

TECH TIP

Computer Science Education Principles

Philosophy

The Tech and San Jose State University's College of Science Jay Pinson STEM Education program believe that ALL youth should be exposed to Computational Thinking (CT), computer science, computing and computer programming. We believe that "Computer Science for All" requires that these areas be woven throughout K-12 experiences to ensure that, in today's technology-driven society, all students gain equitable access to opportunities now and in the future.

We place a special emphasis on computational thinking since it is fundamental to computer programming and problem-solving, skills that allow youth to be creators of digital content, not just consumers. CT is a "problem-solving process" that can be broadly applied across content areas and everyday life. This includes: decomposition, algorithm design, pattern recognition and abstraction (to learn more see our [Computational Thinking Tech Tip](#)).



Principle I: Computational Thinking can be integrated with any subject

Computational Thinking is an approach to problem-solving, applicable to most subjects. Introducing CT, computing, and computer science references that have compelling contexts, allows learners to see direct applications to the real-world and various career fields. Through computing, models can be developed to explore creative topics, such as music theory or real-world issues like climate change. These models can be used to support communication around complex ideas, innovation and data-driven designs. When CT and computing are authentically embedded into different subjects, it allows those in underrepresented communities to engage in computer science.

Principle II: Use Computational Thinking as a foundation

Teaching Computational Thinking can be done in many different ways on and off the computer. Youth can apply CT skills to think logically about solving a real-world problem and, with these strong foundations, be more prepared to develop computer programs as well as non-software solutions to interesting problems. For example, in The Tech's model lessons, youth engage in unplugged activities where they review a data set, identify patterns and infer missing data. Youth then use this robust foundation to develop a computer-based solution.

Principle III: Focus on Computational Thinking, not the tool or language

There are many tools, languages, environments and platforms available to enhance CT skills. It's important to focus intently on teaching CT skills, responding to the interests of youth, and selecting the tools appropriate for the problems being addressed. For example, a block-based environment may be appropriate for early elementary or as a first coding exposure for older students, whereas a text programming environment may serve the needs of more advanced youth. Regardless of the tools being used, the CT skills being taught should be called out and explicitly emphasized.

Principle IV: Tinker and explore!

Youth should be given time to tinker with and explore a new programming environment or computer program. This non-evaluative tinkering and exploration time increases equity such that youth with less experience can develop some comfort and confidence in a low stakes activity. When young people are confronted with a new program and provided with a chance to explore it, they can systematically manipulate it and develop an understanding of how it works.

Principle V: Ask questions, resist the urge to provide the answer

When youth are learning CT and programming, the types of questions they are asked can influence mindsets and growth. Ask open-ended questions that promote collaboration and active learning, such as: “What do you think will happen if...” or “What first step might you take to accomplish your goal?” The educator doesn’t need to know all of the answers; they are there to facilitate.

Principle VI: Iterate!

Designing a computer program or model is an ongoing and iterative process based on the data and evidence available. For example, the OS software on their phone has gone through many iterations and continues to be modified. Youth should be asked to actively iterate and improve their work.

Principle VII: There are many solutions to a given problem

In programming, there are many ways to solve a problem. Educators should celebrate when youth develop functional solutions that are different from what is expected. As with travel, there are many routes one can take; some prioritize time efficiency while others are shorter in distance but may require more time. In the end, regardless of the route taken, one will arrive at the destination. In general, the computer program and output should be assessed on creative success that meets stated goals and/or rubric items. Processes of iteration and optimization can enhance creativity and help reach stated goals.


Principle VIII: Call out connections to computer science!

It is critical to call out the CT elements that youth are experiencing in the moment and in many different contexts to help them see the connections of these skills to real life and to the field of computer science. When we explicitly make these connections, youth realize that they can develop proficiency in CT skills that can be used to problem-solve. This “call out” normalizes the academic language and expands access to computer science, a sometimes intimidating field, by reducing the “fear” of the unknown. Youth who otherwise may never have considered computer science may begin to see themselves as computer scientists.

Principle IX: Celebrate diversity, provide a safe space for ALL and embrace a growth mindset

Diversity of skills and interests should be celebrated, leveraged and valued. A flexible structure and a safe space should be provided for ALL to explore and grow their computing skills. Adopting and modeling a growth mindset (learning from mistakes and effort), is crucial. When youth are not finding immediate success in their program, they can test smaller sections of their program to check what is working (block testing). If youth have met the given objectives, help them set a goal to improve their program. There will be different levels of familiarity and ability in any group and it’s important, from the beginning, to establish a collaborative learning culture where everyone feels empowered to explore and persevere.

These computer science principles were developed in partnership with San Jose State University’s College of Science.

 For more resources and lessons on computer science and computational thinking see thetech.org/ctlessons.