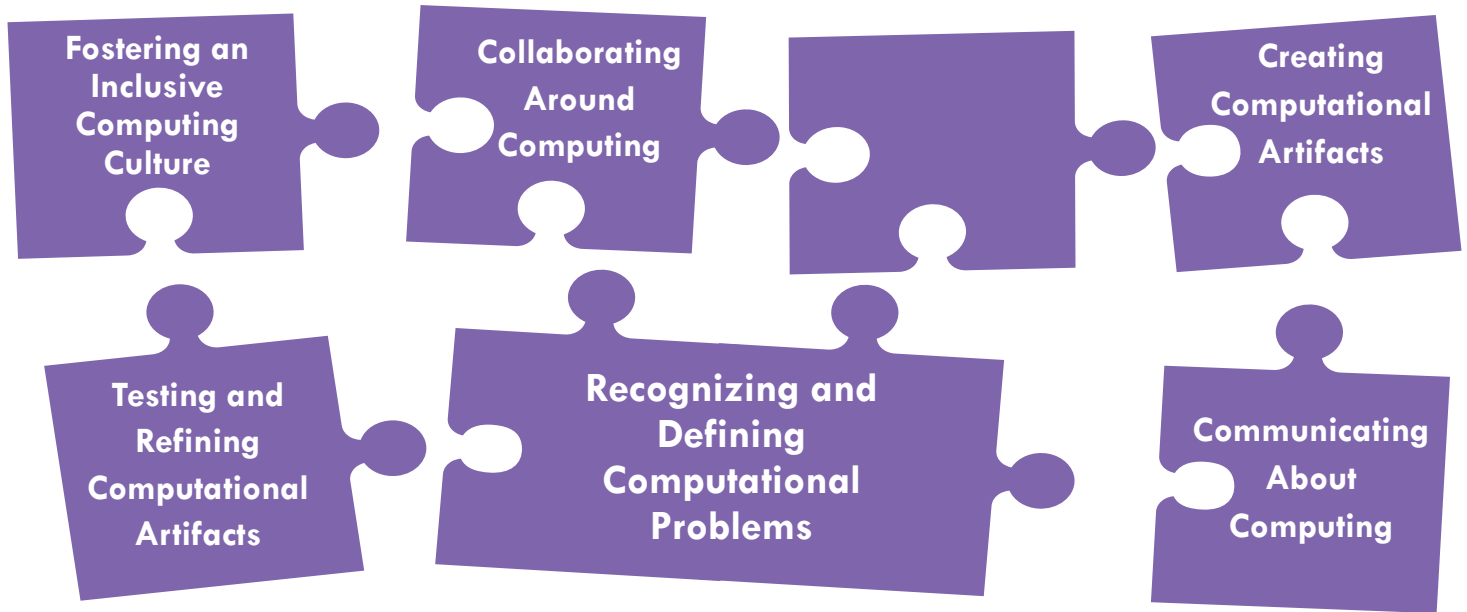


Practice Language⁶:

Core Practices



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.

By the end of Grade 5, students should be able to:

1. Include the unique perspectives of others and reflect on one's own perspectives when designing and developing computational products.

At all grade levels, students should recognize that the choices people make when they create artifacts are based on personal interests, experiences, and needs. Young learners should begin to differentiate their technology preferences from the technology preferences of others. Initially, students should be presented with perspectives from people with different backgrounds, ability levels, and points of view. As students progress, they should independently seek diverse perspectives throughout the design process for the purpose of improving their computational artifacts.

⁶ K-12 Computer Science Framework. (October 2016) Retrieved from:

<https://k12cs.org/wp-content/uploads/2016/09/K%E2%80%9312-Computer-Science-Framework.pdf>

2. Address the needs of diverse end users during the design process to produce artifacts with broad accessibility and usability.

At any level, students should recognize that users of technology have different needs and preferences and that not everyone chooses to use, or is able to use, the same technology products. For example, young learners, with teacher guidance, might compare a touchpad and a mouse to examine differences in usability. As students progress, they should consider the preferences of people who might use their products. Students should be able to evaluate the accessibility of a product to a broad group of end users, such as people with various disabilities. For example, they may notice that allowing an end user to change font sizes and colors will make an interface usable for people with low vision.

