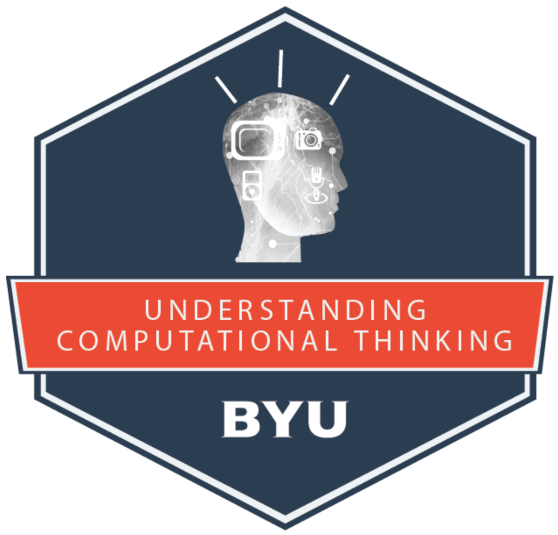


Understanding Computational Thinking

Badge Guide

Enoch Hunsaker
Brigham Young University
2018

Badge Introduction

Computational Thinking (CT), considered a fundamental literacy of the 21st Century, is increasingly gaining traction with educational associations and school systems around the world. Earners of this badge demonstrate ability to apply skills, attitudes, and approaches associated with CT in their own problem-solving process as well as an understanding of the rationale for teaching CT to 21st-Century students.

It is estimated that this badge will take approximately 2 hours to complete.

Recommended Prior Knowledge

This is a beginner-level badge. No prior knowledge is necessary.

International Standards Alignment

ISTE Standards for Educators

- **Learner Standard 1a:** Set professional learning goals to explore and apply pedagogical approaches made possible by technology and reflect on their effectiveness.

ISTE Standards for Students

- **Computational Thinking Standard 5:** Students develop and employ strategies for understanding and solving problems in ways that leverage the power of technological methods to develop and test solutions.

Intended Learning Outcomes

1. Define computational thinking (CT) and basic terms associated with it.
2. Apply basic CT principles in a variety of problem-solving contexts.
3. Articulate rationale for teaching computational thinking.

Acknowledgements

Some of the following content is adapted from Integrating Computational Thinking (Hunsaker, n.d.), a chapter in the open educational textbook K-12 Technology Integration published on Pressbooks. Special thanks to Peter Rich for invaluable insights and contributions.

Requirement 1: CT Conceptual Quiz

After learning about Computational Thinking (e.g., from this Badge Guide), take the quiz below. ***You are allowed to take the quiz only once,*** *and you will need to login to your Google account to do it. If you achieve a score of 90% (24/26) or greater, simply take a screenshot that includes your name and your score and attach it to your submission form.*

If your score is less than 90% (23 or lower), review your responses and the Badge Guide, determine the correct answers, and do one of the following:

- *Access the email sent to you by "BYU Instructor" that contains a Google sheet with your score and a space to write corrections for any questions you missed. Enter your corrections and paste the URL of the sheet onto your submission form (**make sure anyone with a link can view**).*
- *If you find it easier (or if you do not receive the email from BYU Instructor), create a screen recording of your quiz results in which you pause on each question you missed and verbally explain the correct answer. Upload this video to YouTube,* ***make sure anyone with a link can view*** *it, and paste the URL onto your submission form.*

Understanding Computational Thinking - Quiz

What is Computational Thinking

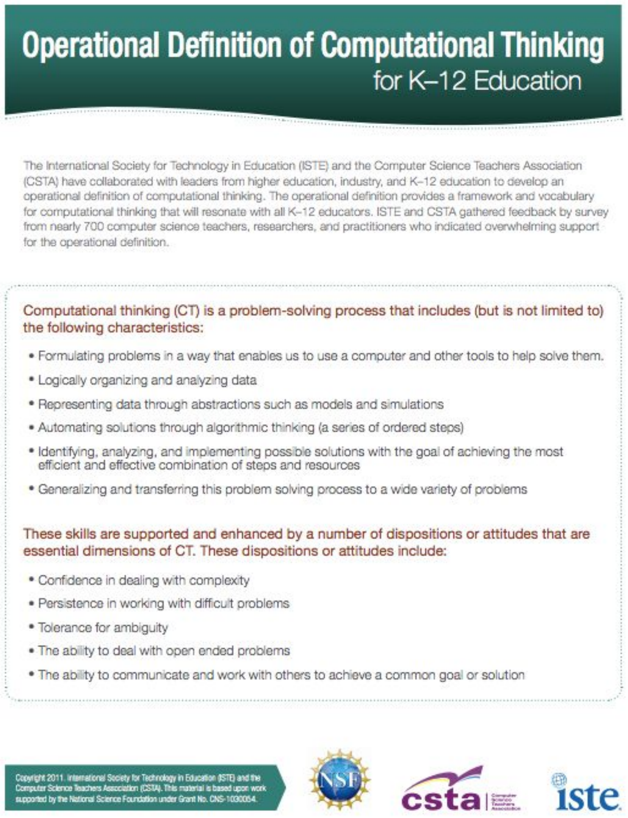
In order to pass the above quiz, you will need to understand computational thinking (CT) and several of the skills, attitudes, and approaches commonly associated with it.

