Incorporating Computational Thinking and Coding in BC Secondary Mathematics Classrooms

by

Yuan-Ting Erica Huang

B.Ed., University of British Columbia, 2011

B.Sc. (Hons.), University of British Columbia, 2006

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
Secondary Mathematics Education Program
Faculty of Education

Approval

Name:	**Yuan-Ting Erica Huang**
Degree:	**Master of Science**
Title of Thesis:	**Incorporating Computational Thinking and Coding in BC Secondary Mathematics Classrooms**
Examining Committee:	**Chair:** Peter Liljedahl Professor
	Nathalie Sinclair Senior Supervisor Professor
	Sean Chorney Supervisor Assistant Professor
	Rina Zazkis Internal Examiner Professor
Date Defended/Approved:	March 10, 2020

Ethics Statement

Abstract

There has been considerable attention on the term “computational thinking” (CT) over the past decade in the education community. With a global movement to include coding in the school curriculum, British Columbia (BC) also introduced coding to the K-12 curriculum in 2016. There have been on-going discussions about what CT is, why we should teach CT (and coding), and how we should teach it. However, there has been little research on the current state of affairs in BC with respect to teacher practices related to CT. By surveying, observing and interviewing BC secondary mathematics teachers, this study focuses on teachers’ perspectives on how to incorporate CT and involve coding in classrooms. Results showed that most teachers understood CT as being about problem-solving skills. CT and coding have not been taught frequently but are incorporated in various ways, primarily using block-based programming. Despite challenges, teachers found that these CT and coding activities elicited a high-level engagement and were accessible to a wide range of students.

Keywords: computational thinking; coding; mathematics education; computer science education; STEM; BC curriculum

Acknowledgements

I would like to express my sincere gratitude to Dr. Nathalie Sinclair. Thank you for your patience and guidance throughout this journey. Your wealth of knowledge, valuable feedback and continued support contributed tremendously to this work.

I would also like to thank Dr. Sean Chorney for reviewing my thesis drafts. Thank you for your thoughtful comments and attention to detail, which helped refining my thesis.

I would like to extend my gratitude to all my professors and fellow classmates in the Secondary Mathematics Education program. I thoroughly enjoyed our conversations about mathematics education and learned much from each and every one of you.

I am also grateful to all the teachers that participated in this study. Thank you for providing insightful responses and welcoming me into your classrooms.

Moreover, I want to thank my students, who signed up for my Computer Science class in grade 12 and shared memories of coding with the orange Scratch cat when I taught them in Mathematics 8. Such enthusiasm motivated me to continue to explore computational thinking and coding in mathematics education.

Finally, I would like to thank my family for the unconditional support in all my endeavors.

Table of Contents

List of Tables

List of Figures

List of Acronyms

ACM	Association of Computing Machinery
ADST	Applied Design, Skills and Technology
AP	Advanced Placement
APCSA	AP Computer Science A
APCSP	AP Computer Science Principles
BC	British Columbia
CAS	Computing At School
CS	Computer Science
CSTA	Computer Science Teachers Association
CT	Computational Thinking
DGS	Dynamic Geometry Software
DME	Dynamic Mathematics Environment
ISTE	International Society for Technology in Education
MIT	Massachusetts Institute of Technology
RPS	Rock-Paper-Scissors
SFCPF	Structuring Features of Classroom Practice Framework
SFU	Simon Fraser University
STEM	Science, Technology, Engineering and Mathematics

Chapter 1. Introduction

A fellow teacher Alicia invited me into her classroom as a guest, as we collaborated on designing a coding task for her mathematics class.

I noticed Blaise when he entered Alicia's class. He walked straight to the back of the room and sat down. During the lab session, he worked by himself and did not interact much with other classmates. While I circulated around the room, he turned to me when I was close by, and said that the challenge was impossible to complete.

"How could you change the length of this?" He asked, pointing to an edge of a shape his program drew on the screen.

I thought that he likely did not complete all the steps on the handout that led to the final challenge. I flipped the page over and pointed to an earlier step that showed how to use a variable.

Then Blaise seemed to have an "aha" moment and went back to work. Towards the end of the class, he called Alicia over to show her what he had done. A few of his classmates gathered around him to watch his demonstration. Blaise was the only student in the class that solved the challenge. Alicia was excited and complimented him after he explained the code and his reasoning.

After the class, Alicia and I talked about how the class went. She was happy about the lesson, and this was also when I found out that Blaise was failing her mathematics class.

Knowing that a struggling student had a chance to shine made my day.

As a rare computer science major among my colleagues, and a teacher with experience working in the industry as a software developer, I have tried coding tasks in mathematics classes since the first year the new BC curriculum was implemented. However, at the time I was the only mathematics teacher at my school to do it. Very few teachers shared the same enthusiasm. It was great to have a moment like this, a reminder of what really motivated me to conduct this research in the first place. I *witnessed this repeatedly in my own classroom, but now I also saw it in another colleague's classroom.*

Then I could not help but wonder, if programming could be such a powerful tool to engage students, how come I am still doing this research, many years later after Seymour Papert's work on LOGO? How could teachers be better supported so that more people are willing to incorporate coding in classrooms?

When the British Columbia (BC) Ministry of Education introduced coding in the curriculum in 2016 (Silcoff, 2016), it was met with both excitement and skepticism. I was one of the few mathematics teachers enthusiastic about this change, but I was also very aware of many concerns, such as lack of resources, insufficient training, and an already overcrowded curriculum, that this addition brought among teachers. A few years after the implementation of the new curriculum, bringing coding into classrooms still seems far from being entrenched in practice in BC classrooms, especially at the secondary level.

Before I began my graduate studies in the secondary mathematics education program at Simon Fraser University (SFU), I tried several coding activities in my classes, ranging from using "Hour of Code" (https://hourofcode.com/ca) tutorials to designing coding tasks for a unit on algebra. These activities were met mostly with enthusiasm by my students. The student engagement led me to believe that the effort was worthwhile. However, I always felt that I needed to integrate coding into mathematics more naturally, and I also wanted to help other colleagues do it.

As part of my course work, I had the opportunity to re-read Seymour Papert's Mindstorms: Children, Computers, and Powerful Ideas (1980). I also had the chance to learn more about Wing's (2006) paper on computational thinking (CT) that led to the recent worldwide movement to include coding, or more generally computing, in school curriculum. Moreover, through course projects I collaborated with colleagues, including Alicia who I mentioned earlier, to create coding tasks for mathematics classrooms. My interest started with coding in mathematics classrooms but expanded to include CT more broadly. CT is a complex term. Although it appears in curriculum documents, its meaning is not elaborated and widely shared. Having exposure to more theory, as well as hands-on experience, I became more curious about the topic. There are many relevant potential areas for research, such as the impact on students, lesson design and technology usage, but I was especially interested in teachers' perspectives of how to incorporate CT and coding in classrooms. I believe that to see change in education, teachers have to be heard and be involved in the process.

The main intent of this research is to better comprehend teachers' understanding of the term CT, interpretation of coding in the new curriculum and current practice incorporating CT and coding in BC secondary mathematics classrooms. In Chapter 2, I review related literature that discusses a variety of definitions of CT, clarify relevant terms, and discuss CT and coding in the context of both mathematics education and K-12 education. I then present my research questions. In Chapter 3, I explain the theory and methods for this study, which include the theoretical framework utilized for data analysis, descriptions of the survey and interview participants, as well as methods for data collection and analysis. Throughout Chapters 4 and 5, I analyze the research data, which consist of teacher survey, classroom observations and teacher interviews. In Chapter 6, I present the cross-case analysis. Finally, I answer the research questions, make conclusions and suggest future considerations in Chapter 7.

Chapter 2. Related Literature

There have been many studies on computational thinking and coding, as well as how they are incorporated in classes. In this chapter, I first review literature on computational thinking, whose definition is still not agreed upon, as the term may have different meanings in different context. I then clarify relevant terms, including computational thinking, computer science, programming and coding. Next, I focus on literature related to computational thinking and coding in the K-12 context, as well as studies in mathematics education. Finally, I present my research questions for this study.

2.1. Introduction to Computational Thinking

Computer scientist Jeannette Wing popularized the phrase “computational thinking” (CT) in her influential article published in 2006; her work caught considerable attention among the education community. Wing described CT as using fundamental computer science concepts to solve problems. She considered CT a skill set applicable to everyone, not just computer scientists; thus, she suggested adding it to reading, writing and arithmetic as a new competency for students. After articulating the vision that everyone can benefit from thinking like a computer scientist, Wing, along with colleagues Cuny and Snyder (2011), more formally defined CT as “the thought processes involved in formulating problems and their solutions so that the solutions are in a form that can be effectively carried out by an information-processing agent” (p. 1).

Selby and Woollard (2014), to develop a more robust definition of CT, reviewed literature on discussions of CT since 2006 and created criteria to evaluate frequently used terms that are included in various definitions of CT. In their conclusion, they described CT to be a cognitive process that reflects five abilities, which are elaborated below.

- the ability to think in abstractions
- the ability to think in terms of decomposition
- the ability to think algorithmically
- the ability to think in terms of evaluations

- the ability to think in generalizations (p. 5)

Abstraction and decomposition, according to Selby and Woollard (2014), are two key terms that showed up consistently in their review of definitions of CT. Wing (2006) also highlighted these two practices as essential in tackling large, complex problems. The Computing At School (CAS) group's teachers' guide for CT (2015), which Selby and Woollard helped develop, used the example of the London Underground map, as an everyday example to illustrate abstraction – the skill to hide unnecessary details and represent data with important information. Through abstraction, a problem can be made easier with simplified yet good representations. In computer science, abstraction can occur at different levels, from using flowcharts when designing systems to choosing appropriate data structures inside programs. The term decomposition, in computer science, is breaking a problem down to small, manageable parts. A computer programmer may implement many separate functions in a program. Writing an essay involves outlining different parts – an introduction, body paragraphs and a conclusion. Subdividing a problem into smaller, easier-to-solve ones is a technique used in solving problems in other disciplines, and even in our daily lives.

The abilities to think algorithmically, in terms of evaluations and in generalizations are also commonly accepted as comprising CT. To think algorithmically, to solve problems step by step, is what enables solving problems on a computer, or more broadly, an "information-processing agent" which is the term used in Wing, Cudy and Snyde's definition (2011). Clearly defining a sequence of rules to be followed precisely ensures that the solution works the same every time when executed by an information-processing agent – which can be a computer, a robot, any kind of digital device, or even a person. The ability to evaluate means the ability to find the best solution – whether the solution is correct, efficient or user-friendly. Wing (2006) gave the example of making trade-offs between time and space, power and storage when solving computing problems. The ability to think in generalizations, to recognize patterns and make connections, is also crucial to problem-solving. This skill is applied frequently in mathematics classes. For example, CAS (2015) provided in their teachers guide the example of students drawing polygons. After learning about drawing triangles, squares, pentagons etc., the ability to generalize allow them to extend their learning to draw more complex shapes such as a hexagon or an octagon.

While Wing's article has been well received by educators, there is variability in the interpretation of CT – viewpoints from teachers, industry and academia can be different. For this research, Wing, Cudy and Snyde's definition, with the set of CT strategies highlighted by Selby and Woollard, will be used as the working definition of CT. Non-academic resources often list Wing, Cudy and Snyde's definition with four characteristics of CT – abstraction, decomposition, pattern recognition (generalization) and algorithms, leaving out evaluation. Sources cited on Wikipedia, such as British Broadcasting Corporation (BBC) and Google for Education, used this set of characteristics in their descriptions of CT. This may explain why that is what teachers commonly see in CT teaching resources and at professional development workshops. However, in Wing, Cudy and Snyde's definition of CT, the solution to a problem should be "effectively" carried out, which implies assessing the solution in the problem-solving process.

Although the CT movement was ignited by Wing in 2006, the idea of CT started much earlier. In 1962, computer scientist Alan Perlis advocated for teaching programming to all students in colleges and universities to provide students "a step toward understanding a 'theory of computation,' which would lead to students recasting their understanding of a wide variety of topics (such as calculus and economics) in terms of computation" (in Guzdial, 2008, p. 25). The term CT first appeared in Seymour Papert's book (1980). Papert pioneered the idea of supporting children develop procedural thinking skills with the programming language, called LOGO, which he co-invented with fellow artificial intelligence researchers Wally Feurzeig and Cynthia Solomon. He had a grand vision of an educational system in which programming in environments such as Turtle Geometry is leveraged as a learning tool for children.

Unlike most conventional school subjects, there is a strong push towards CT and computer science education from the industry. Many technology giants have related education outreach initiatives. As technology advances, computing is reshaping the workplace, especially in science, technology, engineering and mathematics (STEM) fields. Industry professionals may have a different interpretation of CT. For example, Google's Exploring Computational Thinking project defined CT as involving "a set of problem-solving skills and techniques that software engineers use to write programs that underlie the computer applications you use such as search, email, and maps" (Google for Education, n.d.), with more emphasis on CT's connection to implementing computer applications.

Furthermore, some researchers in academia do not have the same views. Many computer scientists have critical perspectives of the new CT movement ignited by Wing. Denning (2009), for instance, suggested that CT is not "thinking like a computer scientist" and argued that CT has a history in not only computer science but in all sciences. He claimed that CT has already been known since the 1950s as "algorithmic thinking" (p. 28). He also pointed out that the term CT appeared in discussions by scientists when advocating for making computation the third approach to science, in addition to theory and experiments. Thus, CT is an inadequate representation of CS, and Denning was concerned about using it to promote CS. According to Guzdial, Kay, Norris and Soloway (2019), little evidence supports Wing's claim that learning about computation helps students develop every-day problem-solving skills. They contended that students may already encompass some of these skills through interactions with computing everyday. The author provided several examples to illustrate their points. In one of the examples, they pointed out that playing the video game Fornite involves solving complicated problems such as navigating maps and managing ecological systems. Players demonstrate CT skills such as abstraction and decomposition. The authors better valued designing improved computing tools to provide environments that support good thinking in general, over teaching CT in schools. While researchers such as Denning and Guzdial had concerns or even did not see the need of the term CT, as a K-12 educator, I still believe that weighing the advantages and disadvantages, teaching CT in schools is worthwhile.

Even to this date, there is still on-going conversations about what CT is, why it should be taught and how it should be taught. There may never be consensus because different stakeholders have different goals and needs.

2.2. Computational Thinking, Computer Science, Coding and Programming

While there is already confusion in defining CT, the problem becomes more complex when many other related terms are used, such as computer science (CS), coding and programming, all of which appear in many K-12 curriculum standards.

It is easy to mix up the ideas of computer science and computational thinking. Computer science is usually seen a discipline, the study of knowledge of computing and its applications; most universities have a department for computer science. On the other hand, computational thinking, as Wing suggested, is a set of problem-solving skills. Wing defined CS as “the study of computation — what can be computed and how to compute it” (2006, p. 34) and described CT as a skill set that helps us “think like a computer scientist” to solve problems.

It is also common to confuse CS with programming or coding, and this is a long-term issue. Before the 1970s, CS was almost considered as equivalent to programming; most of the courses in an undergraduate CS program were programming courses. However, by 1986, Denning chaired a task force, commissioned by the ACM (Association of Computing Machinery), that aimed to define CS as a separate discipline and distinguish it from just programming. We are now already beyond the narrow view of CS being the same as programming (Armoni, 2016).

Programming and coding are used (or misused) interchangeably nowadays, and some argue that there is a distinction. A program is an entity that contains procedures, or a sequence of computational instructions, that solve problems with a computer. A piece of code refers to the written text, in the chosen programming language, that is part of a program. From a computer programmer’s point of view, programming is broader than coding. Programming means developing an executable program, implying having more designing and problem-solving in the process. If creating a software application is like writing a book, coding is merely writing sentences, whereas programming involves writing, organizing and planning the chapters. Nevertheless, now the two terms are often used as synonyms, especially in the K-12 education community and by the general public. Many educational initiatives such as the Hour of Code and Canada Learning Code promote “coding” when in fact, they aim to promote CS and even have some activities that cover topics that do not involve programming, such as the internet, big data and privacy. The main audience of this research project is K-12 educators, and the term “coding” is widely understood as equivalent to computer programming; therefore, it will be used in the way that is commonly understood unless otherwise specified.

Though the title of this study put CT and coding side by side, they are not completely separated ideas. The relationship is not parallel; coding, or programming, can be utilized as a vehicle to help students improve CT skills. As it is mentioned earlier, Perlis and Papert both saw programming as a way to foster CT. When students program, as they create, debug, iterate, and remix to solve problems, they are exposed to CT skills (Lye & Koh, 2014). Nevertheless, understanding that programming is not the only way to teach CT is essential. Some raised the concern that in practice, most CT curricula focus on programming and do not specify CT strategies (Armoni, 2016). Denning (2009) also showed concerns about this fascination of CT from people outside the field; he anticipated that without a good grasp of what CT is, people may return to understanding CS as being equivalent to programming. There are efforts to counter this misperception, such as Computer Science Unplugged (http://www.csunplugged.org) and cs4fn (http://www.cs4fn.org), which were created with the objective of providing teachers resources to go beyond just coding, such as puzzles and activities that teach the binary number system, cryptography and searching algorithms. In addition, the College Board designed a new Advanced Placement (AP) Computer Science Principles course that focuses on CT practices and provides students a broader understanding of computing; only one of the seven big ideas of the course involves programming.

2.3. Computational Thinking and Coding for K-12 Education

Wing's call to action of teaching CT certainly created a movement at the university and college level. Locally, at the University of British Columbia, the CS department redesigned its introductory course for non-CS majors and named it "Computational Thinking". The CT movement also gained traction in K-12 education. Barr and Stephenson (2011) claimed that waiting until students are in college to introduce CT concepts is no longer sufficient, as students nowadays are greatly influenced by computing on a daily basis. Much of this push seemed to be driven by the technology sector, which also influenced education policy makers. BC's Ministry of Education also added CT and coding in the latest curriculum revision; these changes are elaborated and discussed in section 3.2.

Teachers are relatively new to integrating CT into pedagogical activities, and there can be many different definitions of CT. Wing's definition of CT has often been criticized for its vagueness (Tedre & Denning, 2016). Therefore, according to Computer Science Teachers Association (CSTA) and International Society for Technology in Education (ISTE), it is important to "build consensus for an operational definition of computational thinking that would be meaningful to an educator audience" (2011, p. 4). Barr and Stephenson also stated that "passionate debate about the nature of computer science or computational thinking may provide intellectual stimulation for those in the computing fields. However, embedding computational thinking in K-12 requires a practical approach, grounded in an operational definition" (2011, p. 144).

There has been a lot of discussions regarding establishing an operational definition for CT to provide K-12 teachers a framework to incorporate CT in classrooms. CSTA and ISTE (2011, p. 13), in collaboration with partners from higher education, industry and K-12 education, developed a Computational Thinking Leadership Toolkit and proposed the following definition:

"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."

While this definition is considered to "present a rich portrayal of CT," it is still criticized for its vagueness about details (Tedre & Denning, 2016, p. 125). The last four characteristics seem to align with abstraction, algorithmic thinking, evaluation and generalization from the work by Selby and Woollard (2014). Formulating problems for computers to solve is part of Wing's ideas of CT. In her words, "[c]omputational thinking is reformulating a seemingly difficult problem into one we know how to solve" (Wing, 2006, p. 33). This includes using abstractions, decomposition, etc. to represent problems in new and different ways, ways that make problems easier to implement on computers. To logically organize and analyze data, such as finding appropriate data structures, performing statistical calculations and using visualizations, become an important part of computing practices with the advancement of digital technology. One thing to note is that this operational definition specific used the wording “include (but not limited to)” to allow for extensions and variability in interpretations of CT.

To extend on the operational definition, Barr and Stephenson (2011) proposed a model to provide K-12 teachers some concrete examples of how each of these CT concepts can be applied across multiple core school subjects such as mathematics, science, social studies and language arts. For example, abstraction in science class may look like building a model to represent a physical entity; algorithmic thinking in mathematics class is something students demonstrate when finding the greatest common factor of two numbers. Viewing CT as a scientific practice, Weintrop et al. (2016) also developed a CT taxonomy framework for K-12 mathematics and science classrooms. With a broadened view of CT, their taxonomy includes practices in four main categories: data, modeling and simulation, computational problem solving, and systems thinking. Data practices, such as collecting, creating, manipulating, analyzing and visualizing data, are crucial to all sciences. Modeling and simulation include using computational models to understand concepts, as well as to find and test solutions. Assessing, designing and constructing these models are important parts of CT practices, too. The category of computational problem solving, while including many of the strategies in our working definition, was developed with the perspective of solving problems in the scientific domain. Programming, troubleshooting and debugging, which are common practices in CS, were added as problem-solving strategies. As the study of many scientific phenomena involve understanding and investigating large and complicated systems, systems thinking, which can include studying a complex system as a whole, understanding relationships between different components, communicating about the system such as using visualizations and infographics, and defining systems and managing its complexity. This taxonomy was proposed as a response to including CT in school mathematics and sciences curricular materials, with the perspective of using computation in mathematics and scientific context.

With the rise of block-based programming and easy access to free, online programming environments and tools, there has been progress in incorporating CT and coding at the K-12 level. Several of these programming tools, in various degrees, fit the “low floor, high ceiling” guiding principle (Grover & Pea, 2013). One of the most popular tools is Scratch (https://scratch.mit.edu) offered by the Media Lab at Massachusetts

Institute of Technology (MIT), which was inspired by LOGO and the work by Papert. Scratch, a block-based, visual programming language and environment, is designed with the intention of helping kids learn computer programming through creating fun projects such as animations and games. AP Computer Science Principles (APCSP), which emphasizes CT and complements the traditional AP Computer Science A (APCSA) course that primarily focuses on programming in Java, has also gained popularity. The total number of students taking the APCSP exam increased from 43,780 in 2017 to 70, 864 in 2018. The number of APCSP exam takers also exceeded the number of APCSA exam takers for the first time in 2018. (College Board, 2018). CT and CS education certainly made strides in recent years.

It would be reasonable to conclude that incorporating CT and coding in K-12 classrooms is valuable. CSTA and ISTE (2011) highlighted that not only could CT potentially provide students an opportunity to develop problem-solving and critical-thinking abilities, but it is also considered vital for future jobs, as computing plays an increasingly prominent role in every industry. Moreover, CT and coding could possibly support student learning in other disciplines. Despite the potential benefits and the progress in research, CT and coding have not been fully integrated into K-12 education, especially at the high school level. One of the major challenges is teacher training. Preparing teachers, most of whom do not have background in CS, to include these concepts in their teaching, can be a challenging task. The process would require system change, encouragement of teacher engagement and development of resources (Barr & Stephenson, 2011).

2.4. Computational Thinking and Coding in Mathematics Education

Although the term CT was made popular by the CS education community, it has deep roots in mathematics education. Wing (2006) also stated one of the characteristics of CT is that it “[c]omplements and combines mathematical and engineering thinking” (p. 35). She recognized the connection between CS and mathematical thinking, and how mathematics is fundamental to all sciences. Moreover, computation has become increasing important in the discipline of mathematics and sciences, which is another motivation to bring CT into mathematics learning (Weintrop et. al., 2016).

