How coding helps with success in the learning of mathematics

Written by Laura Gini-Newman and Peter Kuperman

How helpful is coding in achieving success in mathematics?

To be a competent mathematical thinker, a person needs to possess a broad range of abilities to think, act and communicate to effectively solve challenging, often unfamiliar, problematic tasks across a range of contexts, situations and purposes involving mathematics (PISA definition of mathematical literacy). Although the usefulness of mathematics is more obvious to many when a person is required to act or communicate mathematically, thinking is at the core of all mathematical competencies and success.

Since a person cannot think, act or communicate effectively without possessing a strong ability to think critically, both mathematical and coding success are highly dependent on the development of these thinking competencies. A synthesis of the mathematical skills or competencies identified by educational ministries and post-secondary educational institutions across Canada, in the American Common Core Standards of Practice, as well in Asia, Australia and the United Kingdom suggests 10 key thinking competencies are needed to proficiently think, act and communicate mathematically. Interestingly, these same competencies are applied in the practice of coding. As such, learning to code helps students learn how to be success in mathematics and vice versa.

The following table clarifies the nature of each of the 10 key mathematical thinking competencies and illustrates the role each plays in mathematics and coding allowing us to understand the degree to which learning to code and learning mathematics are related. It is clear from the details in this table that coding and mathematics require the same types of thinking or reasoning to achieve success.

Thinking Competency: **Sound reasoning**	**Example of coding through** **sound reasoning**	**How sound reasoning in coding helps with math**
Sound reasoning refers to the ability to think about the quality of one's reasoning to ensure that it is sound or reasonable This is an umbrella competency that affects the quality of all other thinking competencies related to thinking, acting and communicating. Reasoning is sound when decisions are made following the assessment of plausible options in light of criteria based on understanding and evidence. These decisions typically involve on-going sense making and reflection to promote the use of the most productive ways to manage problems and learn from mistakes while developing fluency and automaticity.	Let's say you want to create a maze program. Before you start to code, you will think about the parameters and how to best pull the project together. You consider: What is the best size for the maze? What is the best way to organize the code? What algorithm are needed to create the maze? Will the maze have many dead ends or only a few? What problems might be encountered the code is being created and how will the code effectively deal with these problems. In coding, sound reasoning is the on-going process of predicting or imagining what the end result will be and strategically creating the	Learning how to code teaches you how to manage projects and problems. It encourages effective decision making related to planning; to the strategic selection of algorithms and their logical order to create a general plan that is then continuously tested for soundness through a thoughtful implementation process. Coding teaches students how to make a number of decisions important to the study of mathematics in this process: • to know and check that what they are doing as they solve math problems is reasonable and makes sense; • to take the time to be reflective so that when they solve problems, they recognize

Every aspect of mathematics and coding demands sound reasoning. **Mathematicians** reason to develop conceptual understanding, to communicate, structure and represent ideas effectively, to strategically manage problems, to know what tools to use and to what degree concepts and ideas are related. **Coders** reason to help understand new coding languages, to plan strategically, starting from the first line of code, the best way to fulfill the objective of the program to be designed. They reason soundly when they anticipate problem areas and think about how to manage problems and mistakes even as they are being made (debugging).	code the produces what is imagined, constantly selecting, adjusting, replacing, and checking for the effectiveness of its building blocks to best meet the coding objective in the easiest, most-straightforward manner.	weaknesses and errors in their approach, and are able to find ways to strengthen or correct them; • to take the time to come how up with a number of good ideas to solve problems then triage these ideas to determine the best one to use.