The term *computational thinking* was introduced by Seymour Papert in 1980, but it was used in a slightly different sense than is currently accepted (Standl, 2017). In 2006, Jeannette Wing reintroduced, redefined, and reinvigorated computational thinking. Her article (see references) is only three pages and well worth reading, but for the purposes of this badge, it will do to understand the following points:

Abridged Digest of *Computational Thinking* (Wing, 2006) *The following are direct quotes from Wing's article.*
• "Computational thinking builds on the power and limits of computing processes, whether they are executed by a human or by a machine."

- "Computational thinking confronts the riddle of machine intelligence: What can humans do better than computers? And what can computers do better than humans?"
- "Computational thinking is a fundamental skill for everyone, not just for computer scientists. To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability."
- "We have witnessed the influence of computational thinking on other disciplines. For example, machine learning has transformed statistics. Statistical learning is being used for problems on a scale, in terms of both data size and dimension, unimaginable only a few years ago. . . . Computational biology is changing the way biologists think. Similarly, computational game theory is changing the way economists think; nanocomputing, the way chemists think; and quantum computing, the way physicists think."
- "Computational thinking thus has the following characteristics:
  - Conceptualizing, not programming
  - Fundamental, not rote skill
  - A way that humans, not computers think
  - Complements and combines mathematical and engineering thinking
  - Ideas, not artifacts
  - For everyone, everywhere"

Wing's article started a movement that continues to grow in popularity. In 2011, the International Society for Technology in Education (ISTE) and the Computer Science Teachers Association (CSTA) cooperated to create a 1-page Operational Definition of Computational Thinking for K-12 Education. Please read this definition thoroughly.



Operational Definition of Computational Thinking
for K–12 Education

The International Society for Technology in Education (ISTE) and the Computer Science Teachers Association (CSTA) have collaborated with leaders from higher education, industry, and K–12 education to develop an operational definition of computational thinking. The operational definition provides a framework and vocabulary for computational thinking that will resonate with all K–12 educators. ISTE and CSTA gathered feedback by survey from nearly 700 computer science teachers, researchers, and practitioners who indicated overwhelming support for the operational definition.

Computational thinking (CT) is a problem-solving process that includes (but is not limited to) the following characteristics:

- Formulating problems in a way that enables us to use a computer and other tools to help solve them.
- Logically organizing and analyzing data
- Representing data through abstractions such as models and simulations
- Automating solutions through algorithmic thinking (a series of ordered steps)
- Identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources
- Generalizing and transferring this problem solving process to a wide variety of problems

These skills are supported and enhanced by a number of dispositions or attitudes that are essential dimensions of CT. These dispositions or attitudes include:

- Confidence in dealing with complexity
- Persistence in working with difficult problems
- Tolerance for ambiguity
- The ability to deal with open ended problems
- The ability to communicate and work with others to achieve a common goal or solution

You may also find it helpful to watch this video created by the International Society for Technology in Education (ISTE):

Video 1. Computational Thinking: A Digital Age Skill for Everyone. Click, copy, or type this link into your browser to view: https://youtu.be/VFcUgSYyRPg. ***Note:*** *This video is copyrighted by ISTE under Standard YouTube Licence and is therefore not included in the CC-BY license of the rest of this document.*

Components of Computational Thinking

While the ISTE/CSTA definition is thorough, it is also useful for teachers to be familiar with a handful of key terms that they can keep in mind when planning lessons, guiding discussions, commenting on student work, etc. The following table is derived from the documentation of various organizations that seek to define and categorize CT in a useful way for educators (CAS Barefoot, 2014b; Google, n.d.b; ISTE, 2014). This is not intended to be comprehensive, but it does provide a reasonably complete snapshot of the most crucial components of CT.

These components or characteristics of computational thinking are a double-edged sword: possessing these skills, attitudes, and attributes enhances one's ability to think computationally; *and* practicing computational thinking can enhance these attitudes and skills.

Components of Computational Thinking		
Skills	**Attitudes**	**Approaches**
● ***Decomposition***: Breaking down data, processes, or problems into smaller, manageable parts. ● ***Pattern Recognition:*** Observing patterns, trends, and regularities in data. ● ***Abstraction:*** Creating a visual model or simulation of a problem that incorporates only the most important details. ● ***Algorithm Design:*** Developing the step by step instructions for solving this and similar problems. ● ***Evaluation:*** Ensuring that your solution is a good one.	● ***Confident:*** Believing in one's own ability to solve problems. ● ***Communicative:*** Willing and able to communicate effectively with others. ● ***Flexible:*** Able to deal with change and open-ended problems.	● ***Tinkering:*** Experimenting and playing. ● ***Creating:*** Designing and making. ● ***Debugging:*** Finding and fixing errors. ● ***Persevering:*** Keeping going. ● ***Collaborating:*** Working together.