What motivated Papert (1980) to develop LOGO, was his vision for combating mathematics phobia through creating a computerized learning environment that allows children to explore mathematics with a student-centered, project-based approach. The computer, in Papert's opinion, can play a subordinate role to mathematics learning. Papert claimed that the computer can help learners establish a new relationship with mathematics. He believed that one of the powerful features of teaching with computers is the feedback students receive. Students do not need to be told by the teachers whether they are right; they could see themselves if the program worked as they intended. Student, thus, are more willing to take risks and fix mistakes through debugging their programs.

Papert created the "Turtle" – a "constructed computational object-to-think-with" which existed within the LOGO programming environment (p. 11). With LOGO, Papert proposed a new style of teaching geometry, which he called "a computational style of geometry" (p. 55). Strongly influenced by psychologist Jean Piaget, Papert believed in having children learn from exploring and constructing their own understanding, and his vision was that students can acquire many different concepts in the LOGO environment, not just in geometry, and the learning is not limited to mathematical ideas, either. Papert's work inspired generations of educators to explore his ideas.

Although there has been a lot of excitement about how CT can support mathematics learning, there are relatively few studies. Specifically focus on computer programming, Clement (1999) stated that early research results were conflicting, and positive results could be inconsistent or limited. When LOGO first came out and was considered by many educators as a potential medium to foster critical-thinking skills and mathematical learning, studies conducted by the Bank Street College of Education, showed that students programming with LOGO did not show improvement in planning abilities and problem-solving skills that transfer to other context (Pea & Kurland, 1983). With LOGO more integrated into schools, educators developed the perception that programming was too difficult to have strong impact on mathematics learning (Benton et al., 2016). Clement claimed that positive cases relied on prudently planned sequence of programming activities; only exposure to programming would not have a positive effect in mathematics learning.

The call for adding CT as a new competency to every child's analytical ability, as well as the advancement in educational computing tools, especially in block-based programming such as Scratch, led to resurgence in the area of CT in mathematics education. Some noteworthy efforts include Bootstrap based at Brown University (https://www.bootstrapworld.org/), ScratchMaths (2016) by the University College of London and the Lattice Land project by Pei, Weintrop and Wilensky (2018), and these projects are elaborated below.

Schanzer, Fisler, Krishnamurthi and Felleisen (2015) stated that many projects that aimed to improve student mathematics performance through computing project were unsuccessful. They claimed transfer of learning mathematics from computing projects did not occur because "[t]ransfer of learning between domains typically requires both deep structural connections between the domains and explicit instruction in how to apply concepts from one discipline in the other" (p. 616). With this in mind, they developed Bootstrap, a programming curriculum that teaches students to create video games using algebra and geometry. The curriculum is designed to teach specific algebraic concepts, and students program in a functional programming language that better reflects how variables and functions work in mathematics. Preliminary data of their study suggested that Bootstrap improved student achievement on algebra word problems from standardized mathematics examinations in the US. Later in a larger study, Schanzer, Fisler and Krishnamurthi (2018) confirmed that students improved on algebraic word problems with the Boostrap:Algebra module.

ScratchMaths by the University College of London is another project that aimed to integrate CT and Math. It is a two-year math and CT based curriculum for students in grade 5 and 6 (Benton et al., 2016). The study found that the ScratchMaths curriculum had a positive impact on student learning of CT, especially for disadvantaged students. However, the evaluation report stated that many schools did not complete the mathematics portion of ScratchMaths; some teachers showed concerns regarding the pressure of Key Stage 2 SATs, the national standardized assessment for students in grade 6 in England. Based on the result from schools that fully implemented the program, there was no evidence of improvement in students' mathematics test results (Education Endowment Foundation, 2018).

Taking an approach different from Bootstrap and ScratchMaths, which emphasized programming, Pei et. al (2018) developed a project called Lattice Land as part of Northwestern University's CT-STEM project. Students explore geometry concepts in a "mathematical microworld", an idea inspired by Papert. Using a broader definition of CT by Weintrop et al. (2016), students were said to apply CT skills in Lattice Land, such as decomposition, abstraction, modelling and simulation. The design of project interface was similar to dynamic geometry software (DGS) but focused on discrete geometry. Their work demonstrated an example of connecting CT with mathematics, and the authors concluded that a well-designed computing environment can support the learning of the intersection of CT and mathematics habits.

Canadian researchers also made strides in CT in mathematics education. Gadanidis (2017) showed how grade 3 and 4 teachers, with little experience in programming and a variety of tools such as Scratch, the Repeating Patterns coding environment and coded simulations in Python, can integrate CT and mathematics in their classrooms. Savard and Freiman (2016) also conducted a study working with grade 5 to 7 students using robotics-based tasks. Their analysis indicated that student performance depends on the type of feedback — mathematical, digital or socio-cultural, and how students interpreted the feedback. They highlighted the concern that students, used the trial-and-error strategy based on feedback from the robots, which may not reflect understanding of the mathematics concepts. For example, instead of using their discussions on circumference of the wheel and rotation time, students ended up guessing and experimenting with parameter values to get the robot to move within the time limit set by the task.

Incorporating CT and coding in mathematics education still requires much further studies; study results have been tentative. Moreover, there is especially a lack of research at the high school level. Analysis suggested that although the number of studies in the area has grown, most studies are conducted by academics working in CS departments, not from education. Moreover, studies tend to concentrate on teaching programming, rather than supporting the learning of mathematical concepts (Hickmott et. al., 2017).

2.5. Research Questions

Although CT and coding have become part of the BC curriculum, and some teachers see the potential benefits, they may not put emphasis on CT and coding in classrooms. If we want to realize the vision of incorporating CT and coding in more K-12 mathematics classrooms, we must investigate teachers' current practice because 1) how teachers understand CT determines how they implement it in the classroom and 2) we need to meet teachers where they are if we would like to continue to make progress and better our practice. How are mathematics teachers currently incorporating CT and coding? What are we already doing well? How can we improve? For teachers that have not tried it, what are some factors that have prevented them from trying?

I am interested in exploring how BC secondary mathematics teachers are incorporating CT and coding in their classrooms. Specifically, I am interested in answering the following questions:

- How do teachers understand "computational thinking" and interpret "include coding" in the BC mathematics curriculum document?
- What kind of coding and CT activities are developed and used in BC secondary mathematics classrooms?
- What challenges do teachers face and how can they be better supported in incorporating CT and coding?
- To what degree are CT and coding integrated into BC teachers' mathematics classes?

Chapter 3. Theory and Methods

In this chapter, I introduce the Structuring Features of Classroom Framework created by Ruthven (2009), the framework used to analyze this research. I then review computational thinking and coding in the latest BC curriculum, which motivates this research and provides context of the study. Next, I discuss the research setting and provide information on the participants. Methods used for data collection, which include surveying teachers, observing in classrooms and conducting teacher interviews, are discussed in detail. Lastly, I offer an overview of how the data are analyzed.

3.1. Structuring Features of Classroom Practice Framework

This research aims to explore teachers' current practice for incorporating CT and coding in mathematics classrooms. As a pilot for this study, I made classroom observations in two colleagues' classes when they tried coding tasks with their students. Soon after my first observation, I realized that I needed a more structured approach that would enable me to find themes to focus on in a classroom observation. Although using unplugged activities, activities that do not require computers, to incorporate CT and coding in mathematics classrooms is possible, most of the time teachers utilize digital technology. Thus, Ruthven's (2009) Structuring Features of Classroom Practice Framework (SFCPF), developed to help better understand technology integration in mathematics classrooms, is utilized to support my analysis. This framework was chosen as it focuses on teacher's practice, whereas other theories may emphasize on the technology use for student learning.

Ruthven's framework identifies five key structuring features of a mathematics classroom with technology integration: working environment, resource system, activity format, curriculum script and time economy. It names different phenomena in a technology-using classroom; each structuring feature was defined by Ruthven as follows (2014, p. 12):

Table 1. Components of the Structuring Features of Classroom Practice Framework

Structuring feature	Defining characterisation
Working environment	Physical surroundings where lessons take place, general technical infrastructure available, layout of facilities, and associated organisation of people, tools and materials
Resource system	Collection of didactical tools and materials in use, and coordination of use towards subject activity and curricular goals
Activity format	Templates for classroom action and interaction which frame the contributions of teacher and students to particular types of lesson segment
Curriculum script	Loosely ordered model of goals, resources, actions and expectancies for teaching a curricular topic including likely difficulties and alternative paths
Time economy	Frame within which the time available for class activity is managed so as to convert it into "didactic time" measured in terms of the advance of knowledge

An example of how the conceptual framework was applied is an investigative lesson with dynamic geometry, part of a Cambridge study in 2008 (Ruthven, 2009). The teacher used interactive whiteboards and *The Geometer's Sketchpad* (Jackiw, 2011), a dynamic geometry software, in the lesson. The lesson involved students investigating the property of the circumcentre of a triangle. For the working environment, Ruthven observed how moving from a normal classroom to a computer suite could affect the lesson and how new routines were developed related to operating the computers. In terms of the resource system, Ruthven looked at how department policy of encouraging teachers to explore possibilities and report to colleagues affected the teacher's practice and how the teacher found dynamic geometry complementing the usual geometric constructions by hand for this lesson. The activity format, affected by the working environment, followed the format of a teacher-led activity in the classroom and individual student work in the computer suite. The teacher noted how the computer environment supported his own interactions with individual students, but there was less of fostering discussions during individual student activity. The teacher's curriculum script originally was based on experience working with students constructing perpendicular bisectors on triangles by hand using compasses. Through experience teaching with technology, the teacher expanded his curriculum script and highlighted the affordances of the ability to drag and move the geometric figures. Lastly, Ruthven investigated the time economy component, looking at the teacher's perspective of time such as physical time it took to move to the computer suite, didactic time and time invested to teach students a new tool. These key features allowed for investigations into crucial aspects of a technology-enriched mathematics classroom.

While Ruthven's framework is useful in analyzing a mathematics classroom using technology, a classroom incorporating a coding task differs in several aspects. These differences call for modifications to the framework for the usage of this study. In Ruthven's original framework, the focus of curriculum script was on teacher moves related to their pedagogical content knowledge in mathematics. Ruthven's (2009) use of the term "script" is "in the psychological sense of a form of event-structured cognitive organisation" (p. 138). An example of part of a teacher's curriculum script, from the case study of the dynamic geometry lesson mentioned previously, would be a teacher move of asking students what happens when an angle, formed by connecting the circumcentre with two vertices of the triangle, was dragged towards 90 degrees. Reviewing the lesson, the teacher commented on how this led to a class discussion and made him realize the point would fall on the midpoint of an edge, an obvious property that never occurred to him before. The technology helped expanding his available curriculum script for this topic.

However, I realized in the pilot classroom observations that making distinctions between mathematical content and coding related content is important. Through classroom observations, it is evident that students in these classes make use of coding concepts, such as conditionals and loops. Moreover, the classes tend to be more open-ended and less structured than typical mathematics lessons. In a coding lesson, teachers typically have the curricular tasks pre-organized in the format of a handout, or use online tutorials provided by educational tools. Students spent most of the class time to explore the coding task. There may not be much teacher guided instruction; therefore, there could be very little to observe in the classroom in this regard. For the adapted framework, which will be referred to henceforth as the adapted SFCPF, I would like to focus more on learning objectives in the curriculum script component. The curriculum script component in Ruthven's framework is originally used for mathematics learning outcomes, and the adapted SFCPF will include the analysis of coding concepts in the mathematics lessons.

To assist with the analysis of learning objectives related to coding tasks, I use the CT concepts identified by Brennan and Resnick (2012) from the MIT Media Lab, developer of the phenomenally successful block-based programming platform Scratch. The authors highlighted seven highly useful concepts in Scratch projects, and these key concepts can be found in typical programming projects and in non-programming context as well. The concepts are:

- sequences: an activity or task is expressed as a series of individual steps or instructions that can be executed by the computer
- loops: a mechanism for running the same sequence multiple times
- parallelism: sequences of instructions happening at the same time
- events: something that triggers another thing to happen, such as a button click
- conditionals: the ability to make decisions based on certain conditions
- operators: provide support for mathematical, logical, and string expressions
- data: involves storing, retrieving, and updating values

These key concepts are taken into consideration as part of the curriculum script component in the adapted SFCPF.

The adapted SFCPF enables me to answer the research questions by providing a way to document the CT and coding activities developed and used in BC secondary mathematics classrooms. All five components, especially activity format and curriculum script, focus on investigating the lesson observed. The working environment, resource system and time economy are facets surrounding a technology-using mathematics classroom that potentially can pose challenges for teachers.

3.2. Computational Thinking and Coding in BC Curriculum

One of the research questions is about teachers' reaction to BC's new curriculum related to CT and coding. The latest curriculum was introduced in 2016, and it includes CT and coding in the areas of Applied Design, Skills and Technologies (ADST) and mathematics. The addition involves two approaches, one is to create new, stand-alone Computer Science 11 and 12 courses within the curricular domain of mathematics, and the other is to integrate CT and coding into different courses. CS 11 and 12 under mathematics were new courses developed to provide more university-aligned, rigorous CS courses at the high school level. The term "computational thinking" first appeared in

ADST 6, though ideas from CT appear in younger grades. In the ADST stream, there is more focus on projects and hands-on activities. The curriculum is defined based on design thinking principles, such as ideating, prototyping and testing; there is no elaboration on what programming concepts to be included. In the mathematics pathway, coding is a suggested activity to develop the curricular competency of “reasoning and analyzing” from grade 6 to 9 and “reasoning and modelling” starting in grade 10. CT is considered a tool to help students develop logical thinking ability.

At the grade 11 level, CT is defined in the curricular documents, in both Computer Science in mathematics and Computer Programming in ADST. The Computer Science 11 and 12 courses cover most concepts that are covered in a typical introductory CS course at the university level, with more mathematics content such as use of computing for financial analysis and ways to model mathematical problems.

One thing to note is that even in the same curriculum standards, the definitions of CT are slightly altered in different areas. In BC’s Computer Programming 11 and 12 under ADST, CT was described as “formulating problems and their solutions so they are represented in a form that can be solved through an algorithmic process. Key components are decomposition, patterns and generalizations, abstraction, and algorithmic thinking.” On the other hand, in Computer Science 11 and 12 under mathematics, CT was defined as “a thought process that uses pattern recognition and decomposition to describe an algorithm in a way that a computer can execute.” Elements from the most popular definition by Wing (2011) and characteristics described by Selby and Woollard (2014) can be seen in both definitions.

There are numerous initiatives that support K-12 educators and give students exposure to CT concepts and coding practices in BC. When the BC new curriculum was introduced in 2016, the government provided a funding of six million dollars to support introducing coding in the curriculum (BC Gov News, 2016). The federal government, with its CanCode initiative, also funded projects from many organizations that support Canadian youth and teachers in learning digital skills such as coding (Minister of Innovation Science and Economic Development, 2019). Science World BC’s Tech-Up project has delivered professional development workshops and teacher summer training with the theme of CT since 2018. Other projects supporting BC teachers include Canada Learning Code, Kids Code Jeunesse, Cybera and the Pacific Institute for Mathematical

Sciences, etc. However, it is unclear how many teachers have benefitted and how widely spread CT and coding are in BC classrooms.

3.3. Research Setting and Participants

To better understand BC mathematics teachers' current experience, it is crucial to directly ask teachers about their perceptions and examine what happens inside classrooms. The research questions are pursued through surveying teachers, observing in classrooms and interviewing teachers after observations. The research is set in two urban school districts in BC; participants are all secondary mathematics teachers in these two districts. The data were collected from the spring and fall terms of 2019. The school districts, as well as SFU's Office of Research Ethics, gave approval to conduct the research. The data collected included:

- Online survey administered to secondary mathematics teachers
- Field notes from classroom observations
- Audio recordings of teacher interviews

The surveys were sent out to contacts at all secondary schools in the two school districts. After a three-month survey collection period, 19 responses by teachers from 13 different schools were received. Three teachers participated in the classroom observations and interviews. The survey respondents teach a variety of mathematics courses, and their teaching experience range from 3 to 24 years.

3.4. Teacher Survey

The survey was administered online using SFU's online survey instruments. It serves two purposes: 1) to gain information on teachers' background, understanding of the concept of CT and their current practice 2) to search for teachers to participate in the classroom observation and interview portion of the study. Invitations were sent to school principals, department heads, district mathematics coordinators, as well as colleagues I connected with. The survey questions are listed below. The actual survey in the format the participants saw in the online survey tool is included in Appendix A.

There are two types of questions, closed questions, such as rating or ranking questions, as well as open-ended responses. The survey begins by asking the participating teachers for some background information and offers them an opportunity to indicate whether they are willing to participate in further study. The background questions include:

- Which school do you teach at?
- How many years have you been teaching?
- What grades/courses do you currently teach?
- What other grades/courses (if any) have you taught in the past 3 years?
- Would you be willing to participate in further study after the survey, which may involve a classroom visit and an interview?
- If you are willing to participate in further study, what would be the best way to contact you?

The next part of the survey connects the idea of CT and coding to the latest BC curriculum. The first question in this section gathers information on the teacher's familiarity with different tools. A four-point scale with "I Have Used This", "I Would Like to Try This", "I Have Heard of This" and "I Have Not Heard of This" as headings is used. This question also provides insights for the resource system component in the adapted SFCPF, which was discussed in 3.1.

Question: Please indicate your familiarity with the tools for incorporating coding and computational thinking in mathematics classes.

1. Unplugged activities, such as puzzles and board games
2. Text-based coding, such as Python, JavaScript, Java, etc.
3. Blocked-based coding, such as AppInventor, Scratch, Snap!, etc.
4. Web-based resources, such as Code.org, Hour of Code, Khan Academy, etc.
5. Physical computing tools, such as Arduino, 3D printers, micro:bits, Raspberry Pi, etc.
6. Dynamic geometry software, such as Desmos, Geogebra, The Geometer's Sketchpad, etc.

7. Mathematics-based software, such as Maple, Mathematica, MATLAB, programmable calculators, etc.

8. Robots: Dots and Dash, LEGO Mindstorms, mBot, Spheros, etc.

9. Others. Please specify.

As there are a variety of tools that can be used for teaching CT and coding, they are grouped into eight categories. There is an option for teachers to provide suggestions of tools in the “others” category. After the survey is administered, it is found that for option six, Dynamic Geometry Software (DGS) does not accurately categorize the applications mentioned. Dynamic Mathematics Environment (DME), which is a broader term and does not only focus on geometry, will be used in the analysis in later sections. If this survey is used for further studies, DME will be used instead of DGS.

The following two questions connect to the core competencies, curricular competencies and content areas in the BC curriculum. The feature of core competencies is a major change to the redesigned curriculum; core competencies are embedded in all courses and include thinking, communication and social and personal competencies. The curricular competencies and content areas are both subject-specific. The first survey question focuses on the competencies, whereas the second question concentrates on the content, specific topics in mathematics.

Question: Please select the top 5 competencies in the BC curriculum you find most relevant to coding and computational thinking.

Core Competencies

- Communication
- Creative Thinking
- Critical Thinking
- Positive Personal and Cultural Identity
- Personal Awareness and Responsibility
- Social Responsibility

Curricular Competencies

- Reasoning and Analyzing

- Understanding and Solving
- Communicating and Representing
- Connecting and Reflecting

Question: Please select the top 2 topics you feel are most relevant to coding and computational thinking in math classes.

- Number Sense
- Patterns and Relations
- Spatial Reasoning
- Statistics and Probability

The last section consists of open-ended questions.

- How would you describe your background and comfort level in teaching coding and computational thinking?
- How would you describe computational thinking?
- When you see coding as a suggested learning activity in BC's new mathematics curriculum, what comes to mind?
- What are some reasons why you may want to incorporate coding and computational thinking in your mathematics classes?
- How frequently do you incorporate coding and computational thinking in your math classes?
- If you have tried including coding and computational thinking activities in your classes, describe an activity that you think was most successful.

3.5. Classroom Observations and Interviews

Participants were selected based on their willingness to participate, familiarity with the topics of CT and coding, as well as their availability. Out of the 19 respondents, 11 of them left their contact information. In the end, three teachers participated in the classroom observations and interviews.

Each teacher selected for the classroom observation chose one lesson to be observed after discussion, and the observation lasted for the duration of the class period, usually around seventy-five minutes. Each observation was followed by a reflection interview.

During the classroom observations, notes were taken on the classroom environment, teacher resources, structure of the lesson, tasks used, as well as highlights of the lesson content. Then field notes were organized based on the adapted SFCPF.

The post-observation interview questions are listed below and included in Appendix B. Participants' responses to existing questions sometimes prompted additional questions or clarifying questions. The interview started with some general questions about the teacher's overall experience:

- Overall, what did you like about this lesson?
- What challenges did you have, if any?
- If you can make one change to the lesson, what would it be?

Then the questions were organized into five categories based on the adapted SFCPF. For the working environment, the question asked about one change that the teacher could make with the intent to highlight the teacher's major concern.

- If you can make one change to your classroom setup for this lesson, what would it be?

The questions for the resource system purposely separated resources for CT and coding learning outcomes from mathematics learning outcomes.

- What resources for coding and computational thinking did you use for this lesson?
- What resources for the math learning outcomes addressed did you use for this lesson?

Questions for the activity format and curriculum script again highlight the comparison to the teacher's typical mathematics lesson:

- How did you structure this lesson?

- Comparing to your typical math lessons, what do you think are some differences in your lesson that incorporated coding, such as in the sequence of the lesson, assessment, etc?
- What are some changes you made in your way of teaching, such as questions posed, explanation of a concept, etc. teaching the same math concept in your lesson that incorporate coding and computational thinking?

Finally, questions on time economy focused on how much time the teacher spent to prepare the lesson and the teacher's perception of whether the use of time in class was effective.

- How much time did you spend planning this lesson?
- Are there parts of the lesson you feel were not effective use of time due to the use of technology?

3.6. Data Analysis

The teacher survey consists of closed questions and open-ended responses. For closed questions, I provide descriptions of the resulted statistics. For open-ended responses, I find emerging themes and attend specifically to issues discussed in the related literature in Chapter 2, such as how teachers view CT, how they talk about the relation of mathematics and CT, and what type of CT and coding tasks they use.

For the class observations and interviews, I analyze data from each individual lesson using the adapted SFCPF. Field notes from the classroom observation are re-organized according to the five structuring features. The post-observations interviews are audio recorded and then transcribed; the information supplements the classroom observations. Data from each individual lesson are first analyzed individually, and then cases are cross-examined for similarities and differences.

Chapter 4. Analysis of Teacher Survey

In this chapter, I first provide background information about the teachers participating in the survey. Next, I discuss the results on resources for teaching CT and coding. What follows are how teachers describe CT and how they connect it to the BC mathematics curriculum. Then, I present teachers responses motivations and challenges for incorporating CT and coding. Last, I discuss the various examples of CT and coding activities teachers provided.

4.1. Teacher Backgrounds

There is a good mix of novice and experienced mathematics teachers among the 19 participants. They teach a variety of mathematics courses from grade 8 to 12, including calculus. Two teachers also teach science. In addition, two teachers teach ADST courses such as Computer Programming 11, Digital Citizenship/ADST 8 and Accounting 11.

