Great Public Schools for Every Student

Programming and Computational Thinking for Beginners

Educator designs integrated learning experiences to solve problems through algorithms and computational thinking.

Key Method

The educator creates integrated lessons on programming that include the following components: decomposition, pattern recognition, abstraction, and the design of algorithms.

Method Components

What is Computer Science?

Computer science moves beyond using technology tools toward an understanding of how they work and ultimately designing new solutions to enduring human problems. Despite common misperceptions, computer science is not only about programming. Like any scientific discipline, computer science consists of a body of knowledge that informs how people understand and perceive the world around them, as well as practices for exploration, creation, and experimentation. Programming, defined as giving computers instructions to follow, is a practice used in computer science. The field itself is much broader, much as biology is not simply conducting lab experiments.

Why Should Students Learn Computer Science?

- Over 70% of jobs in STEM are actually computing jobs, and most of the others use computer science as a core part of the job.
- Many future jobs and opportunities will require knowledge and skills in the area of computer science. Therefore, students need multiple opportunities to use computer science to explore and understand the world.
- Even a student who does not end up programming in their job will still need to understand the central principles of how data, networking, the Internet, and cybersecurity impact the lives of people in their families and communities.
- Students need to know that when they use a free social media platform, their data can be shared with anyone.
- All of the strands of computer science have drastic impacts on how we live our lives,
- Understanding the basic principles of computer science influences how students will interact with the world around them.

What is Computational Thinking?

Computational thinking can be defined as a method of solving problems that can be applied to problem-solving across disciplines. Although it is essential to the development of computer science, computational thinking can be used to solve problems in all areas. Using computational thinking allows students to see connections between content areas inside and outside of the classroom. There are four cornerstones of computational thinking:

- Decomposition
- Abstraction
- Pattern recognition
- Algorithms

What is Decomposition?

Decomposition is when you break down problems into smaller parts and then plan how to fit them together in a suitable order to solve the problem. This plan is called an algorithm.

What is Abstraction?

Abstraction is the process by which general attributes are associated with objects in order to develop models that can be re-used without having to rewrite the code for each new application.

What is Pattern Recognition?

Pattern recognition is a skill used in computer science where one looks for similarities and patterns among small, decomposed problems. These patterns can help solve more complex problems efficiently.

What is an Algorithm?

An algorithm is a procedure for completing a process. Algorithms are the step-by-step instructions that can be repeated based on rules. Routines in our everyday life are examples of algorithms. Each step within an algorithm is completed in order or selected depending on the input. Algorithms can be written out step by step, such as in a recipe, flowchart, or in paragraph form.

In computer science, an algorithm must use a common set of instructions that are clearly defined and yield consistent results. The instructions are carried out in the correct order to generate the desired result.

What is Programming?

Programming is the art of encoding algorithms into a programming language. The programming language provides the instructions to enable a computing device to perform various tasks.

Teaching Beginning Programming and Computational Thinking

The ability to recognize appropriate and worthwhile opportunities to apply computational thinking is a skill that develops over time and is central to computing. To solve a problem using a computational approach, the problem must be defined and broken down into parts; each part must then be evaluated to determine whether a computational solution is appropriate.

By the end of Grade 12, students should be able to:

- Identify complex, interdisciplinary, real-world problems that can be solved computationally.
- Decompose complex real-world problems into manageable subproblems that could integrate existing solutions or procedures.
- Evaluate whether it is appropriate and feasible to solve a problem computationally.

Beginners' Programming Terms and Concepts

To effectively teach basic programming and computational thinking, educators need to understand the terms and concepts listed below.

Terms

- Algorithm: a series of steps or rules taken in order to complete a task.
- Program: written instructions for a computer to follow. Many programming languages exist and are used for different purposes.
- Sequence: the order of events in a program.

- Event: the actions taken by the computer according to the program being used.
- Variable: the items or names in a computer program that store data for manipulation within the program. Many types of variables exist, each with a different purpose. Variables can be numbers, constants, words, and so on.
- Object: the individual items that can be acted upon by a program. Objects have behaviors and classes.
- Computational Thinking:
- Selection: Within a computer program, actions can be selected based on input. Decisions that impact the sequence of events can change according to the input.
- Iteration: repeating processes until a condition is met within a program.
- Boolean: a system of logical processes for selecting the next step. There are two possible outcomes, e.g., =, >, <.