Table 1. Components of Computational Thinking

Decomposition

Decomposition is “breaking down data, processes, or problems into smaller, manageable parts” (Google, n.d.b). Watch this 2-minute video to better understand this principle:

Video 2. Introduction to Decomposition. Click, copy, or type this link into your browser to view: https://youtu.be/rxsYpP2-omg. ***Note:*** *This video is copyrighted by Robotics Academy under Standard YouTube Licence and is therefore not included in the CC-BY license of the rest of this document.*

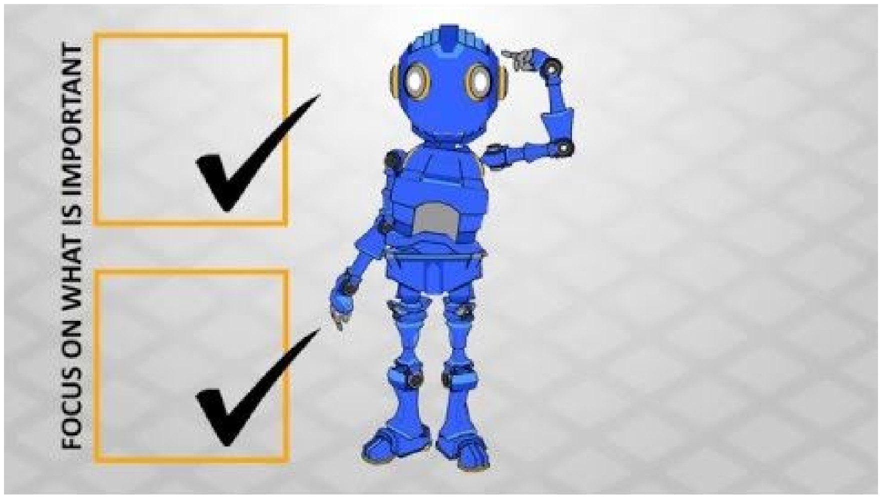
Pattern Recognition

Pattern recognition is “observing patterns, trends, and regularities in data” (Google, n.d.b). To see how this principle is particularly relevant in computer science, watch this 2.5-minute video:

Video 3. Pattern Recognition - Introduction. Click, copy, or type this link into your browser to view: https://youtu.be/cbZUnuyxcVs. ***Note:*** *This video is copyrighted by Computer Science Education Research Group under Standard YouTube Licence and is therefore not included in the CC-BY license of the rest of this document.*

Abstraction

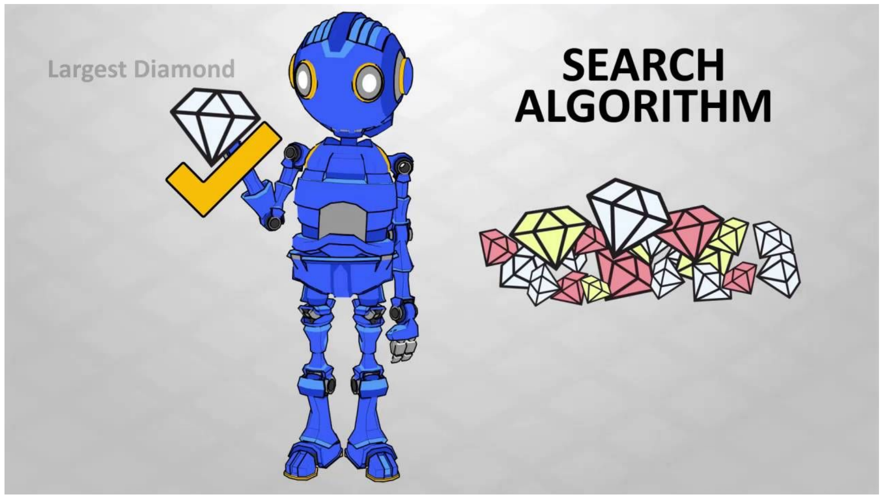
Abstraction is creating a visual model or simulation of a problem that incorporates only the most important details. Watch this 2.5-minute video for more detail:



Video 4. Abstraction - Computational Thinking. Click, copy, or type this link into your browser to view: https://youtu.be/jV-7Hy-PF2Q. ***Note:*** *This video is copyrighted by Robotics Academy under Standard YouTube Licence and is therefore not included in the CC-BY license of the rest of this document.*

Algorithm Design

Algorithm design is "developing the step by step instructions for solving" the problem at hand as well as other problems that may be similar (Google, n.d.b). This 2-minute video may help you better visualize this concept:



Video 5. Algorithms. Click, copy, or type this link into your browser to view: https://youtu.be/ROUV90QmqUA. ***Note:*** *This video is copyrighted by Robotics Academy under Standard YouTube Licence and is therefore not included in the CC-BY license of the rest of this document.*

Evaluation

Evaluation is an important step in a computational thinking problem-solving task because it helps ensure that the algorithmic solution is both functional under a wide variety of circumstances and suitable for accomplishing the task at hand.

Evaluation is not included as a step in Google's definition of CT skills, but CAS Barefoot (2014b) describes evaluation as follows:

CAS Barefoot Definition of Evaluation
"In computer science, evaluation is systematic and rigorous; it is about judging the quality, effectiveness and efficiency of solutions, systems, products and processes. Evaluation checks that solutions do the job they are designed to do and are fit for purpose. One approach to systematic evaluation could be to use:

- specific criteria (for example a design goal or specification)
- user needs (considering who the users are and what they need from the final design)”

Confident

For the purposes of this badge, *confident* is defined as “believing in one’s own ability to solve problems.” Confidence is central to CT because its problems are usually open-ended and ill-defined.

Communicative

For the purposes of this badge, *communicative* is being “willing and able to communicate effectively with others.” CT problems often involve communicating with others in order to effectively find and implement a solution.

Flexible

For the purposes of this badge, *flexible* means “able to deal with change and open-ended problems.” Problems that lend themselves to computational solutions are often, to the naked eye, overwhelmingly large and ill-defined. Solutions often need to be adjusted or “debugged,” and sometimes the data we get from our computational methods may even cause us to adjust our perception of the problem itself. Being adaptable to these changes contributes the persistence needed to solve these types of problems.

