ("strings"), or the logical values "true" or "false." A variable has a data type and is associated with a data storage location; its value is normally changed during the course of program execution. [CAS-Prim, Techopedia] A placeholder for a piece of information that can change (Code.org K–5) ***Note**: This definition differs from that used in math.*

wearable computing: Miniature electronic devices that are worn under, with or on top of clothing.

Key to sources of multiple definitions in this glossary:

CAS-Prim: Computing at School. Computing in the national curriculum: A guide for primary teachers (http://www.computingatschool.org.uk/data/uploads/CASPrimaryComputing.pdf)

Code.org: Creative Commons License (CC BY-NC-SA 4.0) (https://code.org/curriculum/docs/k-5/glossary)

Computer Science Teachers Association: CSTA K–12 Computer Science Standards (2011) https://csta.acm.org/Curriculum/sub/K12Standards.html

FOLDOC: Free On-Line Dictionary of Computing. (http://foldoc.org/)

MA-DLCS: Massachusetts Digital Literacy and Computer Science Standards, Glossary (Draft, December 2015)

NIST/DADS: National Institute of Science and Technology Dictionary of Algorithms and Data Structures. (https://xlinux.nist.gov/dads//)

Techopedia: Techopedia. (https://www.techopedia.com/dictionary)

TechTarget: TechTarget Network. (http://www.techtarget.com/network)

TechTerms: Tech Terms Computer Dictionary. (http://www.techterms.com)

FINAL BILL REPORT: SHB 1813

C 3 L 15 E1 Synopsis as Enacted

Brief Description: Expanding computer science education.

Sponsors: House Committee on Appropriations (originally sponsored by Representatives Hansen, Magendanz, Reykdal, Muri, Tarleton, Zeiger, Lytton, Haler, Senn, Harmsworth, Tharinger, Young, Walkinshaw, Stanford, S. Hunt and Pollet).

House Committee on Education
House Committee on Appropriations
Senate Committee on Early Learning & K–12 Education

Background:

Endorsements.

There are several pathways to endorsement and different types of endorsements. For example, academic endorsements and Career and Technical Education (CTE) endorsements differ—a CTE endorsed teacher may only teach CTE courses, and these courses often will not apply toward core education requirements. There is no academic endorsement for computer science, only a CTE endorsement, which teachers may obtain by demonstrating to a teacher preparation program that they have experience in the field and have met the program's requirements.

Conditional Scholarship for Educators.
A conditional scholarship is a loan that is forgiven in whole or in part in exchange for service as a certificated teacher at a K–12 public school. The state forgives one year of loan obligation for every two years a recipient teaches in a Washington K–12 public school. When a recipient fails to continue with the required course of study or teaching obligation, the recipient must repay the remaining loan principal with interest.

The Retooling Mathematics and Sciences Conditional Scholarship Program requires a K–12 teacher, or certificated elementary educator who is not employed in a position requiring an elementary education certificate, to pursue an endorsement in math or science to be eligible for the program. The conditional scholarship amount is determined by the Student Achievement Council, may not exceed $3,000 per year, and is applied to the cost of tuition, fees, and educational expenses.

This analysis was prepared by non-partisan legislative staff for the use of legislative members in their deliberations. This analysis is not a part of the legislation nor does it constitute a statement of legislative intent.

Summary:

Endorsements and Standards.

The OSPI and the Professional Educator Standard Board (PESB) must adopt computer science learning standards developed by a nationally recognized computer science education organization. The PESB must also develop standards for a K–12 computer science endorsement, which must facilitate dual endorsement in computer science and mathematics, science, or another related high-demand endorsement.

Conditional Scholarship for Educators.

The Retooling to Teach Mathematics and Sciences Conditional Scholarship Program is renamed the Educator Retooling Conditional Scholarship Program. A K–12 teacher, or certificated elementary educator who is not employed in a position requiring an elementary education certificate, may qualify for the conditional scholarship program by pursuing an endorsement in a subject or geographic endorsement shortage area, as defined by the Professional Educator Standards Board.

Votes on Final Passage:

House	91	7

First Special Session

House	88	4
Senate	43	0

Effective: August 27, 2015

Acknowledgements

Sincere appreciation is extended to the members of the Computer Science Leadership team for their time, expertise and commitment in the vetting of these standards.