Table 2 below shows a summary of teacher backgrounds based on their years of teaching experience and comfort level with CT and coding. The comfort level is broken into 3 categories primarily based on teachers' answers to survey question 10, which asks for the description of the respondent's background and comfort level in teaching CT and coding. I chose the open-ended question format to allow for elaboration; however, as the responses presented teachers' own perceptions, they could be open for interpretation.

Table 2. Summary of Teacher Backgrounds

Years of Experience	Number of Teachers	Comfortable	Somewhat comfortable	Not Yet comfortable	Have tried coding in math class
1-5	4	1	3	0	2
6-10	6	1	3	2	1
11-15	4	1	2	1	2
16-20	3	2	1	0	1
21-25	2	0	1	1	1

Teachers comfortable with CT and coding all stated it very clearly, whereas some teachers that do not have as much experience with CT and coding may give responses that require further analysis. Some responses are easier to categorize as *somewhat comfortable*, for instance:

My comfort level for this topic will be 3 out of 5.

Comfortable teaching it with guidance, not comfortable alone.

For more ambiguous responses, I looked at answers from other survey questions, such as the teachers' knowledge of a variety of tools (question 6), how frequent they use CT and coding activities (question 14) and if they have tried activities in their classroom (question 15).

One teacher's response to question 10 was just "entry level." According to the response to question 6, the teacher has used unplugged activities and block-based coding tools. Also, the teacher claimed to have not done CT and coding often enough and would like to do more. Moreover, the teacher gave the example of using strategy games in class for question 15. The teacher seems to be confident and at least *somewhat comfortable* in teaching CT and coding.

In another example, the teacher's response was "I have become more interested in this topic and have been learning about it recently." According to the teacher's answers to question 14 and 15, the teacher has not yet tried any CT and coding activities. For question 6, other than dynamic geometry software, the teacher indicated no experience using any other tools. Based on the information, the teacher was categorized as *not yet comfortable* teaching CT and coding.

Contrary to the common belief that younger teachers are more interested in the topic, both the comfort level and experience level were fairly spread out among the 19 participating teachers. The summary in Table 2 showed that there are teachers trying coding activities in mathematics classes, despite that they may only be somewhat comfortable with CT and coding. Study by Gadanidis (2017) also found that teachers with little coding experience can successfully integrate CT in mathematics. Two of the teachers surveyed showed their willingness to learn coding and incorporate it in their classes:

> I have no background in coding, and learned through collaboration with other math teachers who have some background in coding/programming.
>
> I am relatively new to teaching coding/computational thinking. I've done hour of code with my classes for few years, and last semester was the first year for me to include a mini unit (3 classes) of coding into my math 8 classes.

Several teachers mentioned that they had experience with programming from a long time ago, often from their undergraduate years.

> Some coding experience but so long ago it doesn't feel relevant.
>
> My only experience coding is [half-heartedly] teaching myself the basics of LaTex one semester in Undergrad. So I have little background, and awareness of how to build it into the curriculum beyond Excel.
>
> [I] have taken 1st year computer science courses. [I] have done coding in Microsoft excel for work.

Some of the tools mentioned such as LaTex and Microsoft Excel (or spreadsheets in general), involve coding for specific purposes. This type of coding is different from coding or programming in the discipline of Computer Science discussed in Chapter 2 but is considered by a few teachers as a good way to introduce coding.

4.2. Resources for Computational Thinking and Coding

Table 3 below shows a summary of teachers’ familiarity with tools for CT and coding from survey. It is important to note that these tools may be utilized to support teaching CT and coding, but it depends on what tools are used and how they are used.

Table 3. Summary of Teachers' Familiarity with CT and Coding Tools

	I Have Used This	I Would Like to Try This	I Have Heard of This	I Have Not Heard of This
Unplugged activities, such as puzzles and board games	16	2	1	0
Text-based coding, such as Python, JavaScript, Java, etc.	3	7	9	0
Blocked-based coding, such as AppInventor, Scratch, Snap!, etc.	6	8	4	1
Web-based resources, such as Code.org, Hour of Code, Khan Academy, etc.	8	6	3	2
Physical computing tools, such as Arduino, 3D printers, micro:bits, Raspberry Pi, etc.	6	5	7	1
Dynamic geometry software, such as Desmos, Geogebra, The Geometer's Sketchpad, etc.	18	1	0	0
Mathematics-based software, such as Maple, Mathematica, MATLAB, programmable calculators, etc.	9	6	4	0
Robots: Dots and Dash, LEGO Mindstorms, mBot, Spheros, etc.	4	6	7	2

Most teachers are familiar with unplugged activities and dynamic mathematics environment, which are already commonly used in mathematics classrooms. As discussed in section 3.4, the wording is changed from dynamic geometry software (DGS) to dynamic mathematics environment (DME) to more accurately reflect the tools teachers are using.

For teachers that have tried coding, block-based programming and online tutorials were popular choices among teachers. Three teachers that have used text-based coding all teach Computer Science or Computer Programming; only one of the them had the opportunity to introduce text-based coding in his or her mathematics classes. The activity will be discussed in more details in section 4.6.

Two teachers indicated Microsoft Excel as a tool for incorporating CT and coding. One teacher elaborated in his/her comment:

> I've been working on building opportunities for students to practice using Excel in Math 10 (budgeting, functions, etc.) as a soft introduction to coding through applying cell functions.

One teacher used Explain Everything, a software application with an interactive whiteboard that allows for collaboration. The teacher considered it a tool for teaching CT as it involves everyday problem-solving skills.

> Explain Everything on iPads documenting patterning and work with physical manipulatives. Students needing to think about order and orientation to document the concept or present a problem solution.

The additional examples of Microsoft Excel and Explain Everything raised interesting questions. How do teachers interpret CT? Is entering formula into Microsoft Excel considered coding? How teachers define CT determines what resource tools they see as CT and coding tools.

4.3. How Teachers Describe Computational Thinking

The concept of CT, as discussed in Chapter 2, lacks a precise definition that is commonly adopted by the education community. I am interested in how BC secondary mathematics teachers understand and interpret the term CT. The wording of the survey question was "How would you 'describe' computational thinking?" instead of "'define' computational thinking" as this is anticipated to be a challenge.

Among the 19 responses, ten used the problem-solving rhetoric similar to Wing's description; some sample responses are:

> Problem solving using systematic approach

> Being able to identify what problem needs to be solved, and create an algorithm or equation that can be used to help solve it.

> An algorithmic approach to problem solving.

Most of the other answers mentioned one or a few aspects of CT, for example:

> Generalizing patterns

> breaking down a problem into solvable chunks and pathing it together

> Thinking logically of the vast possibility of course combinations. Step by step thinking, trial and error, a vehicle for teaching growth mindset

> the ability to write down all the possible scenarios in a situation and be able to follow a path to each possible outcome (like a tree diagram)

As discussed in 2.3, there has been a movement towards using unplugged activities. Six of the 19 teachers mentioned the word “computer” in their descriptions of CT, whereas one teacher specifically stated that teaching CT does not necessarily require technology. There were some differences from the descriptions where “computer” was included. The first three descriptions below focused on using the computer as a tool for problem solving.

> Using computer as aid for problem solving
>
> solving problems by writing programs/expressions that a computer software understands and executes?
>
> Identifying a pattern or breaking a task or problem down into a series of steps that can be implemented into an algorithm and executed by either a computer or another autonomous system.

Two descriptions emphasized on communicating with the computer; using the computer was the end goal.

> It is a method to break down a task [so] that a computer is able to execute the task.
>
> Understanding the language to communicate to the computer to get it to do things you want it to do.

One description echoed Wing’s message that CT means “thinking like a computer scientist” (2006).

> "computer" thinking - which involves the use of logic, analytical thinking, communication, problem solving, pattern noticing/recognizing, etc.

Three descriptions stood out as unique. Two of the responses did not seem to be influenced by the frequently referenced papers by Wing. The first description, connecting “computational” to the use of quantities, is what I originally anticipated from mathematics teachers. Some of skills such as pattern creations and algorithmic thinking are common skills in both mathematics and computational thinking.

> The understanding and use of quantities to represent the properties and actions of objects and their relationship to each other. The numerical objects are then combined, ordered and transformed to create patterns, new objects, actions or sets of procedures.

The second description focused on the idea of design thinking, which is more inline with the BC ADST curriculum.

> I would assume it's related to design process thinking. Where you have a goal based on a certain set of needs and you need to determine the set of steps/other requirements to attain the goal.

While it is quite common for coding to be considered as a vehicle to teach CT, one teacher thought otherwise:

> Coding to me doesn't quite fit under computational thinking because I think of coding as a specific way of communication (using a specific type of language/ways to express an idea - almost like a specific "way" to solve a problem).

In this comment, coding was viewed as a way of communication or expression. I wonder if this has to do with the rise of block-based and visual programming, or if the teacher thought of coding as writing code in languages such as the Hypertext Markup Language (HTML) and Cascading Style Sheets (CSS) to create a web site, which involves more creativity.

Overall, the teachers had varied responses, but most of them were under the theme of problem-solving. Few CT concepts were mentioned, such as breaking down steps (decomposition) and step-by-step thinking (algorithmic thinking) and pattern recognition (generalization). One teacher responded with “limited experience” and two indicated the lack of confidence in providing a description but still made attempts.

4.4. How Teachers Connect Computational Thinking and Coding to BC Mathematics Curriculum

According to the survey results, the teachers associated CT and coding mostly to the core competencies of creative thinking and critical thinking. For curricular competencies, the teachers connected CT and coding to the curricular competencies of reasoning and analyzing, as well as understanding and solving.

Table 4. Survey Results of How Teachers Connect CT and Coding to Core Competencies

Communication	4
Creative Thinking	16
Critical Thinking	14
Positive and Personal Cultural Identity	1
Personal Awareness and Responsibility	0
Social Responsibility	3

Table 5. Survey Results of How Teachers Connect CT and Coding to Curricular Competencies

Reasoning and Analyzing	18
Understanding and Solving	15
Communicating and Representing	9
Connecting and Reflecting	9

In terms of curricular topics, all 19 teachers chose “patterns and relations” as the most relevant to CT and coding. This is anticipated, as generalization or pattern recognition is considered one of the characteristics of CT by Selby and Woollard (2014). Number sense and spatial reasoning tied for the second highest. It is surprising that with the mention of experience with DME in the classroom, spatial reasoning is not associated with CT as much. This is interesting because the root of CT in mathematics education started with Papert’s (1980) “computational style of geometry” as described in 2.4 (p. 55). The results may imply that teachers are referring to the use of the graphing features in DME when they teach algebra, and teachers’ responses to question 15, which asks teachers to provide descriptions of a CT or coding activity they considered successful, also confirm this idea.

Table 6. Survey Results of How Teachers Connect CT and Coding to Curricular Topics

Number Sense	8
Patterns and Relations	19
Spatial Reasoning	8
Statistics and Probability	3

4.5. Motivations and Challenges

According to the survey results, there were a variety of reasons why teachers may want to incorporate CT and coding in mathematics classes. The most mentioned reason was that it supports mathematics learning. As discussed in section 2.5, studies have shown mixed results, but many teachers have the impression that incorporating CT and coding supports mathematics learning. Some felt that CT and coding have direct connections to mathematics learning:

> helps with algebra (input to output). helps with looking at situations in a systematic way

> I think there are definite direct applications with graphs and their equations.

> Creating meaningful connections for math concepts that now have a context where the use is clearly valuable. Another opportunity to practice problem analysis, reasoning and procedural ordering of steps.

It was mentioned in several comments that incorporating CT and coding makes mathematics more relevant and fosters problem-solving skills. Some teachers considered it a way to develop skills transferable to mathematics classes:

> I would assume that it would help students to get better at logic and problem solving which would translate to other parts of mathematics.

> The step-by-step thinking ties in with math.

Two teachers discussed how the nature of coding supported the growth mindset in learning. One of the teachers elaborated in the following comment, which echoed the idea by Papert that from debugging their programs based on immediate feedback provided by the computer, students are not afraid of making mistakes when they program:

> Students need help to see that making mistakes is part of the development and learning process. This is a natural idea when students need to debug a program and somehow does not have the "shame" associated with it that getting a math problem wrong does.

Six teachers claimed that CT and coding are useful skills to have in real life, and four teachers mentioned the career prospects associated with CT. Four teachers thought it would be fun or exciting for students. Other reasons included increasing the variety of classroom activities, helping students understand technology better, promoting CS and increasing enrollment for the new Computer Science 11 course, etc.

Despite potential benefits, about half of the surveyed teachers stated that they have rarely or not yet incorporated CT and coding in their classes. I asked, in survey question 12, what comes to mind when teachers see coding as a suggested learning activity in BC's new mathematics curriculum. Though many agreed that this is a positive change, some expressed concerns:

> It makes me think they want actual coding languages to be taught (which then makes me wonder how they want that done since we're math teachers, not comp sci teachers, and would need training or education to do this!)
>
> where do i put it in my curriculum? what simple resources can i quickly implement in my classroom? do i need to buy expensive coding "toys" for my students to use?
>
> In theory, I do like the idea. However, [based] on the current status in my classrooms, I don't know if there is time to in cooperate (incorporate) coding into my classroom.
>
> worried about resources to support (computer lab availability, reliable network)
>
> Teach to what extent/level Any connection between coding and other topics in the curriculum. Any resources available to teachers who do not have background in coding?

Time, resources and training were the main concerns by teachers. In the curriculum document for mathematics courses (except in Computer Science 11 and 12), coding was included without elaboration. Two of the teachers specifically stated that some clarity on how to connect coding with the curriculum would be helpful.

4.6. Computational Thinking and Coding Activities in Mathematics Classrooms

There are a variety of activities teachers have been using in their classrooms. 13 teachers provided responses to question 15 and described a CT or coding activity that they felt was most successful in their classes. Details of three of the examples that I had the opportunities to observe will be discussed in Chapter 5.

Five teachers used activities with block-based coding as examples. For instance, a teacher let his/her grade 8 students create a number guessing game using Scratch, and the game is required to keep track of the number of guesses and let users know when they get the correct number. The teacher specifically highlighted that students need to plan first by writing procedures on paper before starting to program in Scratch.

Another teacher found using Sphero robots a fun and engaging activity. Students watch online video tutorials from Sphero Education and learn how to move the robot and draw squares. Students then are tasked with drawing other shapes such as equilateral triangles and hexagons. As a final task, students complete a maze. The activity is very similar to what people used to do with LOGO, but Sphero offers a much friendlier block-based programming interface.

One teacher with extensive programming experience uses a text-based coding activity with grade 9 students. The teacher chose the programming language Basic and the Basic256 programming environment, which is an open-source program designed to teach introductory programming in Basic. In this activity, students convert algebraic expressions into code in Basic and run the computer program to evaluate these expressions with values user enter. The teacher utilizes word problems from the textbook as the programming exercises. In addition to writing the formulas and converting them into code, students also need to create test cases to verify their work.

Six teachers gave non-coding examples of mathematical investigation activities using the DME Desmos. Teachers provided examples of student exploration by creating drawings and investigating different type of functions. Two sample responses were shown below:

> Desmos investigation into polynomial functions, very engaging, moving visuals, students could work at their own pace.
>
> Desmos activities where students see the connection between the equations they write (the language) and the functions that show up on the screen. They learn to adjust their equations and understand conventions (e.g. 3^2 means three squared) that Desmos graphs use.

These teachers have perspectives similar to Pei et. al. (2018); activities in a non-programming environment such as an DME, which is already commonly used in mathematics classrooms, can support the development of CT skills. There is a great overlap of mathematics habits and CT skills. DME is an interesting case. Barr and Stephenson (2011), in their examples of how CT can be embedded in mathematics, listed the Geometer's Sketchpad as a tool along with a LOGO-like language and Python code snippets, under the categorization of automation, which is not normally understood as a CT skill in mathematics. This showed that how DME is connected to CT can be interpreted very differently.

One teacher expressed the concern that, while the activity was engaging, students may be solving problems by using guess-and-check. This also was seen in the research by Savard and Freiman (2016), who indicated that students in their study, upon receiving feedback from digital technology, heavily relied on the trial-and-error strategy, which could limit students from thinking more deeply. The example provided in the teacher survey, the Linear Marble Slide activity, is an explorative activity where students adjust the linear equations, by changing the coefficients or the constant terms, to move the lines on the graph (the “marble slides”) to the desired positions and orientations. The teacher indicated in the survey response that this was not considered a successful CT activity; because students, through this learning-by-tinkering approach, did not use the mathematics learned to accomplish the task.

> I am at the stage where I don't think any of my attempts have been successful to the degree where I would like them to be. For instance, many of my students have enjoyed working with the Linear Marble Slides activity, but I've found that more often than not they are using strategic guess and check rather than the knowledge they have "learned" about equations of straight lines. They can answer the other questions, but struggle with the "connect & reflect" competency, unless the connection is made explicit.

One teacher described an unplugged “human coding” activity, which is often seen at teachers’ professional development workshops on introduction to coding:

> human coding - 3 ppl/grp person 1 - write instructions for a task person 2 - read out the instructions for the task person 3 - carry out the instructions in attempt to complete the task

Other activities mentioned including board games, strategy games, guest speakers and field trips.

Overall, teachers’ responses indicated that most of them understood CT as about problem-solving. They do not often incorporate CT and coding in classrooms but have tried a variety of tools, such as block-based coding and DMEs. Many teachers in this study believe that incorporating CT and coding support mathematics learning; however, time, resources and trainings are some of the barriers that prevent them from including CT and coding in their classes. In the next chapter, I present results from classroom observations and interviews with three of the teachers.

Chapter 5. Analysis of Individual Lessons

In this chapter, I first provide background information about the three participating teachers and introduce the tools they used in the lessons observed. What follows is the analysis of each teacher's lesson based on the adapted SFCPF discussed in 3.1, as well as discussions of each teacher's overall experience.

5.1. Teacher Backgrounds and Tools Used

As part of this study, I had the great pleasure of working with three teachers, Ada, Grace and Julia. I reached out to nine teachers who had provided contact information in the survey, including three male teachers and six female teachers. It was by coincidence that all three teachers available for further study are female. The data used for analysis in this chapter mainly come from the classroom observations and interviews, and much of teachers' background information was from the teacher survey.

The teachers' background information, such as years of teaching experience, comfort level with CT and coding and what motivate them to incorporating CT and coding in mathematics classes, was collected when they filled out the survey, Ada and Grace have both taught for about six years, and both indicated that they are relatively new to coding. Ada mentioned that she did some programming a long time ago, so the programming languages and skills did not seem relevant. Grace has used online tutorials from Hour of Code, and it was her first year to include a mini coding unit which consisted of three lessons for her Mathematics 8 course. Julia is a more experienced teacher, having taught for 17 years. She said she felt quite comfortable with coding and would like to find natural ways to incorporate it into mathematics classes.

In all three cases, the teachers utilized block-based coding tools. The two tools used by the three participating teachers were Snap! and micro:bits. Both are simple yet powerful tools for teachers to incorporate coding in the classroom.

Snap!, previously known as Build Your Own Blocks, is an extended version of Scratch and is a free, block-based educational graphical programming language. Similar to Scratch, the Snap! editor is web-based and does not require any installation. As shown in Figure 1 below, the interface contains an area for block commands on the left, and in the middle is an area for the script, where blocks can be dragged onto to make programs. The object users program is called a sprite, which is the computer graphics element that the code is applied to. The sprite is like the turtle in LOGO – it is the object to think with. The default sprite in Snap! is an arrowhead. The canvas on the top right is where all the actions take place, based on how the sprite is programmed; the pane below consists of features such as changing the sprite costume, updating the background and adding sound. Snap! is supported by UC Berkeley and is used for the university's introductory CS course for non-CS major students.

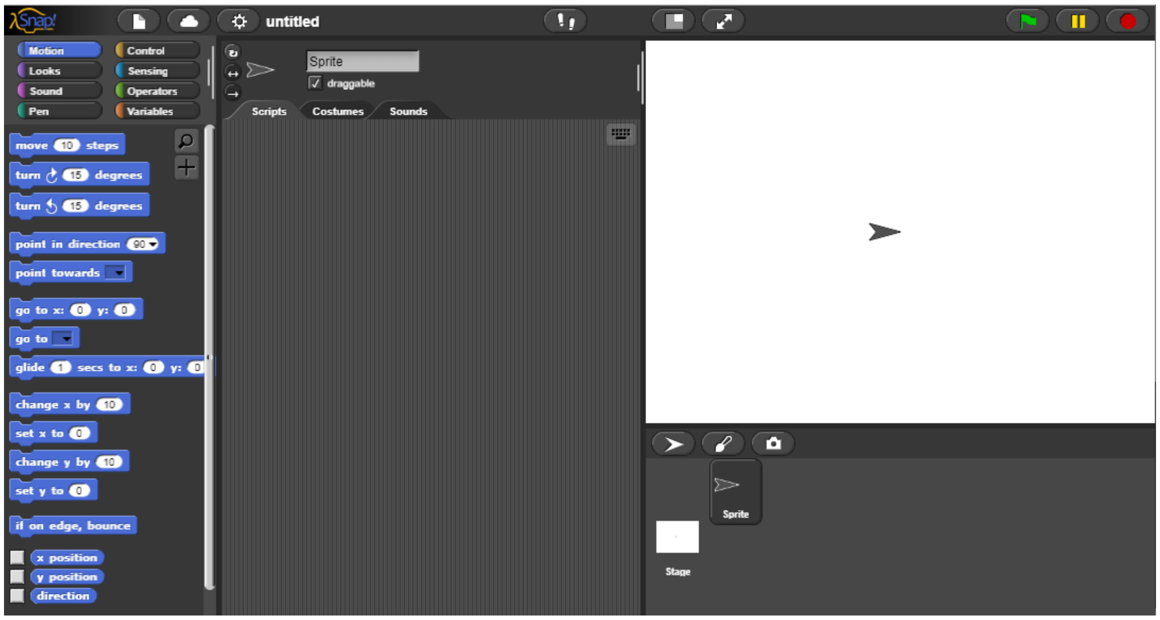


Figure 1. Block-Coding Interface of Snap!

Micro:bit, or the BBC micro:bit, is a small programmable device with an LED display. It is designed by BBC as a part of its "Make it Digital" initiative that promotes computer science education (Cellan-Jones, 2015). Coding with micro:bits can also be done in web browsers, either with the Microsoft MakeCode editor, which has a drag-and-drop block coding interface, or with MicroPython, a Python programming environment designed to run on a microcontroller. Figure 2 below shows the MakeCode interface for micro:bit. On the left is a simulator of the micro:bit device; in the middle is a list of categories for block commands; on the right is the script area for the code.