3. Employ self- and peer-advocacy to address bias in interactions, product design, and development methods.

After students have experience identifying diverse perspectives and including unique perspectives (P1.1), they should begin to employ self-advocacy strategies, such as speaking for themselves if their needs are not met.

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.

By the end of Grade 5, students should be able to:

1. Cultivate working relationships with individuals possessing diverse perspectives, skills, and personalities.

At any grade level, students should work collaboratively with others. Early on, they should learn strategies for working with team members who possess varying individual strengths. For example, with teacher support, students should begin to give each team member opportunities to contribute and to work with each other as co-learners. Eventually, students should become more sophisticated at applying strategies for mutual encouragement and support. They should express their ideas with logical reasoning and find ways to reconcile differences cooperatively. For example, when they disagree, they should ask others to explain their reasoning and work together to make respectful, mutual decisions.

2. Create team norms, expectations, and equitable workloads to increase efficiency and effectiveness.

After students have had experience cultivating working relationships within teams (P2.1), they should gain experience working in particular team roles. Early on, teachers may help guide this process by providing collaborative structures. For example, students may take turns in different roles on the project, such as note taker, facilitator, or “driver” of the computer. As students progress, they should become less dependent on the teacher assigning roles and become more adept at assigning roles within their teams. For example, they should decide together how to take turns in different roles.

3. Solicit and incorporate feedback from, and provide constructive feedback to, team members and other stakeholders.

At any level, students should ask questions of others and listen to their opinions. Early on, with teacher scaffolding, students should seek help and share ideas to achieve a particular purpose. As they progress in school, students should provide and receive feedback related to computing in constructive ways. For example, pair programming is a collaborative process that promotes giving and receiving feedback.

4. Evaluate and select technological tools that can be used to collaborate on a project.

At any level, students should be able to use tools and methods for collaboration on a project. For example, in the early grades, students could collaboratively brainstorm by writing on a whiteboard. As students progress, they should use technological collaboration tools to manage team work, such as knowledge-sharing tools and online project spaces. They should also begin to make decisions about which tools would be best to use and when to use them.

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.

By the end of Grade 5, students should be able to:

1. Identify complex, interdisciplinary, real-world problems that can be solved computationally.

At any level, students should be able to identify problems that have been solved computationally. For example, young students can discuss a technology that has changed the world, such as email or mobile phones. As they progress, they should ask clarifying questions to understand whether a problem or part of a problem can be solved using a computational approach. For example, before attempting to write an algorithm to sort a large list of names, students may ask questions about how the names are entered and what type of sorting is desired.

2. Decompose complex real-world problems into manageable subproblems that could integrate existing solutions or procedures.

At any grade level, students should be able to break problems down into their component parts. In the early grade levels, students should focus on breaking down simple problems. For example, in a visual programming environment, students could break down (or decompose) the steps needed to draw a shape. As students progress, they should decompose larger problems into manageable smaller problems. For example, young students may think of an animation as multiple scenes and thus create each scene independently. Students can also break down a program into sub goals: getting input from the user, processing the data, and displaying the result to the user.

3. Evaluate whether it is appropriate and feasible to solve a problem computationally.

After students have had some experience breaking problems down (P3.2) and identifying sub-problems that can be solved computationally (P3.1), they should begin to evaluate whether a computational solution is the most appropriate solution for a problem.

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.

By the end of Grade 5, students should be able to:

1. Extract common features from a set of interrelated processes or complex phenomena.

Students at all grade levels should be able to recognize patterns. Young learners should be able to identify and describe repeated sequences in data or code through analogy to visual patterns or physical sequences of objects. As they progress, students should identify patterns as opportunities for abstraction, such as recognizing repeated patterns of code that could be more efficiently implemented as a loop.