Thinking Competency: Conceptual reasoning	**Example of coding using conceptual reasoning**	**How conceptual reasoning in coding helps with math**
Conceptual reasoning refers to the ability to independently construct understandings of what defines concepts or ideas, be it in **mathematics or coding,** and to identify and use these understandings to effectively understand problems and how to solve and communicate solutions to these problems. In **mathematics**, all ideas, theorems, and principles are concepts that can be defined using a set of properties; characteristics that hold true for all examples of a concept. In mathematics, all symbols represent concepts. To be able to communicate mathematically, concepts must be	Every coding project pushes the boundaries of what needs to be known in terms of concepts. Take learning how to draw a checkerboard for example. This leads to a thorough understanding of a for loop (repeat 8 times across and 8 times down) and of an if statement (if…then some of the squares are black, or some white). By having students experiment or explore what properties define all loops in coding, students learn about the structure, its meaning, and the different ways it might look. To understand the concept of a loop in coding, students consider	Learning how to develop an understanding of concepts in code teaches students how to independently understand mathematical concepts by teaching a way of thinking that: • creates understanding by exploring a number of examples of a concept to determine what properties define that concept; for example, a polygon is a closed geometric figure; with linear sides, number of internal angles equal to the number of sides and angle size dependent on the number of sides; • helps students to generalize patterns by thinking inductively to seek mathematical truths; for example, a square is a closed

thoroughly understood. Concepts in mathematics include: equality (equal sign); the operations of addition, subtraction, multiplication and division (+, -, x. /), fractions, exponents, rates, functions, a limit or integral…to name of few. Consider another example: understanding that the word 'integral' and that the symbol for integral both mean the area under a curve allows people to convey more information with fewer words helping to make communication more efficient and effective, and universally understandable. In **coding**, the same hold true. To code effectively requires an understanding of a for loop, an if statement, or an object…to name a few. Without this understanding students cannot use the symbolic language of code effectively and independently.	a wide range of different examples of loops such as while, for, do loops in order to generalize what all loops have in common. In computer programming, all loops contain: • a sequence of instructions; • a sequence that is repeated continuously; and • repetition that ends when a certain condition is met. This also allows students to clearly differentiate different loops by considering to what degree a particular type of loop meets these properties. Take an infinite loop for example as a tester. This loop lacks a functioning exit routine. The result is that the first 2 properties are met (the loop repeats continually) however it does so until the operating system either terminates the program with an error or until some other event occurs (such as having the program automatically terminate after a certain duration of time) rather than a condition being met. A loop is a fundamental concept in coding; without it effectively creating and understanding code is virtually impossible.	figure that has 4 sides, 4 vertices, 4 ninety degree angles; and sides of equal length; • helps students understand that symbols communicate a specific generally accepted meaning in code; for example a ";" means the end of a statement and that the instruction is complete while in math an equal sign means the value represented on either side is the same; and • reasoning inductively, when proving in math, can also be seen as infinite loops of reasoning. Internalizing the concept of loop through coding supports the understanding of the deeper, yet related, concept of mathematical induction.

Thinking Competency: Representational reasoning	**Example of coding using representational reasoning**	**How representational reasoning in coding helps with math…**
Representational reasoning refers to the ability to effectively communicate (understand or interpret and visualize or represent) ideas and concepts across a variety of forms. This is one component of the broader thinking competency of communication.	Learning how to build a computer card game requires a coder to think about how they are going to represent the 52 cards in a deck of cards. They could use numbers and letters, e.g. 9H for the 9 of hearts; a simple graphic, or an animated graphic or any other set of symbols intended to communicate this idea. In laying	Representing ideas in many different ways is a fundamental competency in coding and can only be achieved if the coder reasons effectively about which representations are most appropriate given their purpose and the needs of the user, and how all effectively capture the idea the coder wishes to communicate. This competency is equally

In **mathematics**, this might include words, graphs, symbols, tables, pictures, visuals, algebra. For example, the idea of equality can be shown using pictures, sets, geometric figures, numbers and symbols. Variance can be shown using a bar graph, bell curve, symbol or number. In **coding**, representational reasoning is all about building good user interfaces making it easy for the user to see what a computer program is designed to represent. This is basically a way of creating a good communication channel between the user and the person or team who designed the computer program or software application. With strong representational reasoning skills, this communication channel will be robust and easy to understand.	out a hand of cards on the screen, you could place them side by side, overlap them, show only half the card, or just enough to identify the card. A coder also has to decide, of the 52 cards, which ones will be in the player's hand. This information will come from a database, in which the cards will be represented by numbers and symbols that will also have to connect to how the cards will be visually displayed. They will decide how to sort the cards in the player's hand properly, e.g. will you group them by number or by the four suits. All of these decisions require representational reasoning so that all the representations communicated in the code show the same ideas accurately and in the most appropriate way to effective meet the purpose of the code. Needing to make all these choices when designing a computing project requires a student to reason about how to represent ideas to build a user interface.	important in mathematics. Developing this reasoning capacity in coding classes helps students be more successful in mathematics as students become better able to: • show mathematical ideas in many different ways effectively; • select the most appropriate representations to help them solve a problem at various stages in the problem-managing process (e.g. how to better understand what the problem is; how to best communicate a solution to a problem given the context and user of the solution) • better understand how various mathematical ideas are related or connected (e.g. seeing visual representations of addition and subtraction allows a young child to understand how they are both similar but different in important ways; seeing different functions graphically, in table form and algebraically helps older students understand how various function are both similar and different in important ways)