Tinkering

Tinkering is defined as “experimenting and playing” (CAS Barefoot, 2014b). It should be “fun, free, creative and full of questions and surprises” (CAS Barefoot, 2014f). Tinkering is also linked with the fostering of perseverance (another key CT approach) and logical reasoning. Computational methods are well-suited to tinkering because computers can execute commands and provide feedback in a matter of nanoseconds, making it natural and easy to try something out, see its effect, and adjust quickly.

Creating

Creating (“designing and making”) (CAS Barefoot, 2014b) is fundamental to computational thinking. The product of the creative process of CT is an algorithm—a step by step process that both humans and machines can understand. Ultimately, this algorithm powers a solution that leverages the power of computational automation.

Debugging

Debugging is defined as “finding and fixing errors” (CAS Barefoot, 2014b). “Errors in algorithms and code are called ‘bugs,’ and the process of finding and fixing these is called ‘debugging’. Debugging can often take much longer than writing the code in the first place” (CAS Barefoot, 2014c). Computers are very good at executing commands, but very bad at interpreting what the algorithm designer actually *meant*.

Persevering

Persevering means “keeping going” (CAS Barefoot, 2014b). Solving CT problems professionals tackle can often take months or even years. Even shorter student projects can sometimes be arduous and boring (especially during the debugging stages). Computational thinkers need to have “a tolerance for confusion” (CAS Barefoot, 2014e) and be able to continue their process in the face of that confusion.

Collaborating

Collaborating is, very simply, “working together” (CAS Barefoot, 2014b). While computational thinking doesn’t necessitate working with someone else every minute, there are many opportunities for collaboration throughout the CT process, and taking advantage of these opportunities can help produce a better solution. For example, during decomposition, a problem might not only be broken into its component parts, but those component parts might also be assigned out to different members of a team. People designing and debugging algorithms might improve their design or get out of a rut by bouncing ideas off someone else (CAS Barefoot, 2014a). And evaluations of computational solutions almost always need to involve others.

What Computational Thinking Is Not

In addition to understanding what computational thinking is, there are at least a couple common misconceptions of CT that you need to understand as well:

- Computational thinking is not thinking like a computer; it is not something computers do. Rather, it is something that humans do in order to leverage the power of computers. Thinking like a computer would be more accurately described by the term "machine learning," which is associated with advances in artificial intelligence, or the ability of computers to change their behavior without being explicitly programmed to do so.
- Computational thinking is not the same as coding. CT does have its origins in coding; it is essentially the observed process of computer programmers. Coding itself, however, is essentially one form of *algorithm design* (a part of CT, but not the whole). Learning and teaching code has its own merits, and it is certainly a great way to help students learn to think computationally. But there are also ways to practice computational thinking--and even create algorithms--that do not involve extensive knowledge of programming languages. Thus, while all coders use CT, not everyone who uses CT needs to become an expert coder.
- Computational thinking is not the only way of solving a problem. Other approaches (such as, for example, the scientific method or design thinking) are equally viable ways of approaching a problem. Some problems are particularly suited to a primarily CT approach; some problems may be better served by a primary focus on a different problem-solving method, but may still benefit from thinking about the problem computationally as well; and some problems may not be suited for computational solutions at all. There will be more on this issue later.
- Teaching computational thinking is not the same as teaching technological literacy or digital citizenship. Whereas technological literacy or citizenship involves wise and responsible use of technology, social networks, and all forms of media, computational thinking is more concerned about how we leverage these technologies to solve problems. Both are necessary and important topics for students, but they are also distinct, and one should not be conflated with the other.

Requirement 2: Guided Computational Thinking

Note: Your submission form contains a guide that will help you through all sub-steps of this requirement. It also contains hints that will likely prove useful to you.

Task

Use computational thinking to analyze the problem described below, create an algorithm that helps solve the problem, and then evaluate how well your solution works.

Instructions

1. *Read the problem description carefully.*
2. *Use the spaces provided on the submission form to type thorough, thoughtful responses to each question posed.*

Problem Statement

As an educator in the 21st Century, you recognize that what you know is less important than your ability to learn and retain information. What is current, accepted, state-of-the-art or best practice in both pedagogy and your subject area may be quite different next year than they are today. As a conscientious educator, you feel that it is vital to stay abreast of new developments in your field.

You know that the Internet can be a great way to stay in touch with these developments, but it is difficult to find the time to sift through the web of available information to find the golden nuggets you're looking for. In the bustle of teaching, it can be difficult even to remember to try. And when you do find the time, it is easy to get sidetracked by information that does not matter.

Questions in Sub-Requirements

1. *What are the components of this problem?*
2. *What do you anticipate being the component parts of the solution?*
3. *What repeating patterns do you see in the information provided or the data gathered?*
4. *Write, draw, or otherwise represent an abstraction of this problem.*
5. *What automated systems might be useful in solving this problem?*
6. *Write ordered steps that would enable a human, a computer, or (more likely) some combination of the two to carry out the solution. Indicate the agent (who or what) that will carry out each step.*

7. *List each problem component you identified in step 1 and evaluate whether your solution addresses that problem or not. Also, identify any future issues (bugs) you foresee that may need be resolved in the algorithm to make it better.*
8. *How does the algorithm above leverage the power of computers and automation? In what ways might it leverage these capacities more?*

This requirement is designed to guide you through a CT problem in a scaffolded way and prepare you to better formulate and solve CT problems on your own. The submission form contains valuable hints that will nudge you in a certain direction to make sure that you are solving the problem in a computational way. Before you complete this requirement, however, it may be helpful to see a completed model of a similar problem.