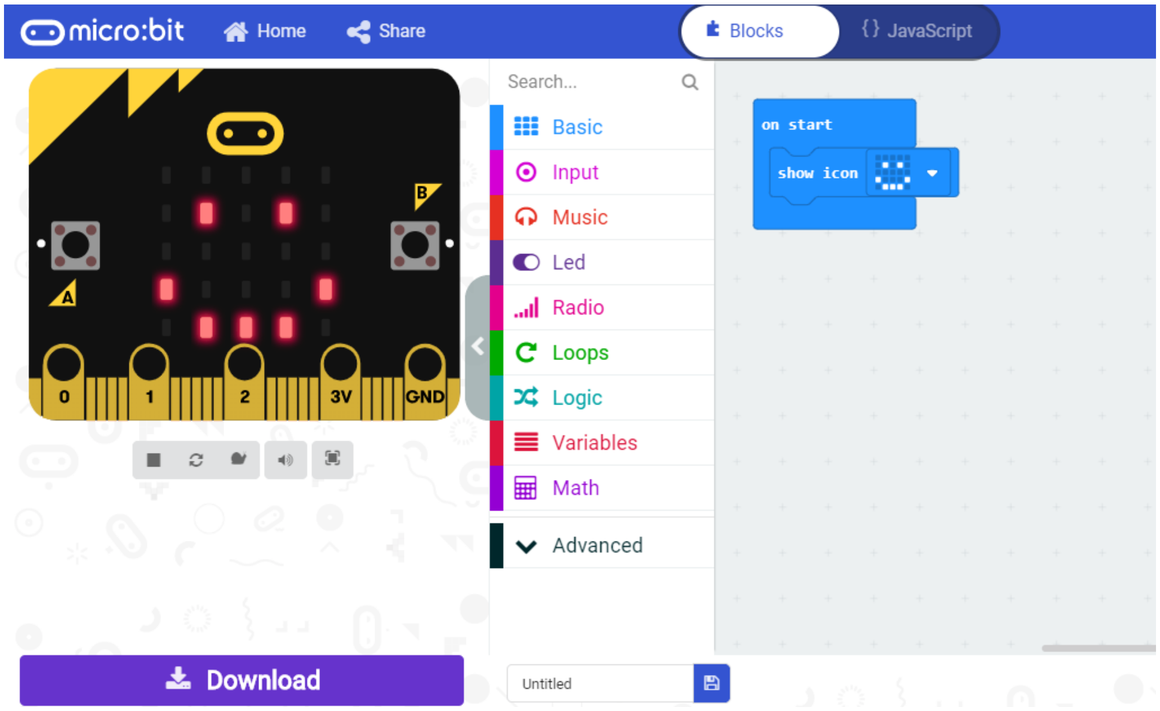


Figure 2. **Block-Coding Interface of micro:bit**

5.2. Ada's Lesson

The coding activity Ada used was for her Mathematics 9 course, and Snap! was the tool of her choice. The lesson was co-designed by a colleague I collaborated with during the pilot of this study and myself. The lesson was a stand-alone lesson that was an addition to Ada's unit on similarity, and the class was primarily a lab session without direct teacher instruction. The coding activity, the format of which is discussed later in this section, involved students applying what they learned about angles in regular polygons and repeatedly drawing shapes to make spirals. Ada made additions to the lesson plan to tie in the topic of similarity. Based on Ada's responses to the survey, she wanted to include coding in her class because she sees it as opening up career paths for students and increasing interest in the school's new CS 11 course. Ada also believed that CT and coding promote accuracy, rigour and creativity in student work.

Working Environment

Ada booked the computer lab for this lesson. The lab is very spacious, and there are 30 computers, set up in six rows. It was quite easy to circulate around in the classroom to answer students' questions. Although it is now common to have classrooms with built-in projectors, the computer lab at Ada's school was not equipped with a projector. She had the option of booking a portable projector, but she decided not to go through the trouble and ran the lesson without one. In the interview after the lesson, we had a discussion about how not having a projector affected her lesson:

> EH: And in terms of the classroom environment, the lab environment, if you could make a change, or several changes, to the classroom setup, what would it be?
>
> Ada: The physical setup. I would definitely have a projector.
>
> Ada/EH: (laughs)
>
> Ada: It is quite lacking in our lab.
>
> EH: How would that change your lesson? If you had a projector, how would you do things differently?
>
> Ada: So… I… enforced a buddy system today so that they could look at the color of coding on one screen while working on another computer screen. If I had a projector I would do more teaching rather than more self-guided activities because I could show them the color on the screen at the time and get them to produce a piece of code as a class together at the same time. So I could pace the class with the intro for something on it a little bit easier. I think had I had that today… I might have done that for the first activity, show them the triangle.
>
> EH: Okay.
>
> Ada: And then allow them to go independent for the rest. Although… saying that I think it's very nice for them to get a break from any teacher instruction at the beginning. They got right in to it. I don't think the lack of the projector really held us back at all. So I don't know… I'm almost in two minds as to whether setting them completely free from the beginning… is maybe even better than giving them a bit of direct instruction at the beginning.

With the nature of coding lessons, it is certainly valuable to have a projector. Not having the ability to do a demonstration, which was different from Ada's normal classroom setting, she chose the teaching strategy of having students work as pairs and use two computers to complete the tasks. At first, she stated that she definitely needed a projector. From her remarks, she stated that she would have demonstrated the first task of drawing a triangle as the class. However, after some thoughts she felt that there were advantages of just letting students work without any direct instructions, at least for this particular activity, which was easily accessible without any teacher demonstration.

Resource System

Ada primarily used the original Snap! web site and modified the handout provided to her. She added in tasks related to similar polygons. Ada mentioned that she actually had the idea for quite a while, before I approached her about the study. When she first saw Scratch, what immediately came to her mind was to create a mathematical learning activity on polygons. Snap!, or block-based coding, visual programming environments in general, provided a natural connection between curricular goals and coding for Ada. In Papert's work on LOGO, he also started with drawing shapes and working with angles in Turtle Geometry. Ada elaborated on how her lack of coding experience prevented her from trying earlier:

> I didn't go the spiral route at all, which was quite interesting for me to see, but I definitely thought polygon activities with Scratch. But my lack of coding experience, so it's just this floating idea in a bubble until you introduced Snap! and really made me have some hands-on experience and made me a bit more comfortable with seeing how accessible it could be, that I felt the idea was doable.

The digital tool Snap! provided Ada a way to extend her lessons on polygons and similarity. Students did not learn new mathematics concepts but applied knowledge they learned from previous classes. Ada commented on how visually seeing students work offered an alternative method of assessing student understanding:

> Did [the students] understand the angle measure needs to be the same? Did they understand the side lengths have to be proportional? And yes, because they are making it happen, visually.

The activity handout provided to Ada determined the activity structure of her lesson. The handout was broken down into three parts. The introduction section highlighted key features of the Snap! coding platform. There were five tasks, the difficulty of which gradually increased. Within the tasks there were instructions and questions guiding the thought processes for the students. For example, there were questions asking students what a command does and instructions for students to try run the program without it. These prompts replaced what a teacher normally does in teacher-guided lectures in a typical classroom and allowed students to work on the tasks on their own. Finally, there were two challenges; students needed to get Ada's initials after completing them. These challenges served as check points.

The handout provided to Ada used Snap!; thus the choice of tool was determined for her. When Ada was asked if she would prefer Scratch, she brought up an interesting point:

> The interface [of Snap!] seems easier to me. I don't know if that' true, but for me the interface for Snap! felt easier. And Scratch felt like… fluffy orange cat… sometimes grade 8s and 9s don't like to see things that make them feel like they were in elementary school. They can have quite a barrier to that.

In addition to user-friendliness, Ada also considered the look and feel of the program. The orange cat that Ada referred to is the mascot and default sprite of the Scratch program. In Snap!, the default sprite is an arrowhead. The target of the Scratch program is indeed primarily children, whereas Snap! has been used in introductory CS courses at the university level.

Activity Format

Ada started the class by asking students to pair up and giving them a few minutes to read the instructions on the handouts. She gave students paper copies of the handout and also made the digital version available on her class web site. As the photocopies were in black-and-while, Ada instructed students to display the digital version of the handout on one computer and complete the tasks in Snap! on another computer. The introduction took just under ten minutes, and students had the rest of the class, which was about an hour, to work on the activity. Ada did not do any demonstration because she did not have a projector, and there was no direct instruction for this lesson. Ada circulated around for the rest of the time to answer questions.

The activity started with students figuring out how to draw an equilateral triangle in Snap! with some sample code on the handout. Students continued to draw a square and then were challenged to draw two similar, but not congruent, regular polygons. The polygons were required to have at least five vertices and must not overlap or be connected by any extra lines.

Next, students created repeated patterns using nested loops to make spirograph-like patterns; sample code was shown using triangles. With code, students were able to automate the process of drawing many polygons, or drawing polygons of many sides, which is harder to do by hand. In addition to making their own patterns, students also were tasked to investigate more complex patterns and try to reproduce them.

A few extensions were included on the handout. Originally there was an extension task of playing with a spirograph design program. Ada added more prompts for students to explore the Snap! program, including changing the sprite and investigating commands students did not use in the activity.

There were two check points that Ada included – drawing two similar polygons and drawing two congruent polygons simultaneously. Students were required to get Ada's signature after completing the tasks. When asked about assessment, she commented:

> We've never done a coding class before, at all. I could not confidently say if it was going to go well. I don't want assessment to be a stress interfering myself or them.

Ada added that although the activity format is different from a usual mathematics class, the way she assessed students during this lesson is often how she assesses students in her class.

> I do a lot of formative assessments from interacting with them, watching what they are doing, and from hearing their conversations, so for myself it's just seeing what they are producing, hearing their discussions, and getting a sense of is anyone really struggling, or is the class as a whole really a group together and they are okay with the task. I think they are.

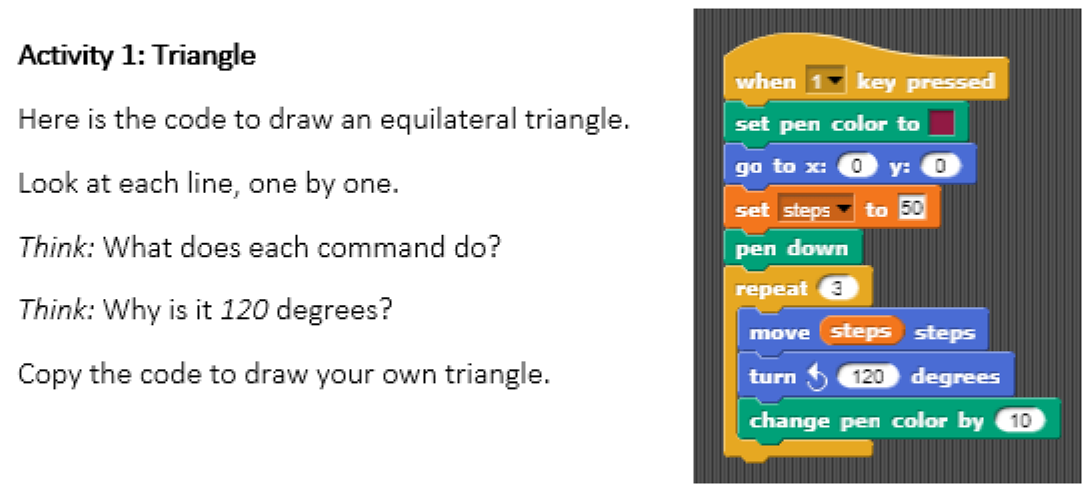
Curriculum Script

The mathematics concepts underpinning Ada's spiral polygons activity were similarity and angles in regular polygons, which students learned previously in class. Students were tasked to draw similar but not congruent shapes, demonstrating their understanding of similarity. They also need to perform calculations of the interior angles to draw regular pentagons, hexagons, etc. On the handout, Ada ensured that proper mathematics terms were used, such as equilateral triangle, similar, congruent, exterior angle, vertices, etc. Reflecting on the lesson, Ada noticed the difference in her conversations with students:

> I think in a regular math lesson we're a little more caught up in the tiny details in our conversation, whereas today it was more of a holistic

overview style of conversation. So for instance, when we were talking about similarity in the classroom, we were very… we talked a lot about the measure of the angles, congruent are the same, and the side lengths are proportional…. And we used those words, you know the ratio, we use proportional, we use angle measures… we didn't use any of that language today. And I did notice that. And I think a lot of that was because… they were….. I didn't need to because I can visually see.

Overall, the students did not have much trouble completing the tasks. The main mathematics concept students had questions about was on exterior angles, especially for the drawing of a regular pentagon. The use of exterior angles was implicitly pointed out in the sample code. As shown in the figure below, students were asked to think about why 120 degrees was used.



Activity 1: Triangle

Here is the code to draw an equilateral triangle.

Look at each line, one by one.

Think: What does each command do?

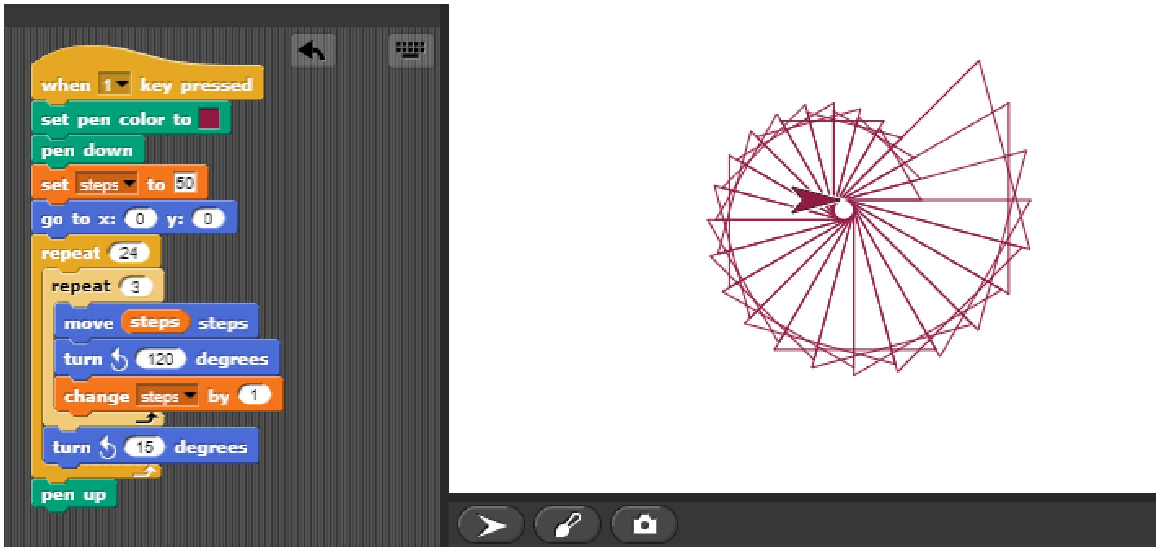
Think: Why is it *120* degrees?

Copy the code to draw your own triangle.

```
when 1 key pressed
set pen color to
go to x: 0 y: 0
set steps to 50
pen down
repeat 3
    move steps steps
    turn ↺ 120 degrees
    change pen color by 10
```

Figure 3. Sample Code in Snap! for Drawing an Equilateral Triangle

There were also other mathematics concepts required that were not directly taught. For example, when students wrote code to repeatedly draw triangles to make a spiral, they needed to keep in mind that a complete rotation is 360 degrees. As shown in Figure 4, after each triangle is drawn, the sprite is turned by 15 degrees, thus it takes 24 triangles to make a full rotation. This was not explained on the handout. Students may have to play with the numbers when they made their own patterns.



```
when 1 key pressed
set pen color to
pen down
set steps to 50
go to x: 0 y: 0
repeat 24
    repeat 3
        move steps steps
        turn ↺ 120 degrees
        change steps by 1
    turn ↺ 15 degrees
pen up
```

Figure 4. Sample Code in Snap! for Drawing Spiral Triangles

Regarding learning objectives for CT and coding concepts, the activity includes several important concepts. The categorization by Brennan and Resnick (2012) outlined in 3.1 is used for this analysis. In this activity, the key concepts include event, sequences, loops, parallelism and data.

Event (using key presses) and sequences (ordering of the steps required to complete the drawing) were utilized throughout. Loops were first introduced in the drawing of regular polygons, and nested loops (a loop within a loop) were demonstrated in the spiral triangles example, as shown in Figure 4. Event, sequences and loops are three key concepts commonly used in simple coding projects. One task Ada added for the activity was asking students draw two congruent shapes simultaneously. This involves the concept of parallelism.

In terms of data, a variable was used to allow the change in the side length of the shape. Ada noticed that this could be challenging for students as a variable in coding is different from a variable in mathematics:

> They stumbled upon the variable being named steps. I think I should have seen that coming a bit more because the variable is normally a letter. Just one letter. And the fact that it had a name meant they weren't clued into the fact that it was a variable.

This highlighted an interesting distinction. As Ada reflected on her experience helping students, she realized that the naming convention is different, thus it may also be hard for students to grasp the concept of a variable in coding at first. Also, unlike in mathematics, the purpose of using a variable is not to represent an unknown or changing value. The purpose is to store a piece of data, in this case the side length of the polygon, so that it can be updated later. Ada commented that a variable can be anything is a concept often missed, and she recognized the need to address that with the students next time.

Time Economy

In terms of time, I noticed that the most of time in the classroom was used to answer technical questions, especially since this was the first time students used the Snap! interface. There was little time spent on directly teaching mathematics. The activity was done after the unit on similarity. Also, prior to this lesson, Ada assigned students homework on investigating angles in regular polygon. She felt confident that students had all the mathematical tools for this activity.

Ada specifically pointed out the challenge with making extensions to the worksheet for students who completed quickly. Some students finished early. I noticed that two students brought up their own coding projects in Scratch when they completed the activity. Ada felt maybe she could have challenged the students more, but she did not have enough knowledge or experience with coding to push students more.

> For the ones that completed the challenges easily because they had a lot of coding experience, they did stay on task playing with coding, and playing with the polygons, which was nice to see, but were they really learning and pushing themselves, I don't think they were. However, did I feel I can guide them, in coding, to an activity to push themselves, no. So my only way for putting students on track are pushing them more mathematically and learning more mathematics would be removing them from coding, which is not what I wanted to do, so I left. I think they did a lot of beautiful, spiral polygons, beautiful polygon art work on coding. Is it something they could have done yesterday? Probably. So extension was lacking on my part, but none of them were off task.

From Ada's remark, it seems like her main goal was to give students exposure to coding. When she had the opportunity to challenge students to explore further in mathematics, she did not want to take away the focus from coding. Overall, she was not bothered by the time spent on using technology, helping students learn the Snap! interface, etc. When asked about whether she felt time spent on technology questions was effective use of her time, she commented that it is the cost of doing an activity involving technology.

Overall Experience

All in all, Ada had a positive experience using the activity. She liked the student engagement and thought the lesson was accessible to all her students. She pointed out that her main challenge was not being able to answer student questions sometimes. Part of it was unfamiliarity with the program. She gave the example of not realizing that the canvas was a finite area, and she figured it out while working with a student.

Ada felt that something valuable coding brought to her class was a different dynamic, when she was learning alongside the students:

> The biggest, most impactful difference, is that I don't know more than some of my students. We have some quite… in my opinions they are expert coders. I don't know if that's true because they are just levels above me, and it makes a completely different dynamic. The students see me as a learner along with them, which I think is very valuable. And I don't think they see enough adults trying to figure out along with them, trying to work with them in solving the problem.

She saw the benefit of modelling problem-solving in the context that she did not know more than the student, as this is uncommon in her regular mathematics classes.

5.3. Grace's Lesson

Similar to Ada, Grace is a younger teacher with little background in coding. Starting the 2018-2019 school year, she began to include in her Mathematics 8 course a mini coding unit that consists of three lessons. She chose micro:bit as the tool for coding. She also collaborated with the other two Mathematics 8 teachers at the school so that all of them teach this coding unit. This was Grace's second time implementing the unit, and the lesson I observed was the third in the unit. Grace stated in her survey that she wanted to incorporate coding in her mathematics classes because it is fun for students. She considers it "a different type of math" and thinks it helps students develop stronger problem-solving skills. Moreover, she believes that introducing CS in junior years can catch more potential students.

In the first lesson of the unit, Grace introduced students to micro:bit and asked students to go through two coding exercises to familiarize themselves with the tool. Students went through the online micro:bit tutorial of Beating Heart (displaying the shape of a heart on the LED display and making it flash) and implemented a simple counter where the number displayed is increased or decreased according to button presses. In the second lesson, students created a rock-paper-scissors (RPS) program by following the micro:bit tutorial. In the last lesson, which was the lesson I observed, students implemented a program called Fizz-Buss and explored other micro:bit tutorials.

Working Environment

The lesson was held in a regular classroom with a school laptop cart that had 30 laptops. Students worked independently but could collaborate with classmates. Desks were put into pods of four to six desks each. The classroom is relatively small, and Grace did point out space being an issue:

> With that particular classroom, there isn't a lot of room for me to walk around, so if having a bigger size would be better for me to really be able to sit beside a student and go over some of the concepts more efficiently, whereas nowadays I felt like I'm hovering students' backs, because of the limited space.

She also mentioned the challenges of using laptops:

> I guess it's not a big thing, but distributing the laptops, I know for sure there are ones that are not going to be working. So handing those out, and then having to make sure that everyone has a laptop that's working, some of them have, what do you call, hmm, use of battery, that doesn't last throughout the class. Those are the little things that add up, and make the lesson not as smooth as I wanted to.

Grace had a few options in terms of computer bookings; she could book the computer lab, library computer stations or a laptop cart. Despite the disadvantage of her classroom size and technical issues with the laptops, she still chose to book a laptop cart. She talked about how being in the same classroom gave her some familiarity, and she believed that it would be better for the students as well:

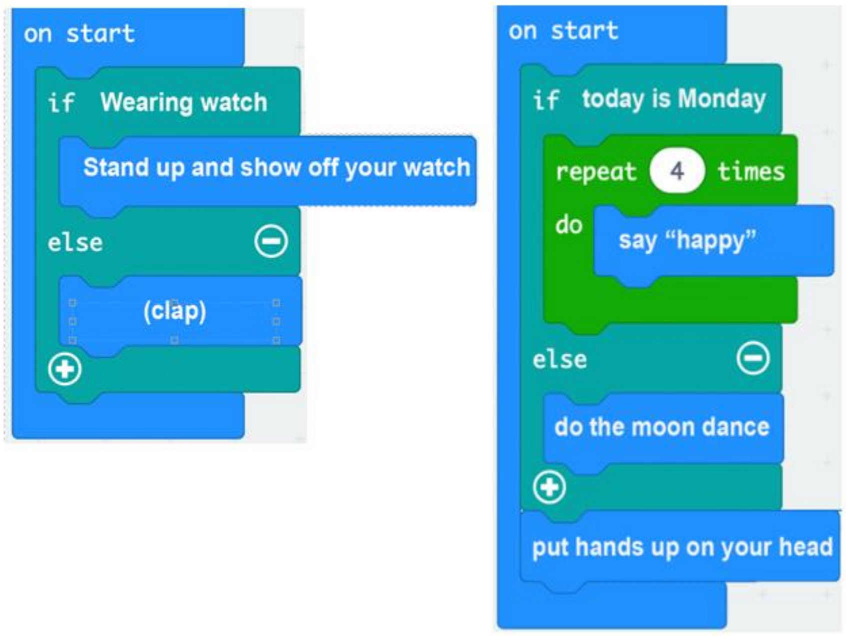
> It was almost like, in a different surrounding, students were way too ecstatic in that different surrounding, and they weren't able to focus on the coding part, and [I was] thinking maybe if they were in a familiar space, it's going to be easier for them.