Washington Computer Science Leadership Team

Gregory Bianchi, STEM Curriculum Developer, Bellevue School District
Jacob Blickenstaff, Program Director, Washington State LASER/Pacific Science Center
Georgia Boatman, Science Coordinator, Educational Service District 123
Callista Chen, Executive Director, Techbridge Girls
Clarence Dancer, STEM Program Supervisor, OSPI
Mark DeLoura, Games and Education Consultant, Satori
Barbara Dittrich, Advanced Placement Program Supervisor OSPI
Maya Donnelly, STEM Teacher, Pasco School District
Ellen Ebert, Science Director, OSPI
Perry Fizzano, Associate Professor Computer Science, Western Washington University
Dan Gallagher, Science Program Manager, Seattle Public Schools
Nimisha Ghosh Roy, District Manager, Code.org
Phyllis Harvey-Buschell, Curriculum Director, Washington MESA
Andrew Hickman, Digital Learning Coordinator, Educational Service District 113
Vickei Hrdina, Science Coordinator, Educational Service District 112
Derek Jaques, District CTE Director, Camas School District
Gregory Kilpatrick, Assistant CTE Director, South Kitsap School District
Thomas LaGuardia, Student, Kent Meridian High School Student
Mechelle LaLanne, Science Coordinator, Educational Service District 171
Juan Lozano, CTE Educational Specialist, Highline School District
Cheryl Lydon, Science Coordinator, Educational Service District 121
Stuart Reges, Principal Lecturer Computer Science and Engineering, University of Washington
Jana Sanchez, Elementary Instructional Facilitator, Mathematics, Everett School District
Tammie Schradar, Science Coordinator, Educational Service District 101
Andy Shouse, Chief Program Officer, Washington STEM
Dennis Small, Director Educational Technology, OSPI
Adam Smith, Computer Science Teacher, Cheney High School
Kathleen Stillwell, K - 12 Curriculum Specialist, Mathematics, Computer Science, Everett School District
Shannon Thissen, Computer Science Program Specialist, OSPI
Gilda Wheeler, Senior Program Officer, Washington STEM
David Wicks, Associate Professor of Curriculum & Instruction, Seattle Pacific University
Ann Wright-Mockler, STEM Educator-in-Residence, Pacific Northwest National Laboratory
Lance Wrzesinski, Business & Marketing Pathway Supervisor, OSPI

Washington Computer Science Learning Standards Advisory Committee

Greg Bianchi, STEM Curriculum Developer, Bellevue School District
Cheri Bortleson, K-5 STEM Curriculum Developer – Bellevue School District
Mark DeLoura, Games and Education Consultant, Satori
Ellen Ebert, Science Director, OSPI
Perry Fizzano, Computer Science Professor, Western Washington University
Anne Gallagher, Mathematics Director, OSPI
Nimisha Ghosh-Roy, District Manager, Code.org
Phyllis Harvey-Buschel, Curriculum Director, Washington MESA
Greg Kilpatrick, Assistant CTE Director, South Kitsap School District
Thomas LaGuardia, Student, Kent Meridian High School Student
Stuart Reges, Principal Lecturer Computer Science and Engineering, University of Washington
Tammie Schrader, Science Coordinator, Educational Service District101
Adam Smith, Computer Science Teacher, Cheney School District
Kathy Stillwell, STEM Innovation Specialist, Everett School District
Shannon Thissen, Computer Science Program Specialist, OSPI
Lance Wrzesinski, Business & Marketing Pathway Supervisor, OSPI

Teacher Standards Review Team

Ray Birks, Instructional Technology Facilitator, Wenatchee School District
Lori Boyd, Computer Lab Director, Moses Lake School District
Michael Conklin, Math/Computer Science Teacher, Central Valley School District
Lori Curtis, Technology Specialist, White River School District
Anna Davis, 3rd Grade Teacher, Moses Lake School District
Maya Donnelly, STEM Elementary Teacher, Pasco School District
Liz Ebersole, Teacher/Librarian, Private School
Connie Haines, Speech Language Pathologist/Assistive Technology, Sumner School District
Ann Hayes-Bell, Technology Integration Specialist, Shoreline School District
Dane Lewman, Business Teacher, Cascade School District
Marc Long, Computer Science Teacher, Kennewick School District
Juan Lozano, CTE Educational Specialist, Highline School District
Tina Nicpan-Brown, 5th grade Teacher, Wenatchee School District
Patricia Percival, Middle School Math Teacher, Everett School District
Wendy Richmond, Elementary Teacher, Richland School District
Adam Smith, Computer Science Teacher, Cheney School District
Dani Ward, STEM Teacher, Bellevue School District

Bias & Sensitivity Review

Clarence Dancer, STEM Program Supervisor, OSPI
Mary Dismuke, Native American Education Coordinator, Clover Park School District
Gwendolyn Haley, Library Services Manager, Spokane City Libraries
Skylar Jones, Sped Education High Needs, Medical Lake School District
Sherry Krainick, WSPTA Legislative Director
Catherine Lee, Webmaster, Chinese American Citizens Alliance
Mynor Lopez, Executive Assistant, WA Commission on Hispanic Affairs
Ann Renker, Assistant Superintendent, Sequim School District
Laurel White, Speech Language Pathologist, Eastmont School District

Chris Reykdal • State Superintendent
Office of Superintendent of Public Instruction
Old Capitol Building • P.O. Box 47200
Olympia, WA 98504-7200