InspireCT

Computational Thinking Practices

For adults, Computational Thinking is a set of practices used in computing to frame problems, create solutions, express ideas, and understand the impact of technology in our world.

For kids, Computational Thinking is finding ways to skillfully bring together what humans are good at and what computers are good at.

What might CT look like in your classroom? Check out these examples of CT integration!

CT Practice	Essential Questions	As a creator, I can...	As a community member, I can...	As a programmer, I can...
Data Collection and Analysis *Collecting information and finding connections and patterns. Computers can make it easier and faster.*	- What information can I keep track of or collect from my world? - How can a computer help me make predictions or see patterns with data?	- Decide what data is useful to collect. - Identify patterns in data. - Use data to make predictions or explore cause-and-effect relationships. - Find the mean, mode, and median for a data set. - Use data to communicate an idea or to make an argument.	- Explain to someone else how information is represented, stored, and transformed by a computer. - Collect, share, and find patterns in data with a team. - Explain to someone else why computers are useful when sorting and analyzing data. - Consider how data about my community is shared, collected,	- Create programs that use variables to store and modify data. - Use mathematical and logical operators to calculate and compare pieces of data. -Use input to change the outcome of a program.

			or protected.	
Algorithm Design *Creating an algorithm, or a series of ordered steps, in order to complete a task (kind of like a recipe).*	- What steps should be taken to complete this task? - Is this the best answer? - Can this algorithm be applied to other tasks? - Can this problem be solved?	- Describe your favorite sandwich and write an algorithm to tell someone else how to make it. - Design a board game and write instructions for how to play. - Write an algorithm to explain part of a natural environment or system.	- Share and explain how an algorithm I've written can be used by someone else. - Compare different algorithms for completing the same task and make an argument for which is best. - Examine the impacts of an algorithm's design on my family or community.	- Use events and sequencing to decide on the order of steps in code and what will begin the algorithm. - Use conditional and binary logic to program decision-making. - Use loops to repeat parts of a program. - Use sensor data to automate part of an algorithm. - Create an algorithm that works for many tasks of a certain type – for example, finding a path out of any maze.
Abstracting and Modularizing *Extracting essential details and repeatable patterns from a more complex system.*	- What patterns can I find? - What details can be simplified to find a solution? - Can this solution work for my community? - How can patterns or parts from one system or solution be used in	- Describe how I might use patterns to express an idea. - Create a visual of an idea or system and decide which details should be included and which should be left out. - Create a diagram to break a larger task down into smaller pieces and explain the	- Read a text, summarize its meaning, and share the most important parts. - Check to make sure my idea or solution can work for people other than myself. - Solve a problem collaboratively by identifying tasks that can be handled by different members	- Organize different parts of the code (for example in Scratch) and explain the reasoning. - Identify a program function that is used multiple times and save it to reuse. - Program an interactive model of a

	another?	connection between the parts. - Examine a problem to look for similarities, repetition, or conditional relationships.	of a team.	system to experiment with changes over time or test a hypothesis. - Modify, remix, or incorporate portions of an existing program into my work, to develop something new or add more advanced features.
Debugging *Testing a solution and working through problems as they arise.*	- What is not working the way I would like it to? - How can I test a smaller part of the problem?	- Revise a model or idea over time to show changes in understanding. - Break down different parts of something created to determine which parts are not working.	- Clearly articulate the problem to a friend. - Modify ideas after getting feedback. - Suggest changes to a classmate's model, plan, or idea.	- Isolate different parts of a program or solution in order to figure out what's causing the issue. - Add comments to code to confirm understanding. - Explain choices made during program development using code comments, presentations, and demonstrations.

These practices were summarized from a variety of sources, including: SF CS4All, NYC CS4All, ISTE/CSTA CT Standards, "A K-6 Computational Thinking Curriculum Framework: Implications for Teacher Knowledge" by Angeli et al, K12CS Framework, Computing Progression Pathways (UK), CSTA CS Standards.