Thinking Competency: **Structural Reasoning**	**Example of coding using** **structural reasoning**	**How structural reasoning in coding helps with math…**
Structural reasoning refers to the ability to understand and use structure and form to effectively communicate ideas. Structural reasoning ensures not only knowledge but understanding of the generally accepted conventions of the	Learning how to sequence the steps of a solution with the fewest steps possible in an order that logically develops the solution without including redundant ideas is another fundamental competency of computer science.	Being able to reason structurally involves the ability to communicate effectively using logically sound structures and forms that tend to be, for the most part, generally accepted language conventions. Learning to code helps students

language of mathematics or computer programming so that it can be read and written by anyone in the field. This form of reasoning is another component of the broader competency of communication. In **mathematics**, this includes an understanding and use of the rules of logic in both an informal or formal sense; including inductive and deductive logical reasoning to arrive at generalizable structures and forms and to identify, interpret and create specific examples of these generalizations. Take for example the form and structure of an equation using a variety of operations. BEDMAS is a generalized convention that allows an equation to effectively communicate its meaning. In **coding,** generalized structures and forms that are the conventions of a computer language are needed to build computing projects that derive solutions in a logical manner. For example, being able to understand that a loop is a structure that allows the same steps repeated multiple times in the same way allows the coder to communicate these steps more effectively in a manner understood by all computer scientists. Reorganizing a set of steps (called refactored in computer science) allows the same end result to be delivered in a more efficient and logically sound way. All coding conventions, e.g. tabbing, spaces, indentations, parentheses, brackets formatting make the code readable.	An example of this in coding is when ordering a list of names alphabetically, there are several sorting algorithms. The most basic one (albeit not the most efficient) involves comparing pairs of names one by one, deciding which one comes first, and setting them into the correct place in an array. These are the logical sequence of steps taken to develop the solution to the problem of ordering alphabetically without including redundant information or steps. These steps serve as a map for the writing of the specific code that will perform the task of sorting. Another example of how structural reasoning in coding works is trying to locate an address on a map after it has been typed in. There are at least 4 parts to an address: a) the street number, b) the street name, c) the city, and d) the Postal Code or Zip code. There are also the province or state and the country, but let's consider the first four in this example. Once an address has been entered into a software application, the problem to be solved is to find the correct location of that address on a map; a map which contains hundreds of millions of addresses. The solution is to narrow down all the addresses in 4 stages until we get to the unique address that was originally entered. Is it more efficient and logical for the 1st step to be "All the addresses with street number X"; "All the addresses with street name Y"; "All address in a specific city"; or "All addresses with a specific postal code"? What should logically go next step? Figuring out the answers to these questions involves structural reasoning.	develop the capacity to reason structurally in mathematics so that they can: • know how to appropriately use the structure and form of the language of mathematics to effectively and universally communicate mathematical ideas, problems and solutions (simplified form of an equation using BEDMAS) • understand why and how a structure has become a generalized way to communicate (reason inductively to understand why and how mathematical structures and forms have been created and eventually how to think through the development of new innovation structures of their own; for example, does it matter where to place an equal sign in an equation and if so why?) • understand and effectively apply logic to describe and solve mathematical problems and present solutions effectively (in the simplest, most justifiable way without the inclusion of unnecessary or redundant evidence) Learning to code allows students to understand the importance of structure and form, of logical sequences, and of presenting information in a proper order, using appropriate conventions to format ideas in the most effective manner that will be accessible to all its users. How students reason structurally in coding is also how they reason structurally in math. In both math and coding structural reasoning also helps the limited capacity of the human brain to deal with limitless amounts of complex information in an effective manner. The complexity of rich, meaningful problems students

		are invited to solve is most often far beyond what even the highest trained mind can contemplate at once. The way both coders and mathematicians tackle this limitation is through structural reasoning.