2. Model phenomena and processes and simulate systems to understand and evaluate potential outcomes.

Students at all grade levels should be able to represent patterns, processes, or phenomena. With guidance, young students can draw pictures to describe a simple pattern, such as sunrise and sunset, or show the stages in a process, such as brushing your teeth. They can also create an animation to model a phenomenon, such as evaporation. As they progress, students should understand that computers can model real-world phenomena, and they should use existing computer simulations to learn about real-world systems. For example, they may use a preprogrammed model to explore how parameters affect a system, such as how rapidly a disease spreads.

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.

By the end of Grade 5, students should be able to:

1. Plan the development of a computational artifact using an iterative process that includes reflection on and modification of the plan, considering key features, time and resource constraints, and user expectations.

At any grade level, students should participate in project planning and the creation of brainstorming documents. The youngest students may do so with the help of teachers. With scaffolding, students should gain greater independence and sophistication in the planning, design, and evaluation of artifacts.

2. Create a computational artifact for practical intent, personal expression, or to address a societal issue.

Students at all grade levels should develop artifacts in response to a task or a computational problem. At the earliest grade levels, students should be able to choose from a set of given commands to create simple animated stories or solve pre-existing problems. Younger students should focus on artifacts of personal importance. As they progress, student expressions should become more complex and of increasingly broader significance.

3. Modify an existing artifact to improve or customize it.

At all grade levels, students should be able to examine existing artifacts to understand what they do. As they progress, students should attempt to use existing solutions to accomplish a desired goal. For example, students could attach a programmable light sensor to a physical artifact they have created to make it respond to light.

Practice 6: Testing and Refining Computational Artifacts

Testing and refinement are 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.

By the end of Grade 5, students should be able to:

1. Systematically test computational artifacts by considering all scenarios and using test cases.

At any grade level, students should be able to compare results to intended outcomes. Young students should verify whether given criteria and constraints have been met. As students progress, they should test computational artifacts by considering potential errors, such as what will happen if a user enters invalid input.

2. Identify and fix errors using a systematic process.

At any grade level, students should be able to identify and fix errors in programs (debugging) and use strategies to solve problems with computing systems (troubleshooting). Young students could use trial and error to fix simple errors. For example, a student may try reordering the sequence of commands in a program. In a hardware context, students could try to fix a device by resetting it or checking whether it is connected to a network. As students progress, they should become more adept at debugging programs and begin to consider logic errors: cases in which a program works, but not as desired. In this way, students will examine and correct their own thinking. For example, they might step through their program, line by line, to identify a loop that does not terminate as expected.

3. Evaluate and refine a computational artifact multiple times to enhance its performance, reliability, usability, and accessibility.

After students have gained experience testing (P6.2), debugging, and revising (P6.1), they should begin to evaluate and refine their computational artifacts. As students progress, the process of evaluation and refinement should focus on improving performance and reliability. For example, students could observe a robot in a variety of lighting conditions to determine that a light sensor should be less sensitive.

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.

By the end of Grade 5, students should be able to:

1. Select, organize, and interpret large data sets from multiple sources to support a claim.

At any grade level, students should be able to refer to data when communicating an idea. Early on, students should, with guidance, present basic data using visual representations, such as storyboards, flowcharts, and graphs. As students progress, they should work with larger data sets and organize the data in those larger sets to make interpreting and communicating it to others easier, such as through the creation of basic data representations.

2. Describe, justify, and document computational processes and solutions using appropriate terminology consistent with the intended audience and purpose.

At any grade level, students should be able to talk about choices they make while designing a computational artifact. Early on, they should use language that articulates what they are doing and identifies devices and concepts they are using with correct terminology (e.g., program, input, and debug). Younger students should identify the goals and expected outcomes of their solutions. Along the way, students should provide documentation for end users that explains their artifacts and how they function, and they should both give and receive feedback. For example, students could provide a project overview and ask for input from users. As students progress, they should incorporate clear comments in their product and document their process using text, graphics, presentations, and demonstrations.

3. Articulate ideas responsibly by observing intellectual property rights and giving appropriate attribution.

All students should be able to explain the concepts of ownership and sharing. Early on, students should apply these concepts to computational ideas and creations. They should identify instances of remixing, when ideas are borrowed and iterated upon, and give proper attribution. They should also recognize the contributions of collaborators.