She also mentioned that it would be great to have laptops in the regular classroom year-round, instead of just a few times a year, recognizing that it may not be feasible. She thought it would be convenient if opportunities arise to relate to some coding, and laptops were available, students could just take out the laptops and try something.

Resource System

In terms of teaching resources, Grace primarily relied the online tutorials from the micro:bits web site. The school has a class set of micro:bits that she could use. While planning the unit, she and two other teachers looked at some videos as teaching resources. For this particular lesson, Grace showed a video featuring Bill Gates explaining the if statement, one of the key coding concepts in this activity, and sharing his experience learning computer programming when he was young. Grace also participated in several coding workshops for teachers. For the main activity of the lesson, she got the idea from a workshop she attended at Science World.

The micro:bit online tutorial was used during the previous lesson, where students followed the instructions step by step to learn the concept of conditionals (if statements) and loops. Here are two examples from Grace's PowerPoint slides:



```
on start
  if Wearing watch
    Stand up and show off your watch
  else
    (clap)
```

```
on start
  if today is Monday
    repeat 4 times
    do
      say “happy”
  else
    do the moon dance
  put hands up on your head
```

Figure 5. Grace's Examples Demonstrating Conditionals and Loops in MakeCode

Grace created instructional materials in the same programming environment MakeCode to show additional examples to reinforce the concepts, and the students, on the day of observation, did a follow-up activity that did not rely on micro:bit's online tutorials. Grace did not want students to just be able to follow coding tutorials but can apply what they learned with a new task.

Grace added that for her other grade 8 class, a university student comes to volunteer. She would then have the ability to get the volunteer to work with a small group of students that finished early and introduce a new task.

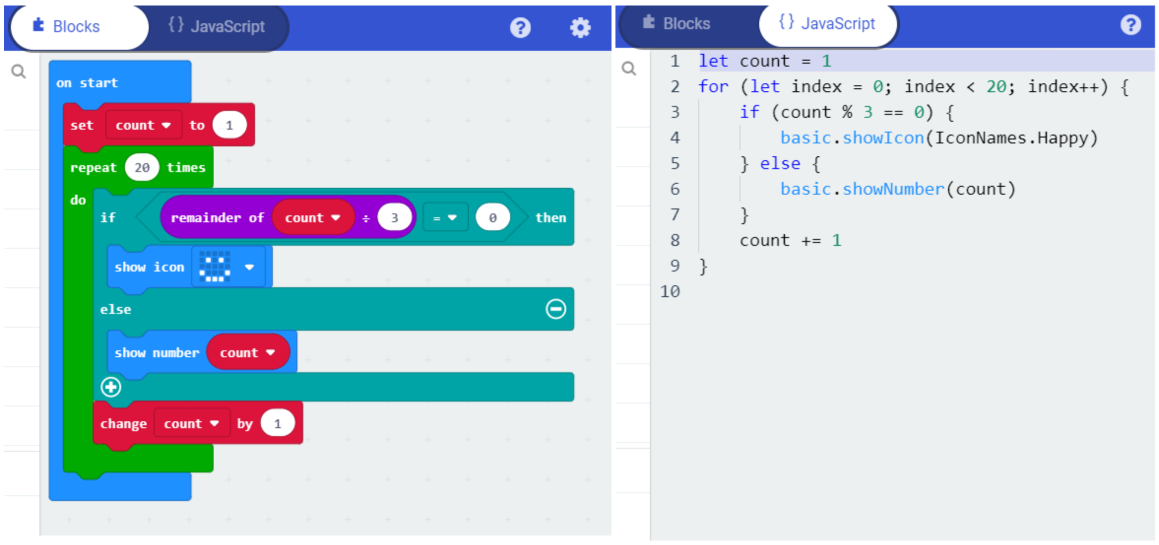
Activity Format

Similar to Ada's class, the majority of Grace's class time was spent on lab work. Grace, however, explicitly taught two of the coding concepts – conditionals and loops. She spent about five minutes doing an exercise with students, explaining how if statements and loops work. Following the explanation, Grace showed a video. Students then started working on the main task after about fifteen minutes of class time, and they had about an hour to work. Grace circulated around to answer student questions during the work period.

The main activity students were working on was Fizz-Buzz. Students programmed the micro:bit to display numbers one to twenty. The micro:bit would display an image (for instance, a smiley face) when the number is a multiple of three and display another image when the number is a multiple of five. When the number is a multiple of both three and five (or fifteen), it would display a third image. Otherwise, it would just display the number. After about 35 minutes of working on the activity, most students got their programs to work. Students had been working with the simulator instead of the actual micro:bit device during this time. Some students that finished early went around the class to helped others. Similar to Ada's class, a few students brought up their own coding projects in Scratch.

When there was about 20 minutes left, Grace showed on the projector screen where to find other micro:bit tutorials, encouraged students to explore and gave a few suggestions. At this time, she also told students they could come and get micro:bits to try their program if they wanted to.

With about five minutes to the end of class, Grace introduced text-based coding and pointed out the JavaScript button in the MakeCode environment, which displays students' programs in JavaScript code instead of the blocks. As shown in Figure 6 below, there are buttons for users to toggle between blocks and JavaScript code. The figure shows the same program displayed in two different modes.



```
let count = 1
for (let index = 0; index < 20; index++) {
    if (count % 3 == 0) {
        basic.showIcon(IconNames.Happy)
    } else {
        basic.showNumber(count)
    }
    count += 1
}
```

Figure 6. Blocks and JavaScript Code in micro:bit's MakeCode Interface

Overall, Grace's lesson used the format of starting with a short teacher-led activity, showing a video, having students work on the main Fizz-Buzz activity and allowing some exploration time. When reflecting on the lesson, Grace felt that one of the main challenge was not having enough for students that finished early to work on:

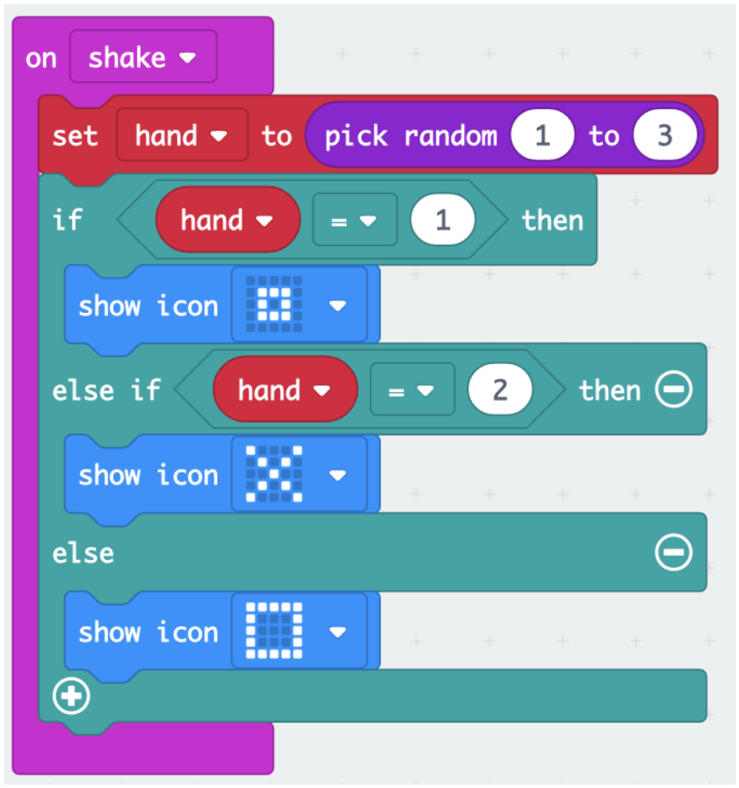
> I felt that there were students done really early, and I just encouraged them to maybe help other students, or do something different with the original code, you know, a little more complex, or… things like that. For example, instead of using the repeat block, try using a while block… So I felt a lot students were done early, and I didn't have a lot to have them work on, in terms of moving forward from there. But for the average student, I think it was a good pace.

Grace did the lesson with all her grade 8 classes. The class I observed was the Mathematics 8 Honours class; this class would get an extra day of coding and the opportunity to create their own projects with micro:bits. Students also had the option of doing text-based coding, following tutorials from CodeHS (https://codehs.com/), an online platform that supports schools teach CS.

Curriculum Script

Unlike Ada's lesson, Grace's coding unit was not tied to a specific unit in her Mathematics 8 course. Students needed to apply the concept of multiples in the Fizz-Buzz activity. Grace mentioned that she briefly reviewed integer division and remainder the day before, but students likely relied on what they already knew to complete the task.

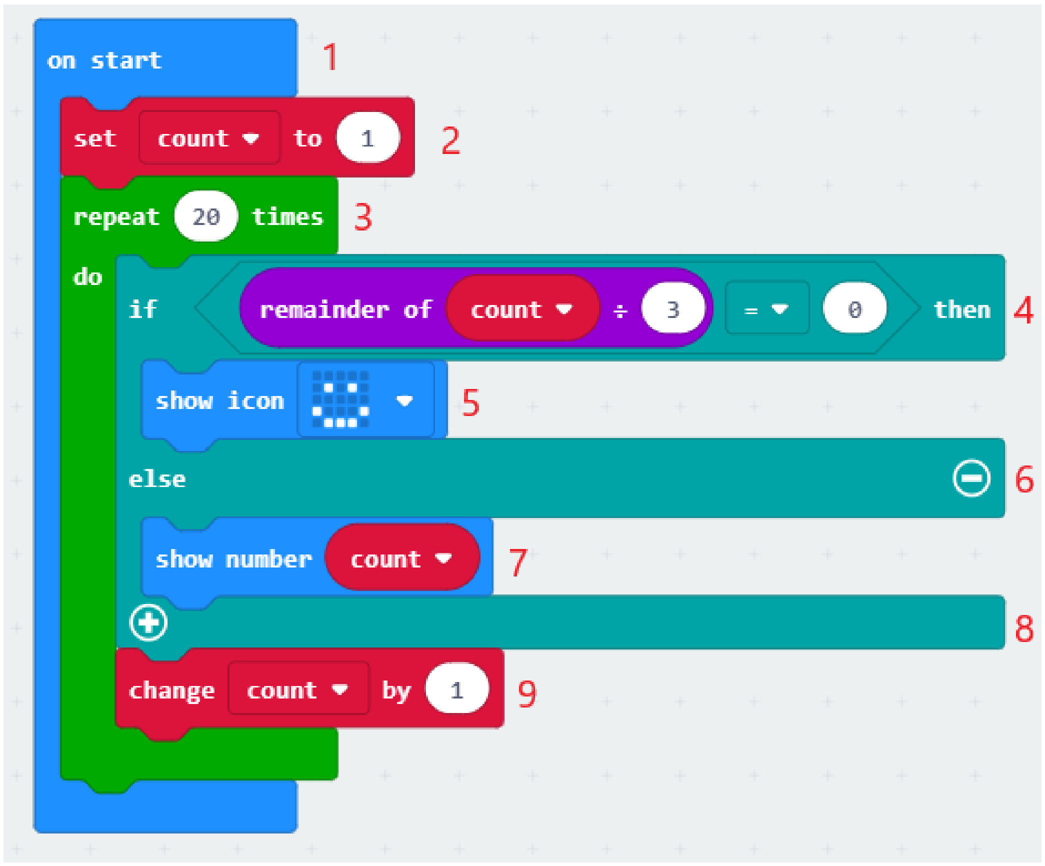
For the learning objectives in CT and coding, Grace put her focus on conditionals in her hook activity. She mentioned that she let students explore with the RPS activity on the previous day, and she realized that a lot of students did not understand it, so she decided to review the concept. As shown in Figure 5 before, Grace incorporated physical actions such as standing up and clapping. She asked individual students questions like "Why didn't you stand up?" and "Why did you clap?" in the whole class discussion. She explained the if-else if-else structure using an analogy: "You're going to either one of the categories. Think of it as filtering." The use of if-else-if-else construct in the RPS program is demonstrated below in Figure 7.



```
on shake
    set hand to pick random 1 to 3
    if hand = 1 then
        show icon
    else if hand = 2 then
        show icon
    else
        show icon
```

Figure 7. **If-else-if-else Construct in the RPS Program**

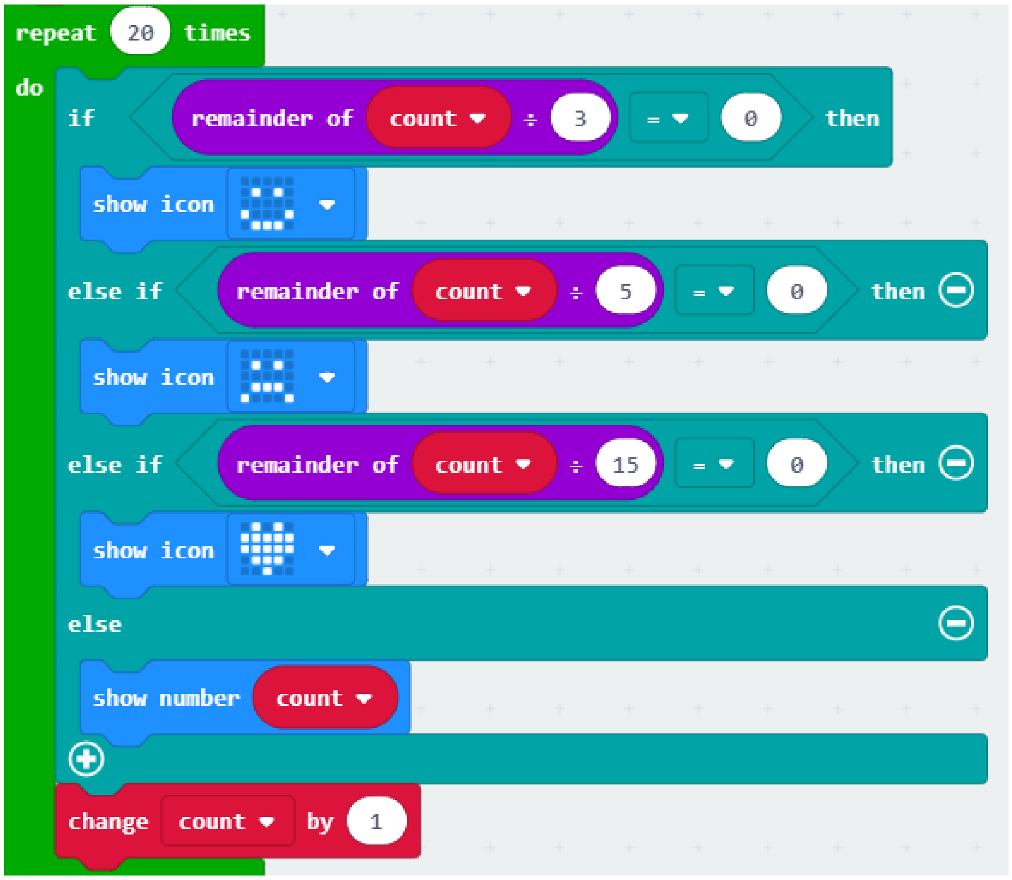
Below is a partial implementation of the Fizz-Buzz program. The variable *count* keeps track the number from 1 to 20. The program displays a smiley face if *count* is a multiple of 3; otherwise, it displays the number.



```
on start                                                  1
    set count ▾ to 1                                      2
    repeat 20 times                                       3
    do
        if remainder of count ▾ ÷ 3 = ▾ 0 then            4
            show icon ▾                                   5
        else ⊖                                            6
            show number count ▾                           7
        ⊕                                                 8
        change count ▾ by 1                               9
```

Figure 8. Sample Code for Partial Implementation of Fizz-Buzz

In addition to conditionals, the activity also required students to understand sequences, loops, data and operators. As shown in Figure 8, there is a sequence of steps to create a property functioning program – creating a variable for the number (line 2), checking if the number is a multiple of three (line 4), executing the code twenty times (line 3), incrementing the number (line 9), etc. The order of the sequence is important. I noticed few students making mistakes with this concept. For example, incrementing the number outside the loop would result in the check being executed 20 times when the number remained to be 1. Another mistake I noticed in the implementation of Fizz-Buzz was with the number 15, which is both a multiple of 3 and a multiple of 5. A few student used an incorrect sequence, as demonstrated in Figure 9 below, and then realized they needed to check for 15 first; otherwise the program would execute the first if block (because 15 is a multiple of 3). The immediate feedback gave students an opportunity to think about the logical error they made.



```
repeat 20 times
do
  if remainder of count ÷ 3 = 0 then
    show icon
  else if remainder of count ÷ 5 = 0 then
    show icon
  else if remainder of count ÷ 15 = 0 then
    show icon
  else
    show number count
  change count by 1
```

Figure 9. **Student Mistake in Fizz-Buzz Implementation**

Grace found loops to be a relatively straightforward concept for students. She went over the repeat block and the while block with students the previous class, and she felt it was not difficult for students to understand the concept. I noticed that some students did not use loops in the program; Grace pointed out that some students did not have a good grasp of how to use a variable to store the number:

> They are not really understanding how variables are being used. Not a lot [of students], but I saw a few students to just manually have the specific numbers print whereas I want the number to be a variable. So few were not understanding how to utilize that.

Students also used both mathematical operations and logical operations. The block "if remainder of count ÷ 3 = 0" involves both a mathematical operation (division) and a logical operation (comparison), but the block, with plain English, made usage of these operations intuitive for students.

Time Economy

In terms of time spent for planning, Grace worked with two colleagues at the same school on a professional development day and created the PowerPoint slides for the three-lesson unit. For this lesson, she spent about an hour the day before to add extra slides for the activity. As for time spent in class, Grace did not feel that there was ineffective use of time due to the use of technology.

Overall Experience

During the interview, Grace rated her lesson 3 to 3.5 on a scale of 5. Her main challenge was not having enough for students to work on. She thought about giving students an opportunity to create their own programs but there was not enough time for students to start on it. Given this was her second time doing the unit, she felt that she better understood where to spend more time on and mistakes students might make.

Grace also mentioned that she would like to try something new next year:

> I think I want to change it up because I get a little bored with the same activity. So maybe I'll look for something that's similar difficulty level but that's a little bit different… It would be nice to have those alligator clips, or lightbulbs, something I add to the micro:bits, so it's more interesting. Cause otherwise I feel like the simulator does exactly the same thing. You need to have something a bit more physical.

She felt that having students playing with the physical micro:bits was more engaging, but using the buttons is the same in the simulator. What made the experience different and more interesting was using the accelerometer in the RPS program on the previous day. Students had fun shaking the actual micro:bit device to play rock-paper-scissors. Overall, Grace's experience was relatively positive, and she is willing to challenge herself and try new activities with micro:bits next year.

5.4. Julia's Lessons

Julia is an experienced teacher who feels comfortable with CT and coding but has not done any technology supported CT activities with her classes for a long time. She is looking for ways to incorporate coding in her classes. In Julia's survey response, she emphasized on how immediate feedback provided by technology can help students develop into better problem solvers. After discussion, we co-planned the coding lessons for her Mathematics 8 course, utilizing the micro:bit as a tool to support the learning of probability.

The unit consists of four lessons. With permission from Grace, Julia used her slides and modified them for her probability unit. On the first day, students were introduced to the micro:bit tool, doing the same Beating Heart and counter tasks to get familiar with the micro:bit device. In the second lesson, students made the RPS program following the tutorial. The next day students created a coin-toss program, and they used both the RPS and the coin-toss programs to conduct a probability experiment. For the last lesson, Julia discussed the experiment with the students. I observed both the second and the third lessons of the unit.

Working Environment

Julia made different arrangements for the two lessons. For the first lesson I observed, students worked in their regular classroom with laptops borrowed from the library. This was due to a mix up with another teacher when booking the computer lab, but this turned out to work very well for Julia:

> The lab I initially booked was mis-booked, miscommunications between teachers, so I ended up… Our tech ended up putting us in our classroom with laptops, which actually turns out to be a gift, because the students were comfortable in the classroom, well-behaved and focused.

For the second lesson, the class started in the regular classroom. Students spent about twenty minutes coding at the library computer stations and then returned to their classroom for the probability experiment.

Resource System

In terms of resources, Julia used PowerPoint slides from Grace and familiarized herself with micro:bits going over tutorials from the micro:bit web site. She also made use of online videos. The video she showed on the day I observed was a video titled “Why is Programming Important?” from Code.org. She used the video as a hook to initiate a discussion with students why they might want to learn programming.

Julia borrowed a class set of micro:bits from Grace’s school. She commented on the value of having the physical micro:bits:

> One of the things about the microbit which I think is very important, and I did not understand that until this lesson, was how important it is to have the tangible piece. If I evaluated this independently of this lesson, I would have said why would you bother with the microbit. You got the simulator and you're coding and you're making them do something. And the microbit, the physical object, increased engagement. So it really helped kids see that code goes inside things, which I know, but I can't assume they know. And they actually experienced it. And the fact that they used it to do a couple different things is neat.

Knowing that micro:bit is compatible with Scratch, Julia also contemplated if she would use Scratch instead. She concluded that she would still use the micro:bit interface, as it helps students attend to the functionalities more:

> Some of the kids would be exposed to Scratch, so being Scratch compatible, might mean they would jump into it much faster. But on the other hand, it was not a set of difficult programing tasks, and I think for students to experience a different interface, is to also develop the thinking piece… I think, if I was to run this again, I would still use the micro:bit environment.

Julia also pointed out how features of MakeCode help her guide students navigate the interface:

> When I was working with kids there was that diamond, like all of the pieces are shaded and color coded. That didn't occur to me until I was working with the students. Once I had that information in my head, it made it a lot easier, you know, so I could adjust my questioning. Did you notice that all the logic control… is pink? Or whatever the color was. Did you see anything in the menu? So that allowed me to guide them a bit better?

Julia believed that the technology supported her in teaching concepts in the probability unit. The micro:bit tool itself did not teach the concept directly; Julia still had to do the mathematics instruction. However, programming the micro:bit provided a different context for the probability experiments she did with students in the past in two ways: 1) The "random" coding block reinforced the concept of randomness and 2) there is the flexibility to program the tool with different probability assignments and explored the idea of unfairness.

At the beginning of the first lesson I observed, there were some technical issues with student laptops. Julia had to get the school's tech liaison teacher to help. She commented on how the school has a fairly poorly maintained laptop cart, and laptops were harder to control. Nevertheless, she would still prefer to have students using laptops in her regular classroom for the familiar setting.

When being asked about resources, Julia also commented that she noticed the class went much better when there were two adults in the room:

> I think a resource [teacher] in the classroom to help would be huge. And I'm pretty comfortable with a) classroom management and b) the technology piece… but I found it very useful to have someone in the classroom to help… I'm not sure if that's a possibility but that might speak to team teaching with other teachers.

Activity Format

The first of Julia's lessons that I observed was similar in terms of structure comparing to the other two teachers' lessons. The class started with a video on why programming is important from code.org as a hook. Students then worked on the RPS program following the micro:bit tutorial. However, explicit mathematics instructions were interweaved throughout the coding unit. When most students were done creating their RPS programs, Julia gave students the next task and wrote on the board: "Make an unfair game where all three events are not equally likely. You have a fair game. Now make it an unfair game." Towards the end of the class, students wrote their reflection about what it means for a game to be unfair, and how their modification met their definition. The breakdown in terms of time was as follows: the introduction took about 10 minutes, the main RPS task was about 35 minutes, the unfair game task took about 20 minutes and the student reflection was about five minutes.