Thinking Competency: **Detail-minded Reasoning**	**Example of coding using detail-minded reasoning**	**How detail-minded reasoning in coding helps with math…**
Detail-minded reasoning refers to the ability to pay close attention to details when completing a task to fully understand the complexity of the task and to respond with accuracy and precision. In **mathematics**, this refers to the ability to consider every relevant detail in a problem no matter how small or insignificant it may first appear, and to consider how it may impact on the arrival of a solution to a mathematical task or problem. For example, when asked to find the total number of oranges across a set of 5 baskets, all fruit must be considered to determine which are oranges in order to arrive at the most accurate count of oranges. Or, if a student is trying to determine their final mark on a test given in decimal form to 4 decimal places, the student must consider all numbers following the decimal to decide which numbers are relevant to determine their mark as a percentage whole number. In **coding**, a programmer must pay close attention to the details of the code to clarify whether the code will effectively meet the	Often in computer science being able to count properly in abstract situations means the difference between a program that works and one that falls apart. Take for example, if you have a field that is 100 meters wide and you want to put fencing along the entire length with fence posts every 10 meters, how many fence posts do you need? The answer is 11 – you need one for every 10 meters plus one more at the very beginning or else the first section of fencing will fall down. This is known as the fence-post problem, which illustrates the “off by one” error that occurs often in computer science. This problem illustrates the importance of detail-minded reasoning in computer science. When building a video game, figuring out when two objects collide, or if when a hero jumps from one platform to another, if he successfully lands on the new platform, are important. Paying attention to elements that may initially seem trivial and unimportant may have a significant impact on the outcome of a program. In this case, the coder must decide the degree of error he should allow because visually, players will be upset if it looks like	Learning to think about all details or possible outcomes when coding ensures a coder creates a well-functioning successful program. The ability to do so is also of fundamental importance in mathematics. When students reason in this way in coding they transfer this ability to math so that they: • understand all the potential complexities of a math problem before attempting to solve it; • ensure they derive solutions that are ***accurate and precise*** rather than arriving at incomplete or inaccurate conclusions/solutions prematurely; • generate a ***fulsome*** set of mathematical ideas to consider when solving a problem; • learn to persevere.

purpose the program and to make sure it does so in all situations, including odd-ball situations where a creative or modified approach to the solution is required such as with Edge Cases – cases that are non-standard like being at the very edge of a screen, having a price less than zero when applying a refund, or having a four point play in basketball (foul plus a three pointer). It is also needed when dealing with unusual situations in which there are many permutations at play.

Often in computer software development a non-technical person asks for a new feature in the program, for example, I want all employees to get a notice when their paycheck has been deposited. In a typical case if each employee's bank information is correct, everything runs smoothly. The job of a good software engineer is to think about all the ways this feature can break down. Let's say the bank has locked or frozen the employee's account – the bank information is correct but an odd-ball (edge case) situation has occurred. What should the software do? Being detail-minded means that before the employee gets mad because the software doesn't work properly, the engineers have actually accounted for all plausible unusual situations so the employee gets an appropriate message given the situation. In essence, thinking about all the details needed to solve the problem allows a coder to handle the nuances and complexities of all possible situations, not just the simple or obvious standard situations.

they made the jump but the code says they didn't.
The same issue exists when a video game is trying to figure out if a hero and an enemy or a hero's bullet and an enemy, have collided. If the collision does or does not happen leads ultimately leads to the player being either rewarded or punished. If it looks like something happened one way, but the software records it a different way, a very significant "off by one" type error occurs. Minimizing or eliminating the difference between the code and the visual experience by the user involves detailed-minded thinking.

