Arizona Computer Science Standards

Computer Science Practices

ARIZONA DEPARTMENT OF EDUCATION
HIGH ACADEMIC STANDARDS FOR STUDENTS
October 2018

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.