(As a teacher, it may interest to you that the teaching strategy employed in this progression is the gradual release of responsibility model, which is a research-based best practice for integrating computational thinking (and a great many other things) into your classroom [Hunsaker, n.d.]).

Sample CT Problem: A Model

This problem is not the same one you see in requirement 2, but you may find it useful to have a complete model to refer to. The form you see below *is* very similar to the form you will be using on your submission in requirement 2. Think of the information in each entry field as what a teacher might say to his or her class while modeling CT problem solving for students.

Solving a CT Problem: A Model *This Model is courtesy of Peter Rich, a professor of Instructional Psychology & Technology at Brigham Young University, who specializes in Computational Thinking.*
Task Use computational thinking to analyze the problem described below, create an algorithm that helps solve the problem, and then evaluate how well your solution works. **Instructions** 1. Read the problem description carefully. 2. Use the spaces below to type thorough, thoughtful responses to each question posed.

Problem Statement

As a teacher, you send out a newsletter to students' parents each week with class updates—goals, challenges, successes, homework, etc. You love to personalize these letters so that parents can see what and how their own child is doing in the classroom, and so that you can issue a personal challenge based on each student's individual weekly goals. Parents love it, too, and have expressed how much they appreciate the personal touch and feedback. The problem is that it takes so much time to create a personalized letter for 30 students each week. If you don't find a better solution, you may have to start sending out a more generic letter.

Decomposition:
Breaking down data, processes, or problems into smaller, manageable parts

1. What are the component parts of this problem?

You look at the problem and realize there are several parts to this:
1. collecting individual student data
2. updating the newsletter with class-level info,
3. customizing specific parts of the letter, and
4. sending each letter out to the correct parent(s)/guardians.

Pattern Recognition
Observing patterns, trends, and regularities in data

2. What repeating patterns do you see in the information provided or the data gathered?

There are specific parts to the letter that you realize repeat. For example, each week, you report on goals, challenges, and successes for individuals as well as the class. You also report on upcoming homework, which is the same for everyone.

Abstraction
Creating a visual model or simulation of a problem that incorporates only the most important details.

3. Write, draw, or otherwise represent an abstraction of this problem.

Even though your letters are personalized, they follow a specific formula. In fact, your letter is replete with variables where you simply replace one students' name, parents, goals, etc. with another's. You realize your entire newsletter looks something like this:

Dear {{Parent Title}} {{Parent Last Name}},

This week in our 8th Grade English class, we learned about.... Our learning goal was to .... This was difficult because .... Ultimately, though, we ...

Some of the challenges {{Student Name}} faced in particular were {{Description of

Student Challenges}}. {{Student Name}} was able to overcome these challenges and meet {{Student Possessive Pronoun}} goal of {{Last Week Student Goal}}. In particular, {{Student Name}} was able to {{Description of Student Success/Achievement}}

For this coming week, {{Student Name}}'s new goal is to {{This Week Student Goal}}. This is a great goal that we agreed on together, and I would encourage you to support {{Student Direct Object Pronoun}} as you see fit.

Next week, we will be learning about ... I have asked the student to come prepared by ....

Thank you for all you do as a parent. I enjoy having {{Student Name}} in my class!

Sincerely,
Mr. H.

Algorithm Design
Developing the step by step instructions for solving this and similar problems

4. What automated systems might be useful in solving this problem?

You may not be a tech pro, but you remember in your awesome teaching with tech course at BYU that you learned about spreadsheets and mail merges. A mail merge allows you to craft a message and to replace certain variables with information stored in a google sheet. You can't remember how to do the merge off-hand, but you Google it and find the "**Mail Merge with Attachments**" free add-on. You use this to create a personalized letter that, when you export, will automatically be sent to the address you have in your spreadsheet for each parent.

5. Write ordered steps that would enable a human, a computer, or (more likely) some combination of the two to carry out the solution. Indicate the agent (who or what) that will carry out each step.

#	*Step Description*	*Agent*
1	Install the Mail Merge with Attachments add-on to Google Sheets. (See instructions)	Teacher (1 time)
2	Create a mail merge template, which is a Google spreadsheet that will feed into the mail merge. The template should have the following columns: ● Parent Title (Mr./Mrs./etc.) ● Parent Last Name	Teacher (1 time)

	● Parent Email Address ● Student Name ● Student Possessive Pronoun ● Student Direct Object Pronoun ● Description of Student Challenges ● Last Week Student Goal ● Description of Student Success/Achievement ● This Week Student Goal	
3	Populate the following columns on each spreadsheet row with corresponding data from class rolls: ● Parent Title (Mr./Mrs./etc.) ● Parent Last Name ● Parent Email Address ● Student Name ● Student Possessive Pronoun ● Student Direct Object Pronoun	Teacher (1 time)
4	Copy and paste the Sample email in the Abstraction field above into a Gmail Draft.	Teacher (1 time)
5	Interview students to collect data for the following columns and input that data into the mail merge template spreadsheet: ● Description of Student Challenges ● Last Week Student Goal ● Description of Student Success/Achievement ● This Week Student Goal	Teacher (weekly)
6	Configure the mail merge. Choose the draft copied into Gmail in step 4. Update the subject line and first and second-to-last paragraphs with the information relevant for the week.	Teacher (weekly)
7	Push “Run Mail Merge”	Teacher (weekly)
8	Send a personalized email newsletter to each parent.	Mail Merge add-on

Evaluation
Ensuring that your solution is a good one.