The second day happened to be a shorter day, and the class period was only 55 minutes. Julia started the lesson with direct instructions and discussions with students on theoretical and experimental probability. She then gave out instructions for the task, which was for students to create both the RPS and coin toss programs on micro:bits. Students worked in pairs, and one student would program a fair RPS game while the other create the fair coin toss program. Julia demonstrated doing experiment with two micro:bits simultaneously, and students were clear about what they were tasked to do. Students headed to the library to work at the computer stations. The library was close to Julia's classroom, and it only took about 10 minutes from the beginning of the class to having all students settled down in the library.

It only took about 20 minutes before the entire class came back to the classroom, and Julia was pleasantly surprised by how quickly students were able to complete the task and how much confidence they showed. After that, students started their experiment by doing RPS and coin-tossing with micro:bits 20 times. Julia put a table listing all possible outcomes on the whiteboard for each pair to report their data. Julia then led a whole class discussion on the combined data.

When reflecting on the main RPS activity in the first lesson, Julia felt she would also get students to collect data of the unfair case:

> If I could do it again, when they did the RPS, I asked them about whether the game was fair and make it unfair. I think I actually would have asked them to collect some probability data. Rather than having them relying on their intuitive sense.

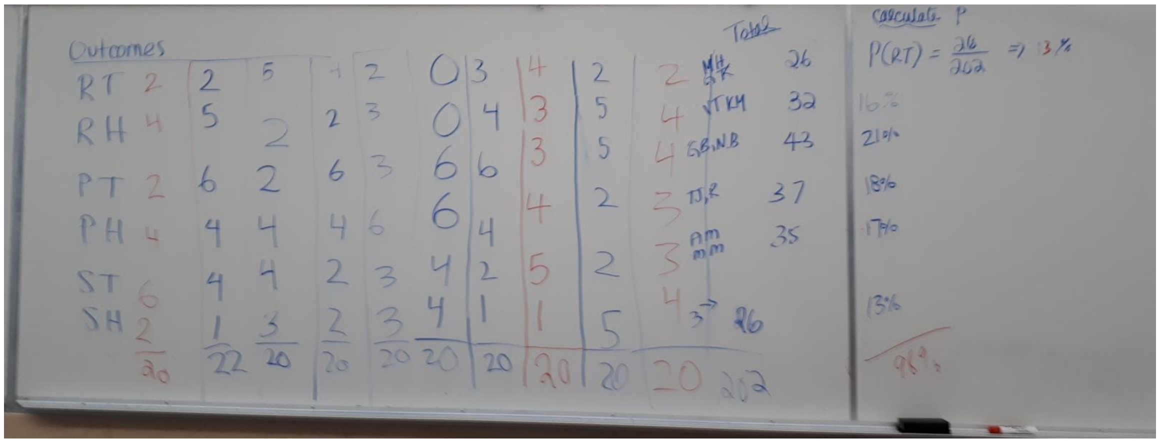
Curriculum Script

Julia and I discussed the learning outcomes prior to the lessons. The lessons focused on the curricular competency of reasoning and analyzing. Students used technology to explore mathematical ideas and test conjecture. In terms of curricular content, the activity supported the learning of theoretical and experimental probability with two independent events.

Julia talked about how the activity gave her context to discuss the idea of randomness:

> What I saw when they started to do the analysis, was that the idea of random wasn't really well set. So they're saying, it's not doing it right. You see I'm not getting any rocks, or I'm not getting any papers, or I'm not getting any scissors. The take was, well it's random dear, you're not gonna get rock, paper, scissors, rock, paper, scissors, just because your program is set up that way. And that conversation I had more than once. So I think it really did help… provided a different window on the concept of randomness. I think it's good foundation for our work with probability.

On the second day, Julia started the class by formally introducing the concept of probability, talking about key terms such as sample space and favorable outcomes, showing the notation and provided a few simple examples such as flipping a coin and rolling a die. After students completed the experiment, the class engaged in a discussion about what they notice about the combined data.



Outcomes												Total	Calculate P
RT	2	2	5	4	2	0	3	4	2	2	MH AK	26	$P(RT) = \frac{26}{202} \Rightarrow 13\%$
RH	4	5	2	2	3	0	4	3	5	4	VTKM	32	16%
PT	2	6	2	6	3	6	6	3	5	4	GB,NB	43	21%
PH	4	4	4	4	6	6	4	4	2	3	TJ,R	37	18%
ST	6	4	4	2	3	4	2	5	2	3	Am mm	35	17%
SH	2	1	3	2	3	4	1	1	5	4	3→	26	13%
	20	22	20	20	20	20	20	20	20	20	202		98%

Figure 10. **Experiment Data from Julia's Class**

Students made some interesting observations, such as:

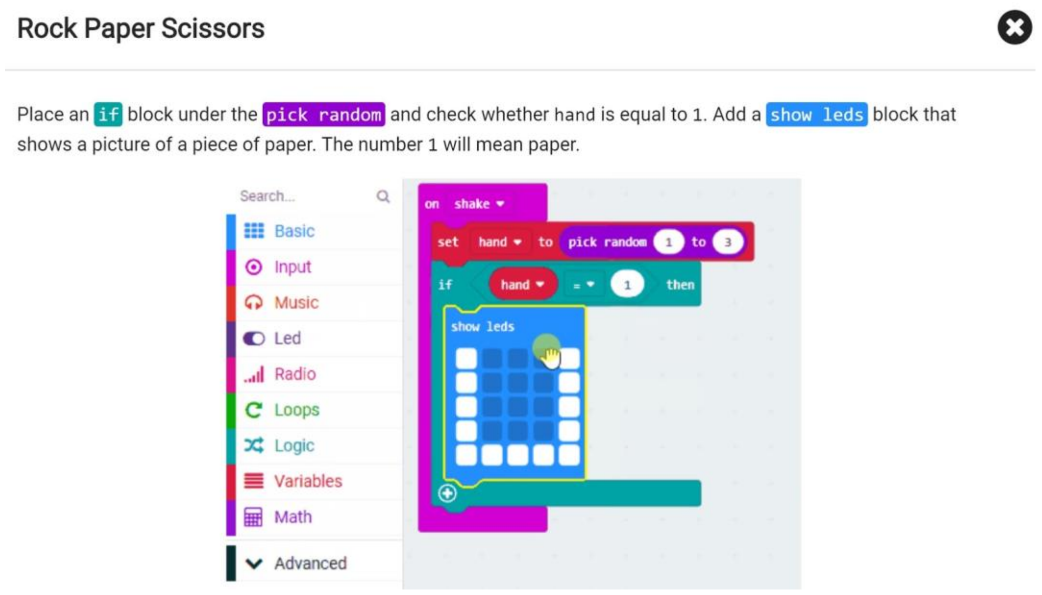
- The numbers (totals for the favorable outcomes) are all double digits
- None of the individual outcomes from the experiment of 20 trials are over 6
- The mean of the totals is 33; all the totals are close to this number.

These provided some interesting discussion points. Also, one group accidentally did 22 trials instead of 20 trials, and Julia took the opportunity to discuss with students why that is okay when calculating probability. Moreover, when students calculated probability, some asked about how to convert 26/202 into a percentage. It gave students an opportunity to revisit a concept they learned previously.

The same experiment can be done with students playing RPS and flipping coins. However, Julia felt that with micro:bits, students had to think about the idea of a random event, and they have the opportunity to explore fairness through the assignment of probabilities to different events in the unfair case of RPS. It gave her some powerful tools to engage in mathematical conversations with the students.

As for learning outcome for coding, the key concepts were sequences, events, data, conditionals and operators. Sequences, events and operators were straight forward concepts. With micro:bit's accelerometer, students experienced the shake event, which was fun for them.

Conditionals was a challenging concept for some students. Students had trouble with the following steps in the tutorial shown in Figure 11, which combine several instructions and usage of blocks from different categories:


Rock Paper Scissors

Place an if block under the pick random and check whether hand is equal to 1. Add a show leds block that shows a picture of a piece of paper. The number 1 will mean paper.

Figure 11. micro:bit RPS tutorial - if block

When students press the + sign at the bottom of the if block, the else block appears:

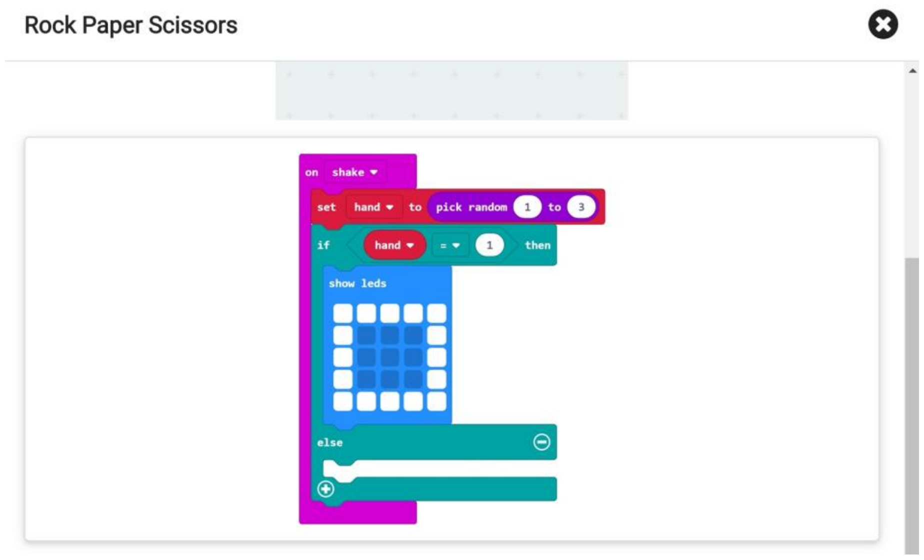

Rock Paper Scissors

Figure 12. micro:bit RPS tutorial - else block

A few students were confused about "else" and did not connect it to the if statement. I also noticed that one student, not following the tutorial, created the program using three if blocks. Students were not explicitly taught the “if-else if-else” construct, but upon reflection, Julia still liked having students learn this through exploration, as it is a concept best learned in context.

As for the use of variable, Julia mentioned that she talked about this with the students in a previous activity, which was to program buttons A and B on micro:bit to make a counter that displays numbers:

> At the end of the first day there was a counter I did. On A they increment the counter, and on B they decrease the counter, and on AB they display it. I pulled the class together at the end and did a debrief and introduced the term initialize. Ask them why that’s part of the code… So what happens instead if I set the counter to 0, I set counter to 15 what would happen? So we did that debrief in the end after they played with it. And you can see lightbulbs going on.

Students then in the RPS program, applied the idea with a variable *hand*, which stores a random number from 1 to 3 and determines whether rock, paper or scissors is displayed.

For operators, only the logical operator for comparison was used, which was intuitive for students. As previously discussed, the interface helped Julia guide students better to find the block. She was able to show students that the block was under logic because of the diamond shape and teal color, not relying on her prior knowledge that the comparison block would be under logic.

Time Economy

In terms of planning time, Julia spent about an afternoon and an evening to prepare for the lessons. A little bit of the time was spent on creating the handouts, but mostly she spent time to walk through the slides from Grace, screen the videos, and coded the programs in micro:bits to make sure she’s comfortable.

As for time during class, Julia did not find inefficient use of class time. Despite having technical difficulties, she felt it is the cost of doing something new, and not beyond what she expects.

Overall Experience

Overall, Julia had a positive experience teaching the coding unit. Student engagement was high, and students were swift in learning the set of basic coding skills to complete their tasks.

> I would say they were very proficient very quickly. That was very impressive. Makes me sad I don't see any other further places to do more of this with them, but I think what it will do is change my… shift my vision a bit so I'll look for more opportunities for coding. And to even have the coding inform the math.

Julia noticed that a student who typically does not enjoy math class showed enthusiasm towards the activity. She also pointed out that coding provided enrichment for high-achieving students:

> My strongest students who I sometimes think get fed left a little bit behind seems like fish with water with the coding, which really isn't a surprise. But it was nice to have something for them. And just to see the kids, like quite enjoying themselves.

Julia mentioned that she wanted to be able to give students that needed a challenge in Python:

> I don't think I would have a hard time coding in Python, but I haven't used Python. So that would be more of a stretch for me.

Given more time, she would also want to try something more complex with micro:bit.

The three teachers Ada, Grace and Julia had positive experiences with their lessons and showed interest in implementing the lessons again. In the next chapter, the three cases are cross-examined based on the five key features from the adapted SFCPF and teachers' overall experience.

Chapter 6. Cross-Case Analysis

In this chapter, I compare three participating teachers’ lessons and cross-examine the data according to five key features of the adapted SFCPF: working environment, resource system, activity format, curriculum script and time economy. I also find common themes that emerged from teachers’ overall experience of incorporating CT and coding in mathematics classrooms. Moreover, I reflect on own teaching practice, comparing with participating teachers’ experience.

6.1. Working Environment

The working environment can play an important role in teachers’ experiences incorporating CT and coding activities in their mathematics classrooms. In the three cases, there were a variety of working environments. Ada chose to book the computer lab, Grace used laptops in the classroom, and Julia used a mix of laptops in the classroom and computer stations at the library.

Grace highlighted the issue of space in her assigned classroom. I noticed during the observations that both Ada’s lesson in the computer lab and Julia’s lesson at the library allowed more space for teachers to walk around and help students. Comparing to a typical mathematics lesson, teachers spent more time circulating around in the room, and space, at least in Grace’s case, made a difference in her experience teaching the lesson. Space is a factor teachers can consider for a lesson involving coding if options are available.

Ada’s case demonstrated another factor from the working environment that could have an impact, which is equipment. Ada taught in a setting that was different from her normal classroom; the lab was not equipped with a projector. She adapted to the situation and adjusted the activity format to deliver the lesson. In addition to working in an unfamiliar environment, not having a projector increased the complexity of Ada’s preparation work.

Both Grace and Julia pointed out how using laptops in their classrooms provided a more familiar environment for the students, despite potentially having more technical issues with laptops. The comment was intriguing to me, as I preferred teaching in the computer lab more than using the laptop cart. As both teachers mentioned, laptop carts tend to be not well-maintained, and I had encountered the issue of batteries, time taken to administer and collect laptops, etc. I wonder if my experience both as a CS teacher that regularly teach in the computer lab and a mathematics teacher that tend to do more mathematics projects involving technology in the computer lab contribute to my preference of booking a computer lab.

It was evident from teachers' responses that they were thoughtful when choosing the working environment, weighing the advantages and disadvantages. The teachers were aware of how the decision could potentially have an impact on their lessons.

6.2. Resource System

All the teachers depended heavily on online resources to develop their lessons. Both Snap! and micro:bit have an abundance of resources on their official web sites. The increase in accessibility to more user-friendly, visual programming tools seemed to enable the teachers, in spite of having little background knowledge, to try incorporating coding in their mathematics classrooms. It is important that teachers are made aware of the wealth of online resources to help them learn CT and coding as well as design lessons.

As mentioned in 2.4, Benton et al. (2016) stated how educators found LOGO programming too difficult. The large number of easily accessible resources to support teachers were not around with LOGO. Block-based programming eliminated some challenge in learning programming syntax for both the teachers and students. However, the teachers were aware that text-based programming can provide more challenges to the students. Grace introduced micro:bit's functionality to switch to JavaScript and gave her Mathematics 8 Honours students the option to explore online text-based programming tutorials. Julia also showed a willingness to learn Python.

It is interesting that both Ada and Julia, when asked about resources, commented on the block-based programming interface. Julia gave a good example of how colour-coding of the blocks helped her guide students better. Both teachers, upon reflection, thought they would continue to use the same tools instead of Scratch. As I observed in the classrooms, there were students bringing up Scratch projects after they completed assigned tasks; Scratch seemed to be a much more familiar tool for the students. This showed that teachers deliberately made choices of the tools and considered factors such as its appearance and functionalities.

Both Ada and Julia commented on how having the handout and PowerPoint slides prepared for them helped them tremendously. For Ada who had the idea but may not be confident enough to try coding before, being able to take an activity that another teacher developed and add to it made implementing the lesson much easier. Grace's experience of co-designing the activities with colleagues also showed the importance of collaboration. This was evident from the teacher survey as well, as seen in section 4.1, where a teacher with no background in coding learned it from colleagues that had experience in programming.

In terms of how these tools supported the curricular goals, Ada stated that with coding in Snap!, she had a different way of checking student understanding. As quoted in section 5.2, Ada saw students demonstrated if they understood similarity and angles in regular polygons "because they are making it happen visually." She did not need to give a quiz that asks students if they know ratios of lengths for corresponding sides of similar triangles are equal, or if they can find the interior angle of a regular pentagon. Julia commented that having students program the micro:bit provided her a better context to discuss randomness. As discussed in 5.4, Julia found that she had several similar conversations regarding randomness, when the RPS program did not work as students anticipated. The technology also allowed students to explore the idea of fairness. If a regular coin were used or an RPS game were played, the weightings of the outcomes could not have been modified.

In addition, another important resource is people that can support the class during the lesson. All three teachers mentioned the collaborative nature of the coding lessons; students who finished early were encouraged to help their peers. Moreover, having other adults in the room can be beneficial. Grace gave the example of how having a volunteer in her classroom for a different block could help with differentiation. Julia also stated how much smoothly the lessons went when an additional adult was in the room. I found this especially true with these two teachers using micro:bits and laptops. There were a lot more responsibilities on the teachers – to sign out and collect laptops and micro:bits, handle dead batteries, fix technical issues, etc. while answering student questions. When I observed in the classroom, I played the observer-participant role because having an extra person can make the experience better for both the teacher and students.

6.3. Activity Format

A lesson involving coding is quite different from a typical mathematics lesson, in which content delivery, in most cases, is through teacher-guided instructions. Each of the lesson observed followed a similar format. They started with an introduction followed by students working on coding tasks individually or in pairs. Grace and Julia added videos as hooks for their lessons. For all of the teachers, the coding tasks were delivered either through the pre-developed handout or online tutorials. They were designed to be self-paced and pre-determined with some flexibility to make slight modifications to fit teachers' needs. Thus, the teachers were in a supporting role while the students were coding, and the activity format naturally involved a large period of time for student exploration.

Ada and Julia both gave minimal instructions on the coding part, whereas Grace spent time to go over computing concepts. She made the choice of explicitly teaching conditionals and loops, which was something she did differently comparing to when she last implemented the same lesson. The time spent, which was only about five minutes, was very short compared to usual mathematics instructions. Ada also commented that next time she would highlight the concept of variable prior to students doing the tasks. Julia, on the other hand, recognized that some students were unclear about the concept of the conditionals, but still liked having students learn this through exploration. None of the teachers, upon reflection, would change the activity format much and only make small adjustments.

Although there were similarities to the three teachers' activity structures, the level of integration of mathematics was different. For Ada and Grace, the focus was more on coding. Both teachers used tasks that involved mathematics concepts students previously learned. Grace's lesson was least connected to mathematics learning outcomes; coding was done as a separate unit. Ada's lesson was tied to a unit in Mathematics 9 and there were more conversations about the mathematics involved, but the lesson objective was more about exposure to coding. This was evident from Ada's remark that she did not want to take students away from coding and push them beyond the prescribed learning outcomes in her mathematics unit.

In contrast, Julia's lesson was most integrated in a unit in Mathematics 8, and this made the activity structure design distinct from the other two teachers' lessons. The first lesson observed was still similar to the other two teachers' lessons, as it was an introduction to the RPS program. The second lesson, however, started with mathematics instruction on probability, followed by a short work period for coding, and ended with student experiment and discussion. If students were using actual coins and playing RPS for the experiment instead of creating programs, the lesson structure would not change much, but as Julia pointed out, coding added value to her lesson by providing the opportunity for students to think about the idea of a random event and explore the concept of fairness.

None of the teachers formally assessed the students with these activities. Both Ada and Grace asked students to show them the programs they had made during the lesson. Ada commented that coding was new, and she did not want assessment to be a source of stress. Julia collected the student reflections, but it was more to inform her of students' exploration of ideas about probability. The teachers did not tie grades to the coding activities. Coding seemed to be seen as an addition or enrichment activity, instead of being part of the curriculum.

6.4. Curriculum Script

In terms of mathematical content of the lessons, both Ada and Julia commented on their conversations with students. Ada highlighted how she had a different style of conversations with students. She emphasized proper mathematical terminology in the handout she gave students, but the technology offered a visual way for her to confirm students' mathematical understanding so that it was not necessary to use the same vocabulary in their conversations. Julia pointed out how having students create the coin and RPS programs on micro:bits sparked different conversations about randomness and unfairness. If real coins and RPS were used for the probability experiment, students would have no opportunity to change the probability weighting of the possible outcomes. With the incorporation of coding, the teachers' curriculum scripts changed comparing to their normal mathematics lessons, and the technology stimulated good discussions on the same mathematics topic.

As for key coding concepts, the following table shows a comparison of the three teachers’ activities, based on the categorization by Brennan and Resnick (2012):

Table 7. Key Coding Concepts Used in Participating Teachers' Activities

	sequences	loops	parallelism	events	conditionals	operators	data
Ada	✓	✓	✓	✓			✓
Grace	✓	✓			✓	✓	✓
Julia	✓			✓	✓	✓	✓

Two concepts that students mainly had trouble with were conditionals and data. Grace thus designed a short activity to review the “if-else if-else” construct, and Julia pointed out that “else” was unfamiliar to students. In terms of data, as all three teachers’ lessons were introductory to coding, variable was the only type of data used. Ada highlighted how the different naming convention of a variable may be confusing to students, and Grace pointed out that some students did not know how to utilize the variable to store data and manually print out the values they needed.

Although the teachers described CT as related to problem-solving, they focused on coding concepts when they thought about CT learning outcomes, and I did not observe them explicitly teach computing practices. As shown in the teacher survey, most teachers talked about problem-solving and characteristics such as breaking down problems (decomposition), pattern recognitions (generalization), etc. Both Grace and Julia, like many of the teachers surveyed, stated that helping students develop problem-solving skills as a motivation to teach CT and coding. However, CT strategies were not evident in teachers’ curriculum script. During the sessions, I also did not observe explicit teaching of computing practices such as testing and debugging, which were mentioned in teacher surveys. This is not surprising. The BC mathematics curriculum documents merely add “include coding” without elaboration, and the term CT only exists in CS 11 and 12 courses. Also, the teachers seemed to assume that students learn CT practices through attending their work on coding tasks, although many research studies have shown transferring of problem-solving skills does not happen automatically, as discussed in Chapter 2. Focusing on programming and not explicitly teaching CT practices is one of the reasons Armoni (2016) critiqued the CT movement.

