washington
STATE LEARNING STANDARDS

kindergarten – 12th grade

Computer Science

K-12 Standards Algorithms and Programming

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.

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

K-12 Algorithms and Programming Standards

Identifier	Level 1A: K–2
1A-AP-08	Model daily processes by creating and following algorithms (sets of step-by-step instructions) to complete tasks.
1A-AP-09	Model the way programs store and manipulate data by using numbers or other symbols to represent information.
1A-AP-10	Develop programs with sequences and simple loops, to express ideas or address a problem.
1A-AP-11	Decompose (break down) the steps needed to solve a problem into a precise sequence of instructions.
1A-AP-12	Develop plans that describe a program's sequence of events, goals, and expected outcomes.
1A-AP-13	Give attribution when using the ideas and creations of others while developing programs.
1A-AP-14	Debug (identify and fix) errors in an algorithm or program that includes sequences and simple loops.
1A-AP-15	Using correct terminology, describe steps taken and choices made during the iterative process of program development.
Identifier	**Level 1B: 3–5**
1B-AP-08	Compare and refine multiple algorithms for the same task and determine which is the most appropriate.
1B-AP-09	Create programs that use variables to store and modify data. Variables are used to store and modify data.
1B-AP-10	Create programs that include sequences, events, loops, and conditionals.
1B-AP-11	Decompose (break down) problems into smaller, manageable subproblems to facilitate the program development process.
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.
1B-AP-13	Use an iterative process to plan the development of a program by including others' perspectives and considering user preferences.
1B-AP-14	Observe intellectual property rights and give appropriate attribution when creating or remixing programs.
1B-AP-15	Test and debug (identify and fix errors) a program or algorithm to ensure it runs as intended.
1B-AP-16	Take on varying roles, with teacher guidance, when collaborating with peers during the design, implementation, and review stages of program development.
1B-AP-17	Describe choices made during program development using code comments, presentations, and demonstrations.

Identifier	Level 2: 6–8
2-AP-10	Use flowcharts and/or pseudocode to address complex problems as algorithms.
2-AP-11	Create clearly named variables that represent different data types and perform operations on their values.
2-AP-12	Design and iteratively develop programs that combine control structures, including nested loops and compound conditionals.
2-AP-13	Decompose problems and subproblems into parts to facilitate the design, implementation, and review of programs.
2-AP-14	Create procedures with parameters to organize code and make it easier to reuse.
2-AP-15	Seek and incorporate feedback from team members and users to refine a solution that meets user needs.
2-AP-16	Incorporate existing code, media, and libraries into original programs, and give attribution.
2-AP-17	Systematically test and refine programs using a range of test cases.
2-AP-18	Distribute tasks and maintain a project timeline when collaboratively developing computational artifacts.
2-AP-19	Document programs in order to make them easier to follow, test, and debug.
Identifier	**Level 3A: 9–10**
3A-AP-13	Create prototypes that use algorithms to solve computational problems by leveraging prior student knowledge and personal interests.
3A-AP-14	Use lists to simplify solutions, generalizing computational problems instead of repeatedly using simple variables.
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.
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.
3A-AP-17	Decompose problems into smaller components through systematic analysis, using constructs such as procedures, modules, and/or objects.
3A-AP-18	Create artifacts by using procedures within a program, combinations of data and procedures, or independent but interrelated programs.
3A-AP-19	Systematically design and develop programs for broad audiences by incorporating feedback from users.
3A-AP-20	Evaluate licenses that limit or restrict use of computational artifacts when using resources such as libraries.
3A-AP-21	Evaluate and refine computational artifacts to make them more usable and accessible.
3A-AP-22	Design and develop computational artifacts working in team roles using collaborative tools.
3A-AP-23	Document –design decisions using text, graphics, presentations, and/or demonstrations in the development of complex programs.

Identifier	Level 3B: 11–12
3B-AP-08	Describe how artificial intelligence drives many software and physical systems.
3B-AP-09	Implement an artificial intelligence algorithm to play a game against a human opponent or solve a problem.
3B-AP-10	Use and adapt classic algorithms to solve computational problems.
3B-AP-11	Evaluate algorithms in terms of their efficiency, correctness, and clarity.
3B-AP-12	Compare and contrast fundamental data structures and their uses.
3B-AP-13	Illustrate the flow of execution of a recursive algorithm.
3B-AP-14	Construct solutions to problems using student-created components, such as procedures, modules and/or objects.
3B-AP-15	Analyze a large-scale computational problem and identify generalizable patterns that can be applied to a solution.
3B-AP-16	Demonstrate code reuse by creating programming solutions using libraries and APIs.
3B-AP-17	Plan and develop programs for broad audiences using a software lifecycle process.
3B-AP-18	Explain security issues that might lead to compromised computer programs.
3B-AP-19	Develop programs for multiple computing platforms.
3B-AP-20	Use version control systems, integrated development environments (IDEs), and collaborative tools and practices (code documentation) in a group software project.
3B-AP-21	Develop and use a series of test cases to verify that a program performs according to its design specifications.
3B-AP-22	Modify an existing program to add additional functionality and discuss intended and unintended implications (e.g., breaking other functionality).
3B-AP-23	Evaluate key qualities of a program through a process such as a code review.
3B-AP-24	Compare multiple programming languages and discuss how their features make them suitable for solving different types of problems.

Download this material in PDF at http://www.k12.wa.us/CurriculumInstruct/learningstandards.aspx.

Please refer to this document number for quicker service: 16-0075.

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