6. List each problem component you identified in step 1 and evaluate whether your solution addresses that problem or not. Also, identify any future issues (bugs) you foresee that may need to be resolved in the algorithm to make it better.

Problem Component	*Evaluation/Possible Bugs*
Collecting Individual Student Data	This component is met in step 5, where the teacher gathers student information.
Updating the newsletter with class-level info	This component is met in step 6, where the teacher configures the mail merge.
Customizing specific parts of the letter	This component is completed in steps 4 and 5 with the newsletter draft and when the teacher adds the data into the spreadsheet. There may be some bugs that need to be worked out here, such as making sure that the static wording in the letter and the customized wording from the mail merge meld into each other well for a seamless letter.
Sending each letter out to the correct parent(s)/guardians	This component is completed in step 6 by the Mail Merge add-on. There may be some debugging that needs to happen if parents change email addresses, students drop the class, etc.

7. How does the algorithm above leverage the power of computers and automation? In what ways might it leverage these capacities more?

After sending out the letters, you wonder why you didn't approach things computationally earlier. You now have a more personalized letter and it takes 1/3 the time (or less) than it used to. What's more, automating this solution has given you more time to work on the interpersonal aspects of teaching. You realize you can take it a step further, though; you've been entering students' goals into the spreadsheet from your personal discussions with them. Instead, you realize that you can send a google form to students, and automatically merge their answers from the spreadsheet that reports the form data. So, each week on Thursday, you now have students write a brief reflection (reporting their email and name, so they can get credit and you can connect it to your data). You merge this data with the rest of your spreadsheet data and create an even more customized letter without increasing your overhead. You may even have time to catch a show this weekend!

Answering Questions in Sub-Requirements 2.1-2.5

The questions in these sub-requirements are intended to guide you down a path that will help you to see a computational solution for the problem posed. **Pay special attention**

to the notes in italics after each question on the submission form. They should be especially useful.

Requirement 3: Open Computational Thinking Project

Using the work you did in requirement 1.2 as a guide, complete the following steps.

3.1 Formulate
On your submission form, describe a real-world educational problem you are facing. It can be local (e.g., I need a better way to communicate with parents), global (e.g., standardized tests do not accurately assess some important 21st-Century Skills.), or anything in between. Formulate the problem in a way that facilitates the leveraging of computational methods to solve it. Determine what information you need, and gather what is necessary to inform your problem-solving process.

3.2 Solve
Use the skills of decomposition, pattern recognition, abstraction, algorithm design, and evaluation to produce an automated set of steps that addresses (or at least partially addresses) the problem you formulated. Record your algorithm on your submission form.

3.3 Report
On your submission form, write a brief report in which you document how you utilized CT skills (i.e., decomposition, pattern recognition, abstraction, algorithm design, evaluation) to solve your problem. Include any relevant artifacts (for example, the image of an abstraction you created in your problem-solving process)

Requirement 3.1: Formulating a CT Problem

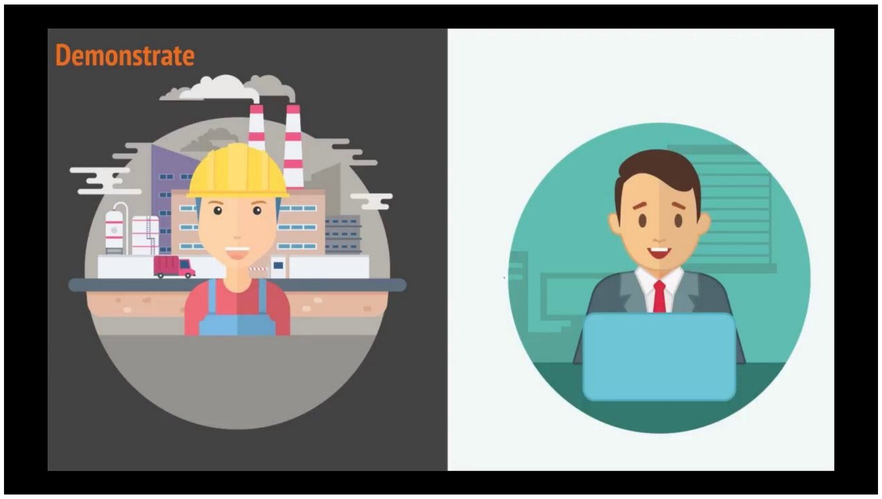
When formulating a problem to solve with computational thinking, whether for yourself or for your students, it is important to follow some guiding principles:

Choosing a Suitable Problem

CT is not the only way of approaching problems, and it is not always the best either. Generally speaking, CT is very well-suited for problems that involve repetitive, predictable tasks. Computers can usually perform these tasks much more quickly, efficiently, and accurately than can humans.