6.5. Time Economy

In Ruthven's analysis (2009), time economy was more related to how much more time it took for the technology to be incorporated into the mathematics classrooms. This was not as relevant in Ada's and Grace's cases, as the teachers had set aside time to do the coding activities. From Grace's response that she wished to have laptops available in the classroom throughout the year so that she could do more coding, it was implied that she found coding to be a valuable use of her class time. In Julia's situation, as she integrated coding into her probability unit, she likely spent a few additional days than she normally would. Nevertheless, she also expressed interest in doing more coding. Julia wished she could find more places in the course to do more; this comment may imply that incorporating CT and coding requires extra planning, or that perhaps it requires re-thinking of the curricular concepts to make them more amenable to coding activities. None of the three participating teachers were concerned about taking time away from covering mathematics concepts. This contrasts with one of the teachers, quoted in section 4.5, had not tried coding on the grounds that there was not enough time to incorporate it in his or her classroom.

Despite having to spend time on technical issues, the teachers seemed to feel that the overhead was the cost of incorporating technology and did not find it inefficient use of class time.

6.6. Overall Experience

Upon reflection, all three teachers felt positively about coding and expressed interest in doing the lessons again. A few common themes emerged from the teachers' overall experience implementing coding lessons in their mathematics classrooms, which were high student engagement, accessibility to a wide range of students and lack of extensions.

In all three of the classrooms I observed, students were enthusiastic and highly engaged in the activities. The tangibility of micro:bits especially seemed to make the experience exciting for students. The first thing Grace said in the survey about her motivation to include coding was because it is fun for students. Julia also commented that "anytime you get engagement there's more learning" during the interview. One can argue that enjoyment does not necessarily imply learning. Nevertheless, in all three cases, the teachers were thoughtful about the potential affordances coding had on supporting learning. The student experience would be different if the teachers just let students go through online coding tutorials, which could also be engaging, but lack the mathematics connections teachers could help students make.

The coding lessons were able to reach a wide range of students. I noticed that students were able to get started immediately in all three classes. Ada commented on how the lesson was accessible to all her students. Everyone was able to enter the activity very easily, but the activity still provided a challenge. The stronger students, in Julia's words, felt like "fish in water" with coding. However, there were a few students who normally did not engage in mathematics classes that also enjoyed the activities. Julia noticed that her students were proficient very quickly and showed a lot of confidence. This was also something I observed in my own classes with coding. As Grover and Pea (2013) claimed, many of the programming environments for learning were developed with the "low floor, high ceiling" principle, and lessons observed in the three classrooms were all self-paced and designed for students with no background in coding. Thus, the lessons were able to reach students at different levels.

While coding provided enrichment for high-achieving students, all three teachers highlighted the lack of extensions as a major challenge when they incorporate coding. In Ada's remark, she felt some of the students were "expert coders". Grace also felt that the pace was good for the average students, but she did not have a lot for students who finished early to work on. Julia also wished she was able to provide extension in Python, despite not knowing the programming language. All three teachers actually planned for extensions in their lessons. For Ada, there were additional Snap! tasks she added to the handout. Grace and Julia both encouraged students that finished to try other micro:bit tutorials. However, the teachers seemed to feel that these extensions were not enough. Ada stated that she did not have enough programming knowledge to guide the students and provide more challenges. Julia, who is more experienced and confident in her ability to learn a new programming language, still found learning enough Python programming to support the advanced students too overwhelming a task. Grace's example of using the CodeHS Python tutorial for her Honours class the following day showed that teachers may not have enough background knowledge or time to devise advanced extensions and had to rely on something that was already developed.

To summarize, there were some similarities in differences in the teachers' experiences and implementations of CT and coding lessons. After analyzing the data, in the next chapter, I address the research questions from section 2.5 and conclude the study.

Chapter 7. Conclusions

In this chapter, I address the research questions outlined in section 2.5. Then, I discuss the limitations and implications of the study. Lastly, I reflect on what I learned from the research both as a teacher and as a researcher.

7.1. Answering the Research Questions

The main purpose of this study was to investigate teachers' perspectives and practice of incorporating CT and coding in secondary mathematics classrooms. After analyzing data from the teacher survey, classroom observations and interviews, the research questions can now be answered.

How do teachers understand "computational thinking" and interpret "include coding" in the BC mathematics curriculum document?

Only five out of 19 survey respondents felt comfortable about teaching CT and coding, but most teachers provided a description of their interpretation of CT. The theme of problem-solving emerged in teachers' responses, which demonstrated the influence of Wing's (2006) article. Teachers may not have read Wing's work, but through many professional development workshops and a large number of resources developed since the CT movement started, teachers' views seem to be indirectly impacted by Wing. Several CT practices were mentioned, such as breaking down steps (decomposition), step-by-step thinking (algorithmic thinking) and pattern recognition (generalization). These practices are not specific to CT and are also seen in mathematics classrooms.

Six of the 19 teachers included the word "computer" in their descriptions of CT. The responses that connected usage of computer to CT mainly in two ways: 1) using the computer as a tool for problem solving and 2) learning to use the computer. A teacher specifically stated that teaching CT does not require technology. Though it was not obvious from the teachers' descriptions, few teachers gave examples of unplugged activities when asked to describe CT tasks used in their classrooms.

Although coding is often considered as a vehicle to teach CT, none of the teachers included the term coding or programming in their description. Some teachers commented that CT is breaking down a task so that a computer is able to execute, which could imply coding or programming, but the connection was not explicit. A teacher even considered coding more as a way of communication or expression and not fitting under CT.

According to the survey results, teachers associated CT and coding mostly to the core competencies of creative thinking and critical thinking. For curricular competencies, teachers primarily connected CT and coding to the curricular competencies of reasoning and analyzing, as well as understanding and solving. In terms of curricular topics, all 19 teachers chose “patterns and relations” as most relevant, but the three teachers observed implemented coding activities connected to other topics such as spatial reasoning and probability. A variety of examples that connected to other curricular topics were also provided.

Using block-based programming and online tutorials as resources for coding was popular. While most teachers understood coding as computer programming, some teachers highlighted a different type of coding, such as writing formula in Excel, as an introduction to coding. In my opinion, this different type of coding is also a good way to introduce “coding” in mathematics classes; it certainly fits the curricular competency of using logic and pattern recognition, under which coding is found in the curriculum document. Coding of this type may even have a closer connection to the discipline of mathematics.

From this data, it seems like teachers are beginning to use or at least to think about CT and coding. Most teachers saw the connections between CT and mathematics, and I was a little surprised by how the theme of problem-solving emerged from the responses. As a mathematics teacher, I had anticipated that my colleagues would connect CT more to ideas such as number sense, statistics or data; however, only one teacher mentioned the word “quantities” in the response. This may indicate how the CT movement was successful in broadening the audience to K-12 teacher, and despite not having a consensus on the definition of CT, having this agreed understanding of CT being about problem-solving, provides us with a set of common ideas to work with, and may be helpful in bringing more teachers on board. Teachers may not yet connect the idea of coding to CT, but it is encouraging to see colleagues, who may not have background in CS, being willing to try coding and include it as part of their mathematics classrooms.

What kind of CT and coding activities are developed and used in BC secondary mathematics classrooms?

Teachers provided a wide range of examples of CT and coding activities in mathematics classrooms. Non-coding examples included strategy games, board games, field trips, guest lectures, mathematical investigation activities using DME, an unplugged “human coding” activity, etc. Among these examples, DME is noteworthy as it is already a tool commonly used in mathematics classrooms. As discussed in 4.6, it fosters CT skills not unique from mathematics learning. This raises an interesting question. To enhance the current practice of teaching CT, should teachers focus on explicitly teaching CT strategies when using DME, or should teachers seek to include other tools that are not already part of the mathematics teaching practice?

Coding activities were primarily block-based. Only one out of the 19 teachers used text-based programming in the classroom. Teachers observed demonstrated three interesting coding activities in mathematics classes: Ada’s spiral polygon activity, Grace’s Fizz-Buzz game and Julia’s RPS and coin-flip probability experiment. Other block-based programming examples provided included a number guessing game in Scratch and Spheros robot programming. With a broader definition of coding, some teachers also used writing formula in spreadsheets as a way to introduce coding.

There are many follow-up questions I can ask upon learning about these activities. Once teachers feel comfortable with these activities, would they want to include more? How could we assess students’ learning of CT? Are there CT and coding activities appropriate for higher grades?

What challenges do teachers have and how can they be better supported in incorporating CT and coding?

Teachers faced many challenges with the new initiative of incorporating CT and coding. According to the survey, teachers were mainly concerned about time, resources and training. From the classroom observations, the working environment and resource system were impactful to teachers’ experience. Teachers had to adjust their activity formats and adapted to new curriculum script to fit the nature of a lesson involving technology. Teachers had to deal with the issue of space, lack of proper equipment, technical issues and additional responsibilities that came with the use of technology. Despite the challenges, all three teachers had positive experience and were willing to implement these lessons again.

Time did not seem like a major issue for the teachers that already incorporate CT and coding in classrooms. Teachers understood that there was a cost for using technology. However, with many other topics to teach in the curriculum, teachers also did not go beyond just a few lessons that incorporated CT and coding in a year. Time was a factor that prevented some teachers from trying. It seemed like if teachers believe in the value of incorporating CT and coding and are supported in terms of resources such as pre-made handouts, tools like micro:bits at their schools, colleagues to collaborate with, they are more likely make time for it. At which level to include coding is a factor as well, it is much easier to do so at the grade 8 and 9 level, whereas incorporating coding at the senior levels, where there are more learning outcomes to cover, and more difficult mathematics concepts to connect with, can be challenging. It may be easier to find time and make connections to content in grade 8 and 9 mathematics classes.

Though many teachers agreed that including coding is a positive change, some clarity in the Ministry of Education’s curriculum document on how to connect coding with the curriculum would be helpful. From the survey, while some teachers have found ways to implement CT and coding lessons in the classroom, some teachers still wonder about where to put CT and coding in the curriculum, and what simple resources they can use so they can get started more easily. Professional development is essential to the success of an educational change, and from teacher responses, teachers are not just looking for simple introductory resources for coding, but thoughtful use of CT and coding that can be embedded in mathematics learning and connected to curricular topics.

Time and resources became less concerning when support could be provided to the teachers. Teachers were more willing to try when there were resources provided to them and collaboration with other colleagues. All three teachers in the classroom observations wished for more resources for extensions so that students with experience in programming could still be challenged. Moreover, people can be valuable resources as well. Having an additional supporting teacher in the room to team teach can help facilitate differentiated learning and reduce the burden on the teacher with the technology use in class.

To what degree are CT and coding integrated into BC teachers' mathematics classes?

Based on the survey result, many teachers that currently incorporate CT and coding do it at most a few times in a school year. Ada, Grace and Julia demonstrated three examples of classrooms that incorporated coding. Ada's coding lesson was a stand-alone lesson and extension for a particular unit. Grace's lesson was part of a separate coding unit in the course. Julia's lesson was more blended into her probability unit. All these lessons, as well as many examples provided by other teachers, were additions to their curriculum. And while the lessons that were used came ready-made, all three teachers put thoughts into these lessons, not just sitting students in front of computers with online tutorials and claiming that they have included CT and coding.

However, this also shows that to systematically integrate CT and coding into secondary classrooms similar to projects like ScratchMaths or Bootstrap can be very challenging. ScratchMaths was a two-year program offered at the elementary level. Bootstrap, while implemented with grade 8 and 9 students, was offered as a separate class on top of the regular mathematics class. For this reason, I chose "incorporating" over "integrating" for the title of this research because I felt that "integrating" might imply doing CT (and coding) throughout the course or blending it into our teaching practice. What teachers have done so far appear to be adding CT and coding to courses. Unless executed in special programs, incorporating CT and coding beyond the current level of integration teachers are doing seems difficult with the current school system. When teachers introduced coding in the classroom, it is likely some students' first exposure to coding, so it is hard for teachers to go beyond the basic level. The mathematics curriculum itself can play a role. A mathematics curriculum designed to work well with CT and coding could potentially improve the level of integration. This then raises the question of how much of the curriculum could change to allow for better integration of CT and coding.

7.2. Limitations and Implications

This study is relatively small in scale, with 19 survey respondents and three classroom observations as well as teacher interviews. It is also inevitable that the teachers responding to the survey tend to be ones that have already shown interest in the topic, so their views may not be representative of the teacher population in BC. Perhaps with a larger study, we would be able to see a wider range of teachers' perceptions of CT and coding and activities in mathematics classrooms. Furthermore, all participating teachers work in urban school districts in BC, and a study done in rural areas in the province may lead to different results. In addition, all three lessons observed were in grade 8 or 9 classes and focused on block-based coding. As claimed by a few teachers in the survey, it is more challenging to find appropriate CT and coding tasks for higher grades. It would also be interesting to observe in classes that utilized text-based programming or non-programming CT tasks.

Future research might look at how teachers across BC are implementing CT and coding in secondary mathematics classrooms on a larger scale. It would also be interesting to find out how many mathematics teachers in BC are currently using coding as a tool after the new curriculum was introduced. As there seemed to be no explicit teaching of CT strategies in teachers' implementation of coding lessons in the classes observed, I am also curious how mathematics teachers teach CT in the CS 11 and 12 courses, where CT was part of the curriculum standards. How do they teach CT specifically? How do they assess students' learning of CT? This study highlights the issue that official curriculum documents have added coding without specifying how to include it in the curriculum. In addition, I am also curious if and how traditional teaching resources such as textbooks address CT and coding.

7.3. Final Reflections

As I embarked on the journey of this research, I learned much both as a teacher and as a researcher.

Participating teachers' thoughtful survey responses and interview comments, as well as lessons I observed in the classrooms, made me reflect on my own teaching practice. I learned from my fellow teachers particular teaching practices like classroom management techniques and activity design. I also noticed more specific CT and coding practices such as if students are using trial-and-error instead of mathematics understanding, what lesson ideas have been explored and how to structure a mathematics lesson to include CT and coding. These are all very valuable to me.

Ruthven's SFCPF (2009) helped me realize the variety of factors that affected our different pedagogical decisions when incorporating CT and coding. Something that stood out for me was the wide range of working environments teachers are in. Seeing another colleague teach in a computer lab without a projector was eye-opening, and quite often as teachers we have to be able to adapt quickly to different situations. It was also interesting to see three different approaches in terms of the activity format when including coding as part of a mathematics classroom. In addition, through working with other teachers and presenting professional development workshops, I realized that for my colleagues who may not have background in CS, to incorporate CT and coding in their classrooms is more challenging than I anticipated. Teachers can often get started, but it would be challenging to go beyond the basics and provide extensions.

I have also grown as a researcher. The process of reviewing the constantly growing literature on CT and coding was fruitful and rewarding, though at times it is confusing to find many distinct perspectives. I now consider CT an evolving concept. I have gained a better understanding of the history of CT and coding in education, and I am surprised by the tension between different stakeholders as well as the critiques on CT and coding initiatives. I began this research with great enthusiasm to help other teachers incorporate coding in mathematics classrooms, and this study enabled me to take a step back and use a more critical lens to examine the recent movement of CT and coding in education.

While high student engagement and accessibility to a wide range of students are enough reasons for me to continue incorporating CT and coding in mathematics classes, one thing I would like to do, and want to encourage my colleagues to try, is to use what we learned about CT and teach CT practices more explicitly. This is not something I observed in the study or did in my own classroom before, but I believe that it would be valuable to the students. Decomposition, abstraction, generalization, etc. are not new ideas, but naming these problem-solving strategies helps us become more aware of them when moments of teaching opportunities arise. If I could offer a suggestion to teachers with no experience and want to try CT and coding in classrooms, I would recommend collaborating with other colleagues and starting with ready-made, block-based coding resources with grade 8 and 9 classes. I hope that the three cases in the study provided inspirations and good examples of lesson ideas.

As there has not been known prior studies on CT and coding in BC since the introduction of the new curriculum in 2016, I hope that this study will raise interests among my colleagues and encourage more intentional incorporation of CT and coding in mathematics classrooms. I also hope that the study demonstrated that despite challenges, teachers showed willingness to incorporate CT and coding in classrooms. If CS education researchers and curriculum developers can support teachers by developing teaching resources and offering professional development opportunities, and if school districts and the Ministry of Education can provide more support for better resources and working environments, I think more teachers would bring CT and coding into their mathematics classrooms.

References

Armoni, M. (2016). Computing in schools – Computer science, computational thinking, programming and coding: the anomalies of transitivity in K-12 computer science education. *ACM Inroads, 7*(4), 24–27. https://doi.org/10.1145/3011071

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads, 2*(1), 48–54. https://doi.org/10.1145/1929887.1929905

BC Gov News. (2016, June 10). $6 million to help connect students with coding, new curriculum and computers. Retrieved from https://news.gov.bc.ca/releases/2016PREM0065-000994

Benton, L., Hoyles, C., Noss, R., & Kalas, I. (2016). Building mathematical knowledge with programming: insights from the ScratchMaths project. In Proceedings of *Constructionism 2016*, 25-32.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. Paper presented at the *American Educational Research Association*, British Columbia, Canada.

British Columbia Ministry of Education. (2016). BC's New Curriculum. Retrieved from https://curriculum.gov.bc.ca/curriculum/

Cellan-Jones, R. (2015, March 12). The Micro Bit - can it make us digital? *BBC News*. Retrieved from https://www.bbc.com/news/technology-31859283

Clements, D.H. (1999). The future of educational computing research: The case of computer programming*. Information Technology in Childhood Education Annual*, 1, 147–179.

College Board. (2018). Student score distributions. Retrieved from https://secure-media.collegeboard.org/digitalServices/pdf/research/2018/Student-Score-Distributions-2018.pdf

Computer Science Teachers Association & International Society for Technology in Education. (2011). Computational Thinking: Leadership Toolkit (1st ed.) Retrieved from https://id.iste.org/docs/ct-documents/ct-leadershipt-toolkit.pdf?sfvrsn=4

Denning, P. (2009). The profession of IT Beyond computational thinking. *Communications of the ACM, 52*, 28–30

Education Endowment Foundation. (2018). ScratchMaths: Projects. Retrieved from https://educationendowmentfoundation.org.uk/projects-and-evaluation/projects/scratch-maths/

Gadanidis, G. (2017). Five affordances of computational thinking to support elementary mathematics education. *Journal of Computers in Mathematics and Science Teaching 36(2)*, 143–151.

Google Education. (n.d). Exploring Computational Thinking. Retrieved from https://www.google.com/edu/programs/exploring-computational-thinking/

Government of Canada. (n.d). CanCode. Retrieved from https://www.ic.gc.ca/eic/site/121.nsf/eng/home/

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher, 42*(1), 38–43.

Guzdial, M. (2008). Paving the way for computational thinking. *Communications of the ACM, 51*(8), 25–27.

Guzdial, M., Kay, A., Norris, C. & Soloway, E. (2019). Computational thinking should just be good thinking. *Communications of the ACM, 62*(11), 28-39.

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*, 41, 51–61.

Papert S. (1980). *Mindstorms: Children, computers, and powerful ideas.* New York, NY: Basic Books.

Pea, R. D. & Kurland, D. M. (1983). On the cognitive and educational benefits of teaching children programming: A critical look. - New Ideas in Psychology, 1(3). Elmsford, NY: Pergammon. 137-168

Pei C., Weintrop D. & Wilensky U. (2018). Cultivating computational thinking practices and mathematical habits of mind in Lattice Land, *Mathematical Thinking and Learning*, 20:1, 75-89.

Ruthven, K. (2009). Towards a naturalistic conceptualisation of technology integration in classroom practice: The example of school mathematics. *Education & Didactique,* 3(1), 131-149.

Minister of Innovation Science and Economic Development. (2019). CanCode. Retrieved from https://www.ic.gc.ca/eic/site/121.nsf/eng/home

Selby, C. C., & Woollard, J. (2014). Computational thinking: the developing definition. Presented at the SIGCSE 2014, Atlanta. Retrieved from: http://eprints.soton.ac.uk/356481/

Savard, A. & Freiman, V. (2016). Investigating Complexity to Assess Student Learning from a Robotics-Based Task. Digit. Exp. Math. Educ. 2016, 2, 93–114.

Schanzer, E. Fisler, K., & Krishnamurthi, S. (2018). Assessing Bootstrap:Algebra students on scaffolded and unscaffolded word problems. *Proceedings of the 49th ACM Technical Symposium on Computer Science Education – SIGCSE' 18*, 8-13. https://doi.org/10.1145/3159450.3159498

Schanzer, E., Fisler, K., Krishnamurthi, S., & Felleisen, M. (2015). Transferring skills at solving word problems from computing to algebra through Bootstrap. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education – SIGCSE '15,* 616–621. https://doi.org/10.1145/2676723.2677238

Silcoff, S. (2016, January 17). B.C. to add computer coding to school curriculum. *The Globe and Mail*. Retrieved from https://www.theglobeandmail.com/technology/bc-government-adds-computer-coding-to-school-curriculum/article28234097/

Tedre, M., & Denning, P. (2016). The long quest for computational thinking. *Proceedings of the 16th Koli Calling Conference on Computing Education Research*, November 24-27, 2016, Koli, Finland: pp. 120 – 129.

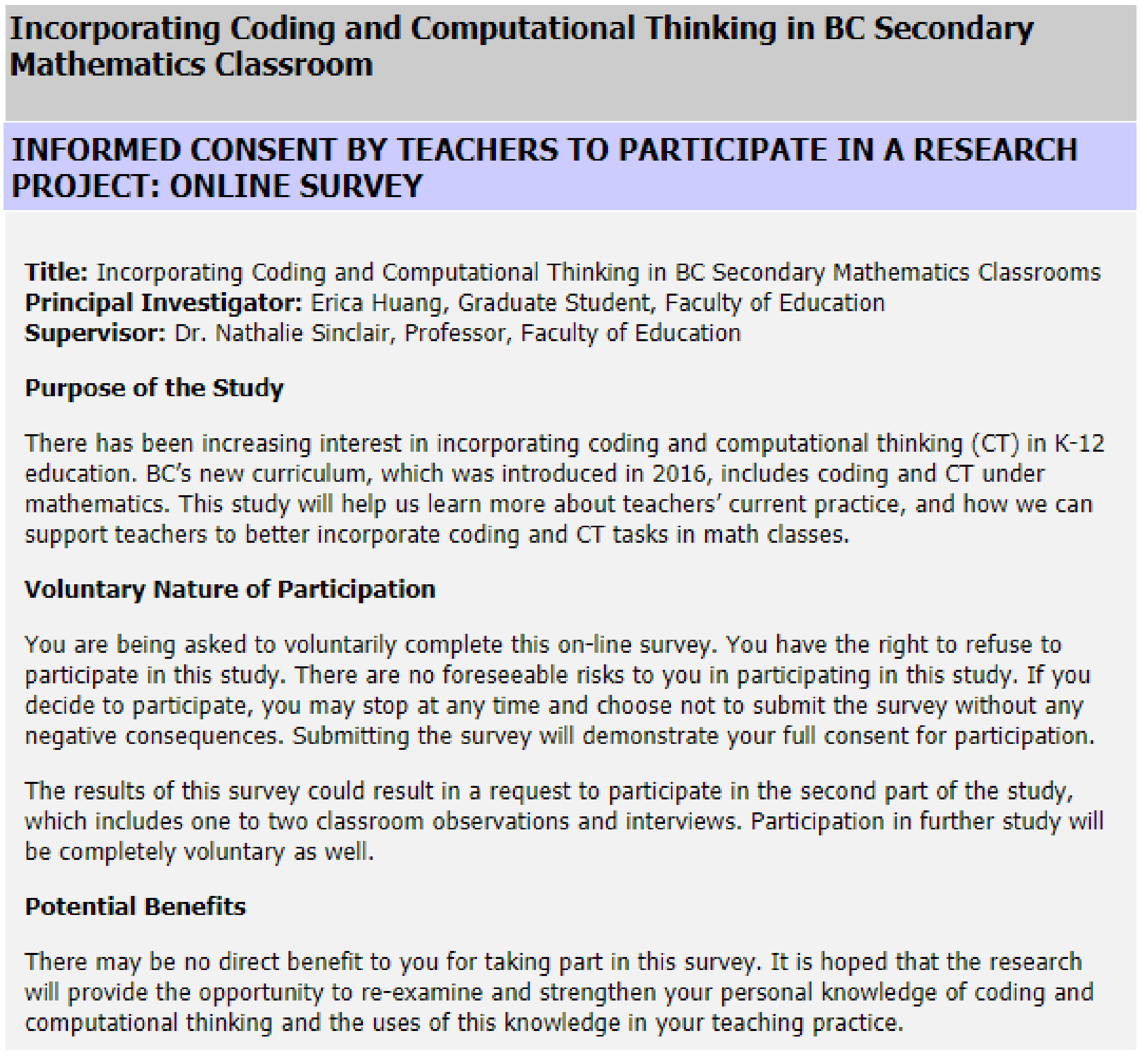
Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., & Trouille, L. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology*, 25(1), 127–147.