Computers are the most ruthless teachers of detailed-minded thinking. A comma out of place can cause the entire program to break down. Even if a program seems to run smoothly, it may produce useless or misleading results if the slightest of details is overlooked in its design. Only when every single possibility is carefully accounted for in the code will the machine produce the desired results.

These are but a few examples that illustrate the way in which good coders think about all problems; they consider all the details, nuances or unusual situations in the problem before beginning to and while they code.

Thinking Competency: Problem-Managing Reasoning	Example of coding using problem-managing reasoning	How problem-managing reasoning in coding helps with math…
Problem-managing reasoning refers to the ability to manage problematic situations in mathematics and coding. It includes understanding when and how a problem in any context can be mathematical or computational in nature and that the very nature of mathematics and coding is problematic. In **mathematics,** managing problems involves understanding that solving problems is itself problematic by nature requiring the selection of appropriate background knowledge, strategies, models, technologies, representations/visuals, conventions and structures, etc. It promotes a powerful and useful general way of thinking. It also involves being able to see the world as mathematical in nature (human behavior and functioning, nature, finances and economics, motion and forces, etc), understanding the significant role it plays in understanding and living within it, asking questions about the world and posing problem about it in mathematical terms. For example, considering whether it is better for a person to cross a field diagonally to get home or walk around a square block is a math problem that can apply the Pythagorean theorem and an understanding of rates and perhaps even vectors. Similarly, **coding** is problematic by nature as it requires that an ongoing set of decisions to be made including what	Coders are required to make many decisions to solve real world problems. They need to figure out what part of the problem needs to be tackled first and how to manage or arrange all the different parts of the problem in the most effective manner. Much like preparing to solve a jigsaw puzzle, coders must understand which pieces are the corner pieces, which are the edge pieces and which ones are the easy middle pieces and hard middle pieces. Let's take for example, a coder who in 2009 thinks he can build a better way for people to call and pay for taxis using their smart phones. The coder needs to break this problem down in manageable pieces. First, she needs to consider who the drivers are going to be and how to get these drivers to use the coder's platform? She need to figure out who the customers are going to be and what will encourage them to start using the app? Other parts of the problem will involve what the road map will look like and how it will best represent all the taxis available and all the people who want a taxi, how to decide which taxi goes to which person, what to do when a person cancels their request or when a driver cancels their availability, and so on and so on. Once these smaller problems are identified, the coder has to decide which of these problems to solve first through the software and how. The programmer who gets it right turns out to be Uber, a company worth 10s of billions of dollars. Coding is problematic by nature and coders requires effective problem managing to be successful.	Managing problems is what coders do. As students learn to code, they learn to manage problems in a way that parallels what they need to do to manage problems in mathematics. Coding helps students become more successful in math because it helps them understand that managing a problem in math requires that: • a number of important decisions need to be made and that they need to be made well (thoughtfully); • decisions need to be sequenced and serve different purposes to allow for larger complex problems to be broken down into smaller, related, more manageable problems that can be more readily solved; and • they understand that math, like code, is a language that can be used to describe a significant number of real world situations and to solve problems within it.

language and conventions are best to use; what appropriate representations to use, what data structures to use, what architecture to use for all the moving parts of an application, etc. Coding can also be used to describe elements of the world in computational terms, and to describe real world situations and solve real world problems. Consider for example the weather. Over the past 12 years, forecasts for daily high temperatures have become more and more accurate. Forecasts can now correctly estimate tomorrow's peak warmth to within 3 degrees of the actual highest temperature about 80 percent of the time. Twelve years ago, the margin of error was 4 degrees. Weather impacts how many fans attend sports events, how much energy is generated from wind farms or used in people's home and many other events where the use of company resources will change based on accurate predictions. Computer models that have lead to more accurate weather forecasting impact widely on how lives are led and how real world decisions are made.		