Concepts

- Flowcharts or pseudocode
- Variables (declaring, initializing, data types, and/or data structures*)
- Variable scope
- Operators/operands
- Logical operators
- Methods/functions
- Arguments/parameters
- User input, reading user input, and/or file input/output
- Conditional statements (if, if-else, switch statements)
- Repetitive statements (while, do-while, for loops; may include recursion)
- Nested structures
- Data structures
- Pair programming
- Use of comments
- Programming efficiency
- Intellectual property rights

Supporting Research

Burgstahler, Sheryl. "Differentiating for Diversity: Using Universal Design for Learning in Elementary Computer Science Education." Universal Design: Implications for Computing Education, ACM Transactions on Computing Education, Oct. 2011, https://staff.washington.edu/sherylb/ud_computing.html

Honey, Margaret, et al. "STEM Integration in K–12 Education: Status, Prospects, and an Agenda for Research." The National Academies Press, National Academy of Engineering and National Research Council of The National Academies, 7 Feb. 2014, http://www.nap.edu/catalog/18612/stem-integration-in-k-12-education-status-prospects-and-an

Lewis, Colleen, and Niral Shah. "How Equity and Inequity Can Emerge in Pair Programming." Association for Computing Machinery, ICER '15 Proceedings of the Eleventh Annual International Conference on International Computing Education Research, July 2015, http://blogs.hmc.edu/lewis/wp-content/uploads/sites/2/2013/07/LewisShah2015_EquitySpeed.pdf

Lewis, Colleen M. "Good (and Bad) Reasons to Teach All Students Computer Science." SpringerLink, Springer, Cham, 1 Jan. 2017, https://docs.google.com/document/d/1R57koI5El5B6jZQyZkmG4NY9MM4wwfJ13V13Yx4gWzw/edit#heading=h.gjdgxs

Resources

Computer Science

Aronson, Leslie, and Jake Baskin. "Guide to Inclusive Computer Science Education: How Educators Can Encourage and Engage All Students in Computer Science." National Center for Women & Information Technology, National Center for Women & Information Technology, 22 May 2019, www.ncwit.org/resources/guide-inclusive-computer-science-education-how-educators-can-encourage-and-engage-all.

Programming

Overview: Algorithms and Programming https://teacherslounge.codevirginia.org/portal/kb/articles/overview-algorithms-and-programming
Tools of Programming
https://teacherslounge.codevirginia.org/portal/kb/articles/tools-of-programming
Computing at Schools (tenderfoot) QuickStart Subject Knowledge: Programming
https://community.computingatschool.org.uk/files/8263/original.pdf

Computational Thinking

Google Computational Thinking Course
https://computationalthinkingcourse.withgoogle.com

Computing at Schools (tenderfoot) QuickStart Subject Knowledge: Computational Thinking
https://community.computingatschool.org.uk/files/8261/original.pdf

Computational Thinking for All
http://www.iste.org/explore/Solutions/Computational-thinking-for-all?articleid=152

Computational Thinking Operational Definition
https://id.iste.org/docs/ct-documents/computational-thinking-operational-definition-flyer.pdf?sfvrsn=2

Digital Promise: Computational Thinking
https://digitalpromise.org/initiative/computational-thinking/

Teaching Resources
Create a Basic Flowchart
https://support.office.com/en-us/article/create-a-basic-flowchart-e207d975-4a51-4bfa-a356-eeec314bd276

Twine
Twine is an open-source tool for telling interactive, nonlinear stories.
https://twinery.org/

Google Drawing
https://docs.google.com/drawings

Draw.io
https://www.draw.io/

Canva
https://www.canva.com/graphs/flowcharts/

Submission Guidelines & Evaluation Criteria

To earn the micro-credential, you must receive a passing score in Parts 1 and 3, and receive a proficient for all components in Part 2.

Part 1. Overview Questions

250-500

Please answer the following contextual questions to help our assessor understand your current situation. Please do not include any information that will make you identifiable to your reviewers.

1. Identify your current position and title, number of years working in education, and description of your school community climate (demographic information).

2. Describe your current level of knowledge in programming, including any formal or informal training you have had.
3. What impact on your students do you hope completing this micro-credential will have?
4. In the field of computer science, women and minorities are underrepresented. How will you intentionally differentiate instruction to engage and inspire underrepresented groups through the design of your unit?

