Arizona Computer Science Standards

Appendix A-Computer Science Glossary

ARIZONA DEPARTMENT OF EDUCATION
HIGH ACADEMIC STANDARDS FOR STUDENTS
October 2018

Appendix A-Computer Science Glossary

The following glossary includes definitions of commonly-used computer science terms and was borrowed (with permission) from the K–12 Computer Science Framework. This section is intended to increase teacher understanding and use of computer science terminology.

Abstraction: Pulling out specific difference to make one solution work for multiple problems.

- (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.

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.

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

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: The process of linking 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.

Binary: A system of notation representing data using two symbols (usually 1 and 0).

Block-based programming language: Any programming language that lets users create programs by manipulating "blocks" or graphical programming elements, rather than writing code using text. (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. The process of removing errors (bugs) is called debugging.

Cloud: Remote servers that store data and are accessed from the Internet.

Code: Any set of instructions expressed in a programming language. One or more commands or algorithm(s) designed to be carried out by a computer. See also: program

Command: An instruction for the computer. Many commands put together make up algorithms and computer programs.

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 models: Used to make predictions about processes or phenomenon based on selected data and features that can be represented by organizational software.

Computational thinking: Mental processes and strategies that include: decomposition (breaking larger problems into smaller, more manageable problems), pattern matching (finding repeating patterns), abstraction (identifying specific changes that would make one solution work for multiple problems), and algorithms (a step-by-step set of instructions that can be acted upon by a computer).

Computer science: The study of computers and algorithmic processes including their principles, hardware and software design, their applications, and their impact on society.

Conditionals: Statements that only run under certain conditions or situations.

Data: Information. Often, quantities, characters, or symbols that are the inputs and outputs of computer programs.

Debugging: Finding and fixing errors in programs.

Decompose: Break a problem down into smaller pieces.

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

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

For loop: A loop with a predetermined beginning, end, and increment (step interval)

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. 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.'

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

Hacking: Appropriately applying ingenuity. Cleverly solving a programming problem. Using a computer to gain unauthorized access to data within a system.

Hardware: The physical components that make up a computing system, computer, or computing device.

Hierarchy: An organizational structure in which items are ranked according to levels of importance.

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: A web transfer protocol that 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 attacks.

Input: The signals or instructions sent to a computer.

Internet: The global collection of computer networks and their connections, all using shared protocols to communicate. A group of computers and servers that are connected to each other.

Iterative: Involving the repeating of a process with the aim of approaching a desired goal, target, or result.

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

Loop: A programming structure that repeats a sequence of instructions as long as a specific condition is true.

Looping: Repetition, using a loop. The action of doing something over and over again.

Lossless: Data compression without loss of information.

Lossy: Data compression in which unnecessary information is discarded.

Memory: Temporary storage used by computing devices.

Model: A representation of (some part of) a problem or a system. (Modeling: the act of creating a model.) Note: This definition differs from that used in science.

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

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.

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.

Operation: An action, resulting from a single instruction that changes the state of data.

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. 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. Common programming paradigms are object-oriented, functional, imperative, declarative, procedural, logic, and symbolic.

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. An extra piece of information that you pass to a function to customize it for a specific need.

Pattern matching: Finding similarities between things.

Persistence: Trying again and again, even when something is very hard.

Piracy: The illegal copying, distribution, or use of software.

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.

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 (n): A set of instructions that the computer executes in order to achieve a particular objective. Program (v): To produce a program by programming. An algorithm that has been coded into something that can be run by a machine.

Programming (v): The craft of analyzing problems and designing, writing, testing, and maintaining programs to solve them. The art of creating a program.

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

Prototype: An early approximation of a final product or information system, often built for demonstration purposes.

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.

Remix: making changes to an existing procedure.

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.

Run program: Cause the computer to execute the commands written in a program.

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

Servers: Computers that exist only to provide things to others.

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.

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

SMTP: A standard protocol for sending emails across the Internet. The communication between mail servers, by default, 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. 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.

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.

Structure: 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).

System: A collection of elements or components that work together for a common purpose. 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 details how devices appear connected to the user. A physical topology is how devices are actually interconnected with wires and cables.

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.

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

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. A placeholder for a piece of information that can change. Note: This definition differs from that used in math.