Wing, J. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33–36.

Wing, J. (2011). Computational thinking — what and why? The Link Magazine, Spring. Carnegie Mellon University, Pittsburgh. Retrieved from http://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf

Appendix A.

Online Teacher Survey on SFU WebSurvey Interface



Incorporating Coding and Computational Thinking in BC Secondary Mathematics Classroom

INFORMED CONSENT BY TEACHERS TO PARTICIPATE IN A RESEARCH PROJECT: ONLINE SURVEY

Title: Incorporating Coding and Computational Thinking in BC Secondary Mathematics Classrooms
Principal Investigator: Erica Huang, Graduate Student, Faculty of Education
Supervisor: Dr. Nathalie Sinclair, Professor, Faculty of Education

Purpose of the Study

There has been increasing interest in incorporating coding and computational thinking (CT) in K-12 education. BC's new curriculum, which was introduced in 2016, includes coding and CT under mathematics. This study will help us learn more about teachers' current practice, and how we can support teachers to better incorporate coding and CT tasks in math classes.

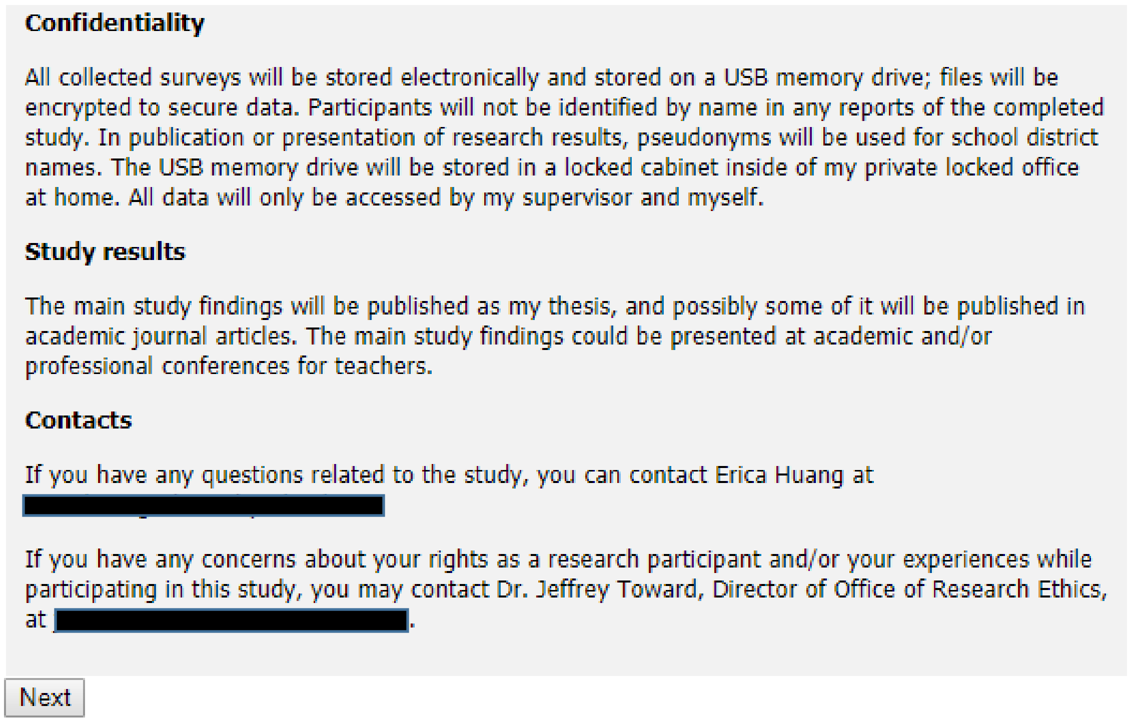
Voluntary Nature of Participation

You are being asked to voluntarily complete this on-line survey. You have the right to refuse to participate in this study. There are no foreseeable risks to you in participating in this study. If you decide to participate, you may stop at any time and choose not to submit the survey without any negative consequences. Submitting the survey will demonstrate your full consent for participation.

The results of this survey could result in a request to participate in the second part of the study, which includes one to two classroom observations and interviews. Participation in further study will be completely voluntary as well.

Potential Benefits

There may be no direct benefit to you for taking part in this survey. It is hoped that the research will provide the opportunity to re-examine and strengthen your personal knowledge of coding and computational thinking and the uses of this knowledge in your teaching practice.



Confidentiality

All collected surveys will be stored electronically and stored on a USB memory drive; files will be encrypted to secure data. Participants will not be identified by name in any reports of the completed study. In publication or presentation of research results, pseudonyms will be used for school district names. The USB memory drive will be stored in a locked cabinet inside of my private locked office at home. All data will only be accessed by my supervisor and myself.

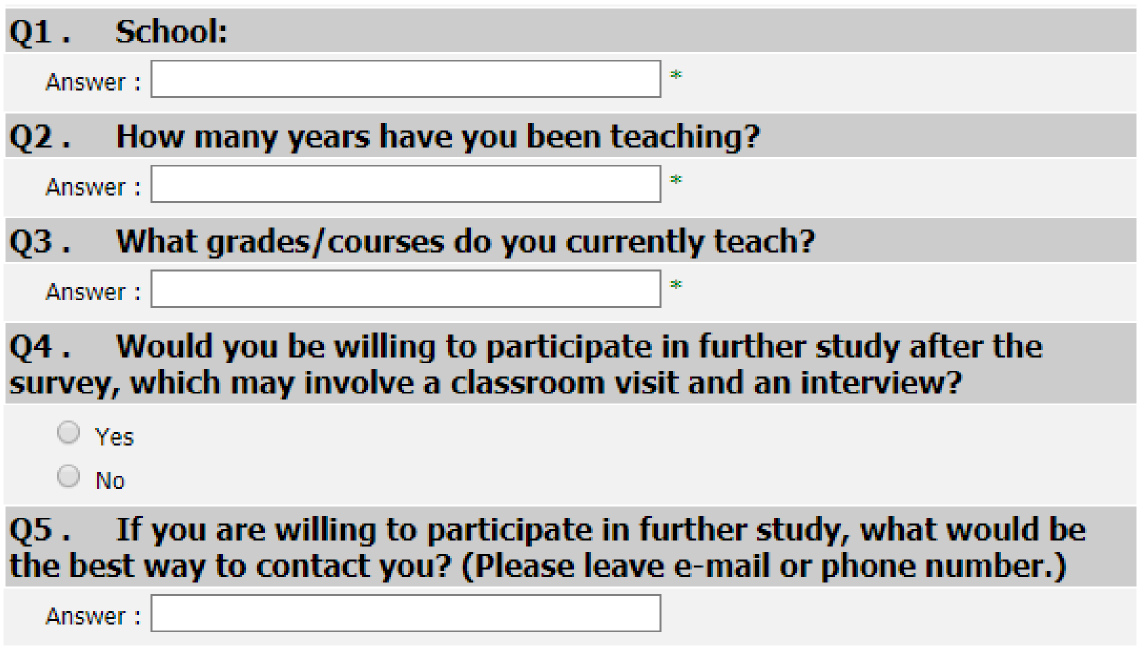
Study results

The main study findings will be published as my thesis, and possibly some of it will be published in academic journal articles. The main study findings could be presented at academic and/or professional conferences for teachers.

Contacts

If you have any questions related to the study, you can contact Erica Huang at [redacted]

If you have any concerns about your rights as a research participant and/or your experiences while participating in this study, you may contact Dr. Jeffrey Toward, Director of Office of Research Ethics, at [redacted].



Q1 . School:

Answer : *

Q2 . How many years have you been teaching?

Answer : *

Q3 . What grades/courses do you currently teach?

Answer : *

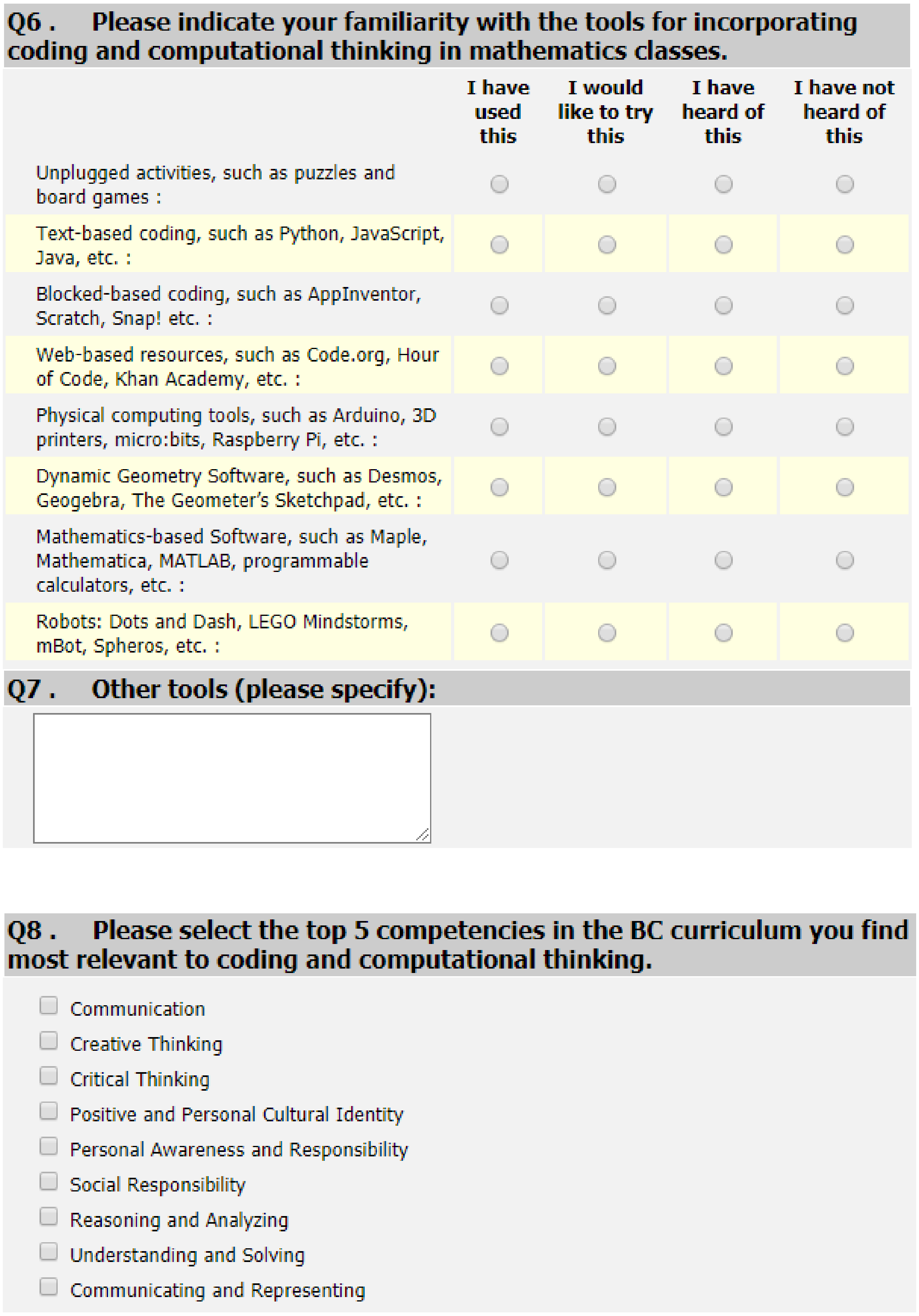
Q4 . Would you be willing to participate in further study after the survey, which may involve a classroom visit and an interview?

Yes

No

Q5 . If you are willing to participate in further study, what would be the best way to contact you? (Please leave e-mail or phone number.)

Answer :



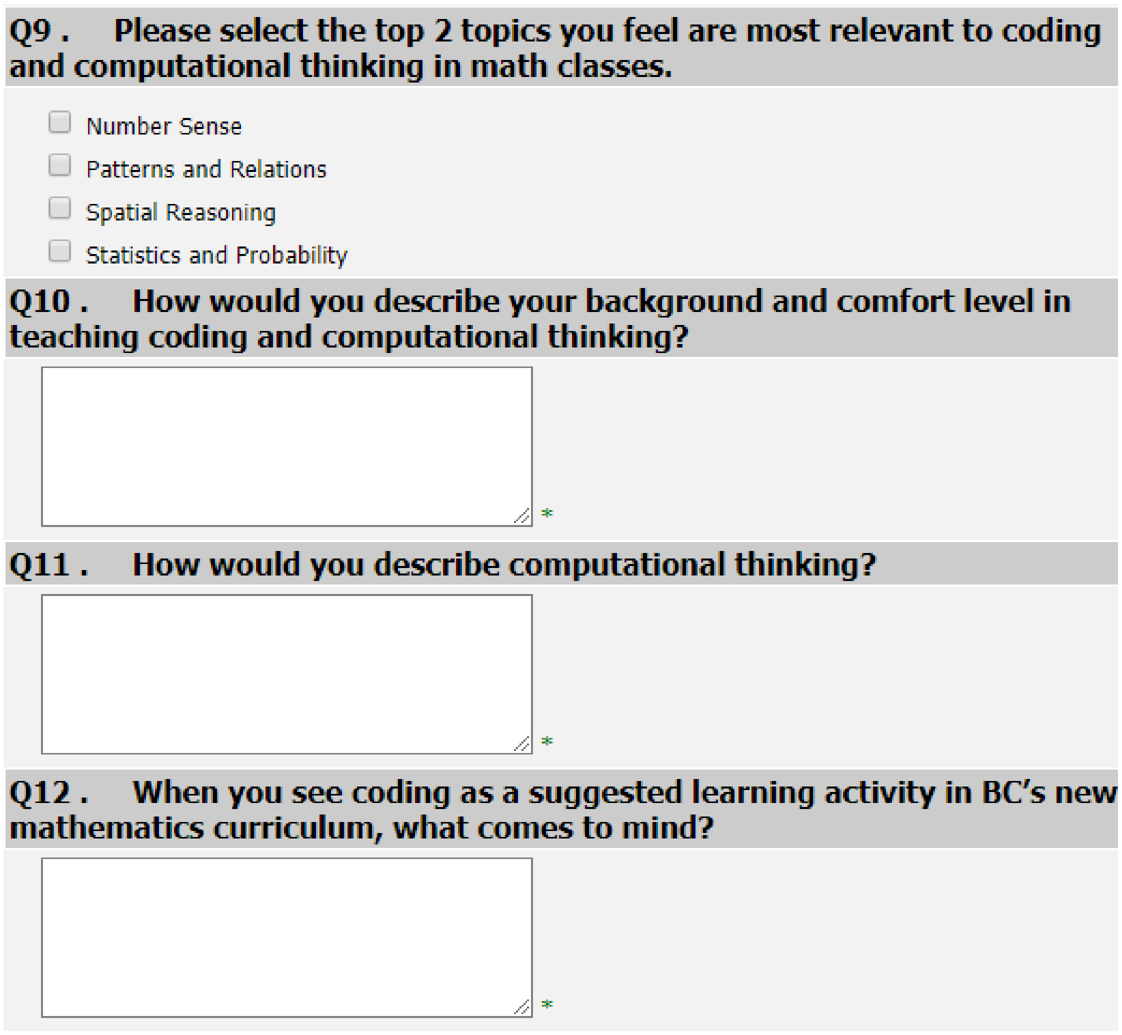
Q6 . Please indicate your familiarity with the tools for incorporating coding and computational thinking in mathematics classes.

	I have used this	I would like to try this	I have heard of this	I have not heard of this
Unplugged activities, such as puzzles and board games :	○	○	○	○
Text-based coding, such as Python, JavaScript, Java, etc. :	○	○	○	○
Blocked-based coding, such as AppInventor, Scratch, Snap! etc. :	○	○	○	○
Web-based resources, such as Code.org, Hour of Code, Khan Academy, etc. :	○	○	○	○
Physical computing tools, such as Arduino, 3D printers, micro:bits, Raspberry Pi, etc. :	○	○	○	○
Dynamic Geometry Software, such as Desmos, Geogebra, The Geometer's Sketchpad, etc. :	○	○	○	○
Mathematics-based Software, such as Maple, Mathematica, MATLAB, programmable calculators, etc. :	○	○	○	○
Robots: Dots and Dash, LEGO Mindstorms, mBot, Spheros, etc. :	○	○	○	○

Q7 . Other tools (please specify):

Q8 . Please select the top 5 competencies in the BC curriculum you find most relevant to coding and computational thinking.

- ☐ Communication
- ☐ Creative Thinking
- ☐ Critical Thinking
- ☐ Positive and Personal Cultural Identity
- ☐ Personal Awareness and Responsibility
- ☐ Social Responsibility
- ☐ Reasoning and Analyzing
- ☐ Understanding and Solving
- ☐ Communicating and Representing



Q9 . Please select the top 2 topics you feel are most relevant to coding and computational thinking in math classes.

- ☐ Number Sense
- ☐ Patterns and Relations
- ☐ Spatial Reasoning
- ☐ Statistics and Probability

Q10 . How would you describe your background and comfort level in teaching coding and computational thinking?

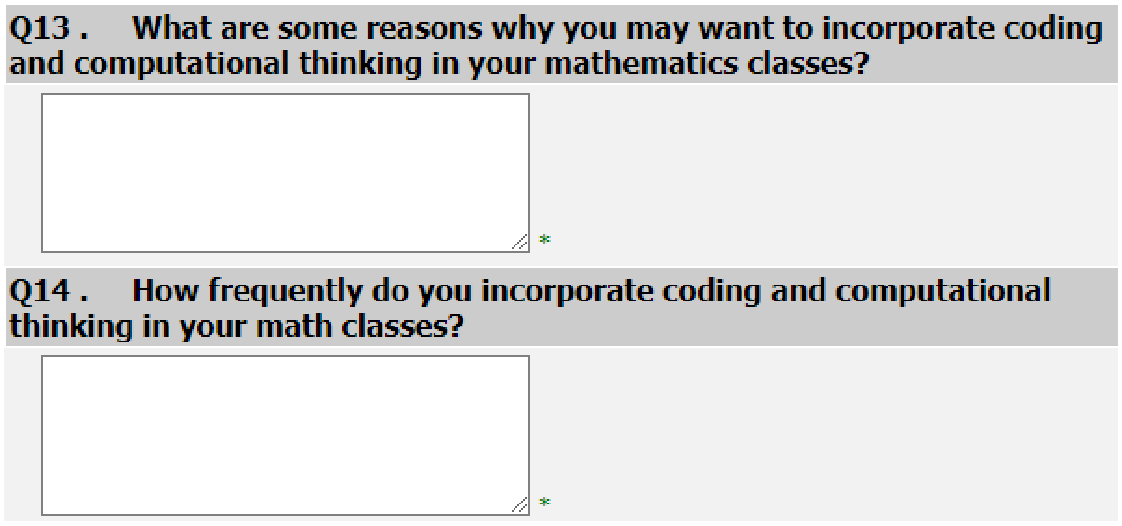
*

Q11 . How would you describe computational thinking?

*

Q12 . When you see coding as a suggested learning activity in BC's new mathematics curriculum, what comes to mind?

*

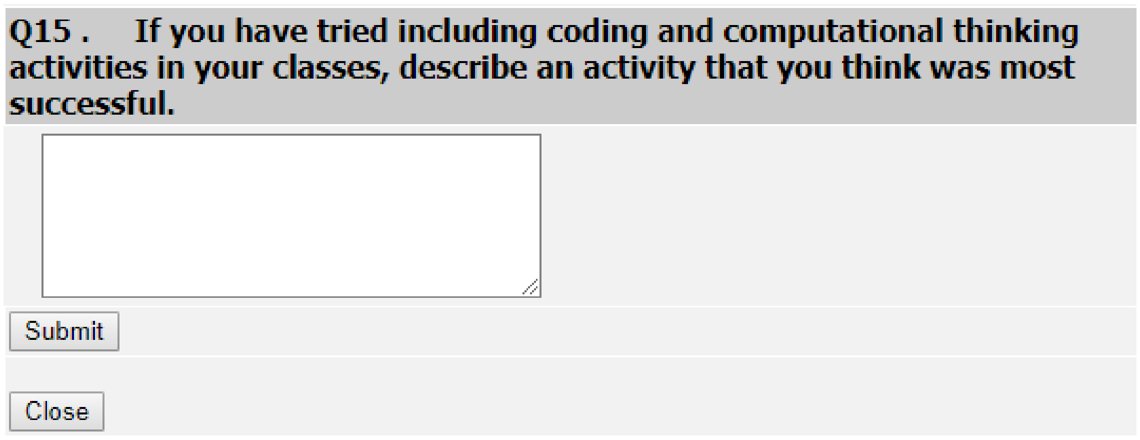


Q13 . What are some reasons why you may want to incorporate coding and computational thinking in your mathematics classes?

*

Q14 . How frequently do you incorporate coding and computational thinking in your math classes?

*



Q15 . If you have tried including coding and computational thinking activities in your classes, describe an activity that you think was most successful.

Appendix B.

Post-Observation Teacher Interview Questions

General Questions

- Overall, what did you like about this lesson?
- What challenges did you have, if any?
- If you can make one change to the lesson, what would it be?

Working Environment

- If you can make one change to your classroom setup for this lesson, what would it be?

Resource System

- What resources for coding and computational thinking did you use for this lesson?
- What resources for the math learning outcomes addressed did you use for this lesson?

Activity Format

- How did you structure this lesson?
- Comparing to your typical math lessons, what do you think are some differences in your lesson that incorporated coding, such as in the sequence of the lesson, assessment, etc?

Curriculum Script

- What are some changes you made in your way of teaching, such as questions posed, explanation of a concept, etc. teaching the same math concept in your lesson that incorporate coding and computational thinking?

Time Economy

- How much time did you spend planning this lesson?
- Are there parts of the lesson you feel were not effective use of time due to the use of technology?