Thinking Competency: Reflective Reasoning	**Example of coding using reflective reasoning**	**How reflective reasoning in coding helps with math…**
Reflective reasoning refers to the ability to continuously seek clarification and a richer understanding of ideas by making on-going iterative connections between	Suppose a coder is tasked with a shopping website they have been hired to build. They start with a few items to purchase and simple ways to search for those items. As more items and more categories of items are added, they	As students learn to code, they learn to make coding decisions by reflecting on the quality of their code, continuously seeking improvement every step they take. Learning to reason in this

existing and new knowledge and understanding. In **mathematics**, reasoning is reflective when there are on-going checks for reasonableness and correctness as new knowledge and understanding is attained. Take for example a student who understanding that two geometric shapes are congruent if all of their measurable attributes are the same. After learning about similarity, the same student can describe congruent shapes as possessing a 1:1 relationship with respect to their measureable attributes; a more precise, richer understanding is attained by connecting new learning to previous learning. Similarly, in **coding**, students continuous ask, now that I have learned something new, how does this affect the stuff I already know, have already completed, and how can I make it clearer and more effective? For example, when coders refactor (revisit old code and rebuild it in a more effective / efficient way), they are reasoning reflectively. Coder continuously revisit their code upon reflection to make their programs more effectively and accurately meet their code's objective. Reflective reasoning is encouraged in both coding and mathematics when students are given a larger more complex task or problem to complete—one they are not fully equipped to handle yet—to which they build a respond or solution over time as they continuously connect new learning	go back to their previous search method and refactor it to account for the new complexities. For example, before the coder had only one brand of item for which to create a search code but then more items and therefore brands need to be added. The code now has to be modified to search for 5 different brands, not just one, requiring the software to do more. In order to make sure the software works properly with the additional functionality, strong reflective reasoning skills are needed to figure out what is working in the existing version, what needs to be added, and how it can be added without breaking the existing functionality. Consider another example. A student coder builds a checkerboard by drawing 64 squares using 64 lines of code. She then learns what a loop is, and refactors the code to draw 8 sets of 8 squares, one for each column. Next, she learns what a nested loop is (one loop inside another loop) and refactors the code to draw 8 sets of 8 sets of a single squares, the first part for each row and the 2nd part for each column. Now the finished code goes left to right in a loop and top to bottom in a loop. When coders see patterns like this one and use the patterns to simplify their code they are reasoning reflectively. They are continuously checking on the quality of their work and iteratively connecting new learning or ideas to the problem at hand in order to make it most effective.	way helps students experience greater success in math as it teachers them how to: **1.** ensure all of their mathematical decisions are reasoned and make sense; **2.** be aware of their learning gaps and how to close these gaps as they continue to learn; **3.** ensure they are accurate and precise in their thinking; **4.** develop clarity and depth of understanding of the math they are learning over time; **5.** develop a growth mind set (continued effort, perseverance, flexibility) and to understand why it is important; **6.** become confident, independent, on-going self-regulators and assessors. **7.** focus on understanding the essential components of a problem and weeding out unnecessary distractors. Reflective reasoning ultimately helps students develop a "thinking toolkit" to be used in the solving of future problems.

back to their previous responses based on old learning to develop richer, more detailed and at times innovative responses.		