That said, don't take everything at face value. There are many problems that may not seem suited for computational methods at the outset, but for which a computational thinking may yield at least a partial solution. Looking at problems consistently through only one problem-solving lens can lead to blind spots. For example, can you imagine trying to solve the mail-merge problem (see Sample CT Problem: A Model section above) with the scientific method? You could probably approach the problem that way, but you would end up with a very different kind of solution, and it probably wouldn't be as effective. Similarly, CT might not be the best approach to take if you're trying to decide how to help your students remain engaged during class time; although, computational thinking might yield you a partial solution that would free up time for you to focus on this highly interpersonal aspect of teaching.

In thinking about different types of problems, problem-solving methods, and how to identify a problem that is well-suited for CT, the following video may prove useful:



Video 6. What is Computational Thinking? Click, copy, or type this link into your browser to view: https://youtu.be/JbpZDu4Mjss. ***Note:*** *This video is copyrighted by McKay Perkins under a Standard YouTube Licence and is therefore not included in the CC-BY license of the rest of this document.*

Framing the Problem Computationally

Sometimes how we word or frame a problem is just as important as what the problem is. The ISTE/CSTA Operational Definition (2011) suggests this idea as the first characteristic of CT: "Formulating problems in a way that enables us to use a computer and other tools to help solve them."

One thing to consider is the idea of where you "slice the pie" so to speak. Every problem has a lot of aspects and complexity to it. A classroom, for example, is a complex system, and every part of how you run it affects every other part. The Mail Merge example (see above) is really only a portion of the overall system of the classroom. CT can't be applied to every aspect of classroom management, but it does provide a really good solution for part of it. If the problem had been formulated differently to include other aspects of classroom management, the solution would likely have been different, or maybe not even a computational solution at all. This is one reason why effective decomposition is so vital to computational thinking.

Ideas for Classroom Computational Problems

If you're stumped about where to begin formulating your open CT problem, consider the following ideas. These aren't fully-formulated, but they can get you started on the right foot:

- Trying to understand student performance trends.
- Item analysis on a recent test (use CT to create an algorithm to automate the reporting of each item, identify problematic items, and improve the test).
- Collecting and displaying examples of student work over time.
- Regularly gathering, filtering, and sorting current events related to specific units so that your materials are relevant and up-to-date.
- Documenting your preschool students' work across the year.
- Writing reminder emails ahead of time that they can be sent out to help students with upcoming assignments and tests (see Google Boomerang add-on).
- Making randomized group assignments to put students into teams, but making sure that everyone gets to work with everyone else at least once throughout the year.
- Providing students with practice tests/quizzes that pull from a bank of items so that the quiz is dynamically generated each time.
- Setting up a classroom point system that will alert you and the students where the class is at and what is lacking in order to reach the goal.
- Keeping your online materials synced with your personal computer (Think cloud storage solutions).
- Collecting, analyzing, and displaying the results of a weekly survey so that you can do "just-in-time" teaching and know what to focus on in class that day.
- Many things that can be solved using IF This Then That (IFTTT) software. Check it out at http://www.appsinclass.com/ifttt.html

Requirement 3.2: Solving a CT Problem

What you're ultimately trying to produce with the process of CT is a solid, functioning algorithm that can give a computer instructions for automating a task. For this requirement, you need only record your solution--that is, your algorithm. You will have a chance to comment on it in the next requirement.

One important thing to keep in mind as you complete this process is the difference between an algorithm and a heuristic. There is a lot of overlap, and people without a lot of exposure to computer science may naturally end up producing a heuristic rather than an algorithm as they complete the CT process. You won't necessarily be marked down if your solution contains some characteristics of a heuristic, but try to keep your solution as close to a true algorithm as possible. Read this short article to understand the difference between the two.

Requirement 3.3: Reporting your CT Process

For this requirement, you need to document how you used the CT process to approach your problem and produce and evaluate your algorithm. This may consist of copying and pasting the same form you used in requirement 2 and filling it out for requirement 3. You may also choose to simply write a few sentences about each of the 5 CT skills emphasized in the badge: decomposition, pattern recognition, abstraction, algorithm design, and evaluation. If you feel it would help your badge reviewer better grasp your CT process and skill, you may also choose to include artifacts you produced along the way, such as your abstraction, or preliminary algorithms you produced, but then adjusted or discarded during your evaluation phase. How you present your work is up to you; just make sure that it demonstrates your grasp of what CT is and how you can use it to solve everyday problems.

Requirement 4: Reflections

4.1: Attitudes & Approaches Reflection
Write 1-2 paragraphs in which you respond thoroughly to each of the following prompts:

- *Write about at least three (3) attitudes (e.g., confidence, communicativeness, flexibility) and/or approaches (e.g., tinkering, creating, debugging, persevering, collaborating) that you utilized (or should have utilized) during your CT Projects (requirements 1.2-1.3 of this badge).*
- *How have CT skills, attitudes, and approaches played a role in your problem-solving habits and processes of the past?*
- *Which CT skills, attitudes, or approaches would you like to more fully integrate into your future problem-solving habits and processes?*

4.2: Computational Thinking Rationale
Write 2-3 paragraphs in which you respond to the following prompts:

- *Why do many feel that computational thinking (CT) should be considered a fundamental literacy of the 21st Century? (Cite 1-2 sources).*
- *In your opinion: How will current attitudes and trends in regard to CT affect education in general as well as your future teaching career specifically?*
- *In your opinion: As an educator, what role (if any) do you have in teaching/supporting CT in the classroom?*

Requirement 4.1: Computational Thinking Attitudes & Approaches

If you need help understanding the attitudes and approaches listed in this step, refer to the Components of Computational Thinking section above.