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

Sources for 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)

Appendix B-Computer Science Practices

There are seven core practices of computer science. The practices naturally integrate with one another and contain language that intentionally overlaps to illuminate the connections among them. Unlike the core concepts, the practices are not delineated by grade bands. Conversely, like the core concepts, they are meant to build upon each other. (Adapted from: K-12 Computer Science Framework, 2016)

Practices-All practices list skills that students should be able to incorporate by the end of 12th grade
Practice 1. Fostering an Inclusive Computing Culture: Building an inclusive and diverse computing culture requires strategies for incorporating perspectives from people of different genders, ethnicities, and abilities. Incorporating these perspectives involves understanding the personal, ethical, social, economic, and cultural contexts in which people operate. Considering the needs of diverse users during the design process is essential to producing inclusive computational products.
1.1. Include the unique perspectives of others and reflect on one's own perspectives when designing and developing computational products.
1.2. Address the needs of diverse end users during the design process to produce artifacts with broad accessibility and usability.
1.3. Employ self- and peer-advocacy to address bias in interactions, product design, and development methods.
Practice 2. Collaborating Around Computing: Collaborative computing is the process of performing a computational task by working in pairs and on teams. Because it involves asking for the contributions and feedback of others, effective collaboration can lead to better outcomes than working independently. Collaboration requires individuals to navigate and incorporate diverse perspectives, conflicting ideas, disparate skills, and distinct personalities. Students should use collaborative tools to effectively work together and to create complex artifacts.
2.1. Cultivate working relationships with individuals possessing diverse perspectives, skills, and personalities.
2.2. Create team norms, expectations, and equitable workloads to increase efficiency and effectiveness.
2.3. Solicit and incorporate feedback from, and provide constructive feedback to, team members and other stakeholders.
2.4. Evaluate and select technological tools that can be used to collaborate on a project.

Practice 3. Recognizing and Defining Computational Problems: The ability to recognize appropriate and worthwhile opportunities to apply computation is a skill that develops over time and is central to computing. Solving a problem with a computational approach requires defining the problem, breaking it down into parts, and evaluating each part to determine whether a computational solution is appropriate.
3.1. Identify complex, interdisciplinary, real-world problems that can be solved computationally.
3.2. Decompose complex real-world problems into manageable subproblems that could integrate existing solutions or procedures.
3.3. Evaluate whether it is appropriate and feasible to solve a problem computationally.
Practice 4. Developing and Using Abstractions: Abstractions are formed by identifying patterns and extracting common features from specific examples to create generalizations. Using generalized solutions and parts of solutions designed for broad reuse simplifies the development process by managing complexity.
4.1. Extract common features from a set of interrelated processes or complex phenomena.
4.2. Evaluate existing technological functionalities and incorporate them into new designs.
4.3. Create modules and develop points of interaction that can apply to multiple situations and reduce complexity.
4.4. Model phenomena and processes and simulate systems to understand and evaluate potential outcomes.
Practice 5. Creating Computational Artifacts: The process of developing computational artifacts embraces both creative expression and the exploration of ideas to create prototypes and solve computational problems. Students create artifacts that are personally relevant or beneficial to their community and beyond. Computational artifacts can be created by combining and modifying existing artifacts or by developing new artifacts. Examples of computational artifacts include programs, simulations, visualizations, digital animations, robotic systems, and apps.
5.1. Plan the development of a computational artifact using an iterative process that includes reflection on and modification of the plan, taking into account key features, time and resource constraints, and user expectations.
5.2. Create a computational artifact for practical intent, personal expression, or to address a societal issue.
5.3. Modify an existing artifact to improve or customize it.
Practice 6. Testing and Refining Computational Artifacts: Testing and refinement is the deliberate and iterative process of improving a computational artifact. This process includes debugging (identifying and fixing errors) and comparing actual outcomes to intended outcomes. Students also respond to the changing needs and expectations of end users and improve the performance, reliability, usability, and accessibility of artifacts.

6.1. Systematically test computational artifacts by considering all scenarios and using test cases.
6.2. Identify and fix errors using a systematic process.
6.3. Evaluate and refine a computational artifact multiple times to enhance its performance, reliability, usability, and accessibility.
Practice 7. Communicating About Computing: Communication involves personal expression and exchanging ideas with others. In computer science, students communicate with diverse audiences about the use and effects of computation and the appropriateness of computational choices. Students write clear comments, document their work, and communicate their ideas through multiple forms of media. Clear communication includes using precise language and carefully considering possible audiences.
7.1. Select, organize, and interpret large data sets from multiple sources to support a claim.
7.2. Describe, justify, and document computational processes and solutions using appropriate terminology consistent with the intended audience and purpose.
7.3. Articulate ideas responsibly by observing intellectual property rights and giving appropriate attribution.