Thinking Competency: **Communicative Reasoning**	**Example of coding using** **communicative reasoning**	**How communicative reasoning in coding helps with math…**
Communicative Reasoning: The ability to effectively use mathematics or code as a language to understand and express ideas. In **mathematics**, this refers to the capacity to communicate effectively in a way that is appropriate to the purpose and audience of the communication. For example, to reason communicatively means making effective decisions about what form of representation to use (visual, graphical, algebraic, geometric), how to structure these forms correctly (nature and position of title for example), how to best interpret and read these forms and how to infer meaning when necessary (by making assumptions). Representational and structural reasoning are component parts of this broader competency. In **coding**, this refers to the capacity to communicate effectively with code and with commenting code in a way that is appropriate to the audience reviewing the code being written. This audience includes: the person coding today, the same person reviewing the code in the future, and any team members present or future who will also be looking at the code to review, refactor or improve it.	Every computer language has generally agreed upon coding conventions – when to use spaces, when to use indentations, how to arrange the code so that it is easiest to read and understand. For example, a coder who uses the language of code effectively makes sure a code is indented properly and separates functionality inside the program into discrete smaller functions that work with each other rather than as one large function which does the same thing but is more difficult for a reader to follow and understand. Also, communicative reasoning ensures that coder knows how to recognize and fix code that breaks down at any particular point. Often a coder is communicating with herself/himself. Properly organizing and commenting in the code will help the programmer to quickly understand for the future what the program does precisely, and enable it to be re-used or modified according to present needs. In code, each line of code is executed in order. A coder could technically write all the software for a complex website like Facebook all in one row without any structure, empty lines, or organization as long as the lines of codes all fit together and are presented in the right order. These millions of lines of code all	Learning to communicate effectively using code is especially useful in learning how to communicate effectively in math. Coding helps students learning mathematics to: 1. understand (for themselves and others) the value and purpose of mathematics as a universal language that can be read and written (accessible and useful) by all people; 2. sustain the continuous development of new mathematical thought; 3. develop the skill of effective argumentation as they are able to justify their ideas to others; 4. work well and collaborate with others. This type of reasoning is often neglected by mathematically oriented thinkers. However, any mathematical ideas, conclusions or results, even brilliant ones, that are not properly communicated are destined to be deemed not valuable and ultimately ignored. In addition, research grants, or any employment requiring mathematical reporting, accountability, or any form of (persuasive) communication (finance, sales, business deals, banking, etc.) must be able to communicate the intricacies, strengths, and potential benefits of a proposed mathematical idea in a manner that is both accessible and eloquent.

Code is not instantly readable like language is – it is a collection of symbols and a very limited number of full words. Instead, programmers with high levels of communicative reasoning will ensure that every piece of code they write will be readable in the future by choosing good variable names and by writing comments inside the code which explain what every chunk of code does, where a chunk can be anywhere from a single line to an entire function worth of code. This being said, a high level of communicative reasoning will be displayed when there are helpful comments every few lines.	written in one row would be called Spaghetti code – something that delivers the correct end result but isn't functional in terms future improvements or modification. It would be difficult to fix problems, make improvements, getting rid of parts that aren't needed, etc. with code communicated in this way. Overall, a coder who spends extra time to properly format their code displays higher levels of communicative reasoning.	

Thinking Competency: **Connective Reasoning**	**Example of coding using** **connective reasoning**	**How connective reasoning in coding helps with math…**
Connective reasoning refers to the ability to understand the nature of how, and degree to which, ideas are related. Connective reasoning in **mathematics** refers to the ability to seek out useful relationships between mathematical concepts or ideas. For example, it is useful to understand the degree to which concepts such as addition and subtraction; addition and multiplication; multiplication and exponents; functions and relations; average and instantaneous rates of change are similar and different. Understanding relationships like this helps to make decisions on how to solve problems with	Consider the design of a website that sells many items; books with a specific number of pages, clothing items in various sizes, TVs of various screen sizes, toys made of a certain amount of pieces. To effectively keep an inventory of all the items being purchased and their price, the coder must sort the items based on their various degrees of similarities and differences. For example, all the items have a quantity, some share similar identifiable characteristics such a size or number to pieces, and have a price. All the items also have elements that are unique to them, e.g. no other item has a number of pages other than books.	Learning how to reason to connect elements, ideas, concepts in code in a useful manner helps students be successful in math because: 1. helps to improve and strengthen their mathematical understanding of important ideas and concepts; 2. helps to ensure they are considering all the mathematical details to decide which and to what degree they are important; 3. helps them to see mathematical relationships and seek meaningful patterns (for example, multiplication can be described as a loop of additions and exponents as loops of multiplication);