- **Passing:** Passing: Response provides reasonable and accurate information that justifies the reason for choosing this micro-credential to address specific needs of both the teacher and the student. Educator includes a learning goal that describes what they hope to gain from earning this micro-credential. Specific details about how you will engage and inspire underrepresented minorities and girls are included.

Part 2. Work Examples / Artifacts

To earn this micro-credential, please submit the following **three artifacts** as evidence of your learning. Please do not include any information that will make you or your students identifiable to your reviewers.

Artifact 1: Algorithm for a process or routine

Write an algorithm for a common routine, recipe, or story that includes multiple steps, repeating or looping actions, and beginning and end of process. Describe how your algorithm can be repeated by anyone and achieve the same result. This may be done by creating a trace table to correspond with the algorithm.

Elements in the final project should include five or more of the following:

- Description of the problem to be solved,
- Variables explained,
- Input or data reading,
- Process(es) to complete the task,
- Loops,
- Comparison or decision steps,
- Output,
- Beginning and end of algorithm.

Artifact 2: Lesson plan

Create a series of integrated lesson plans or plans for each of the following components of computational thinking:

- Decomposition,
- Pattern recognition,
- Abstraction,
- Designing algorithms.

Your lessons should include:

- CSTA Standards and/or State CS Standards addressed,
- Content learning outcomes,
- Description of the lesson (the problem to be solved, the steps and patterns taught, and how you will

explicitly teach each of the four components),

- How you will intentionally engage and inspire underrepresented minorities and females,
- How the learning will be evaluated/assessed.

Artifact 3: Flowchart or pseudocode

Submit one of the final algorithms in either flowchart or pseudocode format for one of the lessons in **Artifact 2**. This may be either a sample of student work or one completed by the teacher as part of the lesson depending on the grade level. It should show the development of the algorithm or process that meets the criteria of the lesson.

null	Proficient	Basic	Developing
Artifact 1: algorithm for a process or routine	Algorithm is created in either a pseudocode format or a flowchart. The task is described sufficiently to understand the process. The code includes: at least five of the required elements. The problem is solved and, if reasonable, a trace table is provided showing the process and output(s).	Algorithm is complete in either pseudocode or flowchart form. There are fewer than five elements within the algorithm. No trace table is used, nor does it show completion of the task.	Algorithm is not complete in either pseudocode or flowchart form. No problem is defined, or the problem is not one that can be solved by an algorithm.
Artifact 1: algorithm for a process or routine Artifact 2: lesson plans	Lesson plan or unit is clearly written. It demonstrates integration of content with technology. Students have the opportunity to experience all components of computational thinking: -Decomposition, -Pattern recognition, -Abstraction, -Algorithms. It is clear to the assessor how each component was or will be taught as part of the lesson. Includes the expected student or classwork to demonstrate learning that matches the lesson.	Lesson plan or unit is clearly written. It includes fewer than the four components of computational thinking OR it is not integrated with a content other than technology. The means of teaching the individual components are not explained in the lesson/unit. It may not include the expected student work.	Lesson or unit plan does not incorporate: all four components of computational thinking AND it is not clearly integrated with a separate content. No expected student work is indicated or it does not match with the expected algorithm design.

Artifact 3: flowchart or pseudocode	Student work or shared work (if taught as a whole-group activity) demonstrates that the algorithm is designed to meet the problem as expected. The final flowchart or pseudocode effectively shows how the problem was solved.	The student work or group work does not include information to show the outcome of the lesson. The algorithm is not in a flowchart or pseudocode format. The algorithm does not effectively solve the problem described in the lesson.	The student work is not included or it does not match the lesson.

Reflection

250-500

Please answer the following reflective questions. Please do not include any information that will make you identifiable to your reviewers.

1. How did completing this micro-credential process impact your understanding of teaching computer science?
2. What was the change in your students' belief in themselves as a computer programmer? (Note whether there are any students in underrepresented groups that were impacted.)
3. How will you continue to include computational thinking and/or programming into your classroom practices?

- **Passing:** Passing: Reflection provides evidence that this activity has had a positive impact on both educator practice and student success. Specific examples are cited directly from personal or work-related experiences to support claims. Also included are specific actionable steps to demonstrate how new learning will be integrated into future practices.