Requirement 4.2: Computational Thinking Rationale

Computational Thinking as a 21st Century Literacy

To answer this question, you may refer to any scholarly article, blog article, video or other digital or non-digital resource you wish. You are encouraged to explore a variety of opinions on this matter, but you can also complete this requirement by simply referring to the resources in the What Is Computational Thinking section above.

Attitudes and Trends in regard to Computational Thinking

To answer this question, you are encouraged to do your own online (or offline) research, but you may also simply consider the following information, taken from the introduction of a chapter on Computational Thinking in Royce Kimmons' K-12 Technology Integration open textbook on Pressbooks (Hunsaker, n.d.).

Why Integrate Computational Thinking *Direct Quote from Hunsaker, E. (n.d.), which is under CC-BY-SA license*
More than ever, we live in a world that is informed and inundated by computer technology. This fact may conjure thoughts of smartphones and personal computers, but increasingly, many everyday and traditionally non-digital objects are being designed to operate via a computer program. Some of these objects include streetlights, car engines, watches, roads, car tires, shoes, and even cereal boxes (Hartigan, 2013). As computer programs become more widespread, computer programming becomes an increasingly relevant skill, and many political bodies are recognizing this fact. Support for teaching computing in K-12 schools is growing in the U.S. and abroad. Several countries, including England, Finland, South Korea, and Australia, require that children learn computing or computational thinking (Rich, Jones, Belikov, Yoshikawa, and Perkins, 2017). Several U.S. states and districts have similar requirements (Partovi, 2017; EdSurge, 2016). The United States has not yet officially adopted such measures, but appears to be moving in that direction. For example, in 2017 the Trump administration announced a yearly investment of $200 million dollars into STEM education, noting that "the nature of our workforce has increasingly shifted to jobs requiring a different skill set, specifically in coding and computer science" (CNN Wire, 2017, emphasis added). Amazon, Facebook, and other major tech companies have committed a sum of over $300 million (over the period of five years) to the new initiative (Romm, 2017). Thus, increasing attention, interest, and enthusiasm are paid to the role that computer science education should have in our schools (Bers, Flannery, Kazakoff, and Sullivan, 2014; Rich et al., 2017; Sullivan and Bers, 2016; Yadav et al., 2016; Yadav et al., 2017). But before computer programming—or coding, as it is sometimes called—many believe that today's youth (and adults) need computational thinking (CT) to better solve the problems of the 21st century. CT may be considered a precursor to learning actual coding or computer programming skills. And while this is certainly true, it can also have a much broader application. The skills, attitudes, and approaches that make up CT are fundamental, universal, transferrable, and particularly appropriate and useful for the computer age. So, while a future computer programmer certainly needs CT, it is not necessarily true that everyone who learns CT should go on to learn coding. Rather, as computer technology becomes more embedded into the fabric of

every industry, professionals in every industry need to be able to think in ways that leverage those computers to solve the problems of the future.

Learning computational thinking can benefit students both economically and academically. Each year there are far more computing jobs added than there are computer science graduates, with significant job growth projected for the foreseeable future (Bureau of Labor Statistics, 2018). Furthermore, studies have linked a host of academic benefits to learning CT, including improvement in student engagement, motivation, confidence, problem-solving, communication, and STEM learning and performance (Rich et al., 2017; Yadav et al., 2017).

References

CAS Barefoot (2014a). *Collaborating.* Retrieved from https://barefootcas.org.uk/barefoot-primary-computing-resources/computational-thinking-approaches/collaborating/

CAS Barefoot (2014b). *Computational thinking.* Retrieved from https://barefootcas.org.uk/barefoot-primary-computing-resources/concepts/computational-thinking

CAS Barefoot (2014c) *Debugging.* Retrieved from https://barefootcas.org.uk/barefoot-primary-computing-resources/computational-thinking-approaches/debugging/

CAS Barefoot (2014d) *Evaluation.* Retrieved from https://barefootcas.org.uk/barefoot-primary-computing-resources/concepts/evaluation/

CAS Barefoot (2014e) *Persevering.* Retrieved from https://barefootcas.org.uk/barefoot-primary-computing-resources/computational-thinking-approaches/persevering/

CAS Barefoot (2014f) *Tinkering.* Retrieved from https://barefootcas.org.uk/barefoot-primary-computing-resources/computational-thinking-approaches/tinkering/

Hunsaker, E. (n.d.). Integrating computational thinking. In R. Kimmons (Ed.) *K-12 technology integration.* Pressbooks. Retrieved from https://k12techintegration.pressbooks.com/chapter/integrating-computational-thinking/

ISTE, & CSTA. (2011). *Operational definition of computational thinking for K-12 education.* Retrieved from http://www.iste.org/docs/ct-documents/computational-thinking-operational-definition-flyer.pdf

ISTE (2014, September 11). *Computational thinking for all.* Retrieved from https://www.iste.org/explore/articledetail?articleid=152

Google (n.d.b). *What is computational thinking?* Retrieved from https://computationalthinkingcourse.withgoogle.com/unit?lesson=8&unit=1

Standl, B. (2017). Solving everyday challenges in a computational way of thinking. In V. Dagien & A. Hellas (Eds.), *Informatics in schools: Focus on learning programming* (pp. 180–191). Cham: Springer International Publishing.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35. https://doi.org/10.1145/1118178.1118215