greater effectiveness (in the simplest, clearest, more convincing) way. Connective reasoning in **coding** is all about being able to take the real-world elements that you are creating code for and properly translating them into a code structure that makes it easiest for the code base and for the user using the application to interact with, use, retrieve and manipulate the real-world elements and their digital equivalents. In essence, connective reasoning in **coding** is about understanding how certain types of applications are best solved with certain algorithms and design choices. For example, an e-commerce website which accesses small amounts of data at a time needs one type of database while a data analytics application that accesses the data very often and accesses lots of data at a time needs a different type of database. Understanding the relationship between different types of databases and how easy it is to access the data helps to make decisions on how to solve problems and build software with greater effectiveness (in the simplest, fastest, most easy-to-use way).	Being able to figure out what are the common and unique elements helps the coder to create an appropriate data structure and database to properly store, access and manipulate the stored items to keep an ongoing inventory from a coding perspective. If coders do not consider the degree to which elements in the database are related, they will build a database with an architecture that makes the use of the software slow, clunky and replaceable. Optimizing database response time means optimizing the length that a user has to wait before a page loads. Waiting 10 seconds or more for a page to load means users will disengage and stop using the software. Connective reasoning in code can ultimately lead to creating code that is deemed valuable and desirable rather than creating one an end that no one wants to use and another coder can do better.	4. supports creative and innovative thinking in mathematics by allowing students to make new and valuable connections; 5. helps to contextualize and decontextualize problems making them easier to solve in more mathematically sound ways. In essence, mathematics is the science of connecting ideas that are precisely defined, and absolute in their truth. Students that come to understand this by making on-going mathematical connections and seeing relationships between various concepts are able to develop a sense of the “oneness” of mathematics.

Thinking Competency: **Strategic Reasoning**	**Example of coding using** **strategic reasoning**	**How strategic reasoning in coding helps with math…**
Strategic reasoning refers to the ability to select appropriate strategies, methods or approaches, and supporting tools, to effectively aid in completing mathematical or coding tasks or problems.	When sorting a large amount of different numbers, e.g. prices, all of which are different from each other, a different sorting algorithm would be more appropriate than the one used to sort a large amount of similar number, e.g.	Reasoning strategically in code helps student be more successful in math as it: 1. supports all mathematical thinking (sense-making, representing, organization, mathematizing, problematizing…),

In **mathematics**, completing a task or solving a problem requires the selection of the most appropriate strategy, method, model and tools from a wide range of alternatives. For example, a mathematician chooses to use a computer or a calculator instead of using mental math or other devices because it will lead to the completion of a task or helps to solve a problem more effectively (more precise, accurate, efficient, fitting to the nature of the task or problem). This is also true of strategies, models or approaches they decide to use. For example, multiplication is more appropriate when multiplying a large number of equivalent sets of values than addition.

This is also true in **coding.** When building software, there are multiple approaches to complete the work. A "quick and dirty" fix may solve a problem quickly but may not allow for solving other problems that come up in advance; building a simple function which might be re-used a few times; or importing or creating a library for use not only for the current problem, but for a long-term set of software challenges. For example, when building software to quiz users on mathematics, you might build a "graphie" library containing multiple functions, classes and models to handle all variations of software needed for coordinate plane math, e.g. functions, derivatives, trigonometry, etc. Once built, this library could be imported whenever content is being developed that involves the coordinate plane.

sorting heights of a large number of people. The latter will only come from a small set of values (from 48 inches to 96 inches as people are generally between 4 feet and 8 feet tall) while the prices in a store might vary from 5 cents to $99,999.99, i.e. a larger set of numbers with tens of thousands of different values. A sorting algorithm must be strategically selected that best fits the nature and quantity of the information being sorted.

Another example is picking a computer language to work in for a specific project. Some languages work really fast but are difficult to use. These might be appropriate for building operating systems like Windows or the Apple operating system. Some languages are very useful for mathematical and statistical manipulation, so if you are building an application whose primary purpose is do this kind of work, you might use a language specifically built for this task. Some languages are built so that the code written is easy to read (good for large teams) while other languages are written to be easy to write (many shortcuts and abbreviations are included within the language).

Understanding the objective of the code to be created helps to select the best strategies and tools to use to meet the objective. Strategic reasoning in code helps to better create and implement more effective solutions.

ensuring that best mathematical decisions are made strategically;

2. supports the development of fluency;
3. helps to deepen understanding; and
4. helps to improve efficiency, accuracy, and precision as the most fitting selections are made to solve a problem.

Note both mathematical thinking and computational thinking are defined by a set of thought processes needed to formulate and solve problematic situations and task in an effective manner; both are iterative processes that move through, sometimes back and forth, 3 well-defined stages: problem formulation; problem solution expression or representation; and solution execution and analysis. All of these stages executed well ensure that the language of mathematics or coding are used effectively to communicate solutions.