



Elementary Students Learning Computer Programming: an investigation of their knowledge Retention, Motivation, and perceptions

Tian Luo¹ · Jilian Reynolds¹ · Pauline Salim Muljana¹

Accepted: 14 April 2022 / Published online: 6 July 2022
© Association for Educational Communications and Technology 2022

Abstract

Students need to learn and practice computational thinking and skills throughout PreK–12 to be better prepared for entering college and future careers. We designed a math-infused computer science course for third to fifth graders to learn programming. This study aims to investigate the impact of the course on students' knowledge acquisition of mathematical and computational concepts, motivation, and perceptions of the computing activities. Fifty-one students at a Boys and Girls Club participated in the study. Data collection procedures include pre- and post-tests, pre- and post-surveys, in-class observations, and one-on-one interviews. Results indicate that students have improved significantly on mathematical and computational concepts. They also tended to believe computer programming is fun, comprehensible, enjoyable, and were able to perceive the value of learning it. Implications and recommendations for future research are also discussed.

Keywords Elementary education · Computational thinking · Computer science education · Coding · Programming

Introduction

Computer programming instruction incorporates mathematical, physical, and process simulation combined, while also placing students in a programmer's role to explore possible careers involving science and discovery, space exploration, drone operation, cybersecurity, and other invaluable possibilities. Researchers and educational practitioners, as well as policy makers, repeatedly urge that waiting until students are in a four-year college to learn computer programming is no longer a viable option (State of Computer Science Education, 2021; Armoni 2012; Kumar, 2014; Prottzman, 2014). Students will need to learn and prac-

✉ Tian Luo
tluo@odu.edu

¹ Old Dominion University, Norfolk, Virginia, United States

tice computational thinking and skills throughout PreK-12 to be better prepared when entering their college degree programs or careers related to computer science and technology.

A multitude of visual and block-based programming tools and applications, such as Scratch, have enabled the learning of programming to become more commonplace and friendlier for children (Armoni et al., 2015; Grover et al., 2015; Gutierrez et al., 2018; Lewis, 2010; Maloney et al., 2008; Maloney et al., 2010; Weintrop & Wilensky, 2017). The growing body of literature related to computer science education in a K-12 setting suggests that young children are capable of learning computational concepts and practices at an early age (Namukasa et al., 2015; Rich et al., 2019; Tran, 2019; Saez-Lopez et al., 2016). This puts early computing education at the forefront and creates an imperative need to develop an informed body of knowledge about learning and teaching computer science and computational thinking in an elementary education setting. Although there are several connections among math and programming and highly correlated relationships between computer programming and success in mathematics (Bubnó & Takács, 2019; Razak & Ismail, 2018), this connection is uncommon in today's educational curriculum (Neri, 2021; Wright et al., 2013). Current literature highlights the needs for the interdisciplinary integration between computer science and mathematics education (Bubnó & Takács, 2019; Fisler et al., 2021), suggesting a knowledge gap on the integration of both disciplines (Powers & Azhar, 2020).

Benefits of programming and computer science instruction

The benefits of computer programming have been long documented in the United States since Seymour Papert incorporated Logo programming in elementary education in the 1970s to the 1990s. When Logo programming tools were made accessible to young learners, they created a powerful learning experience for these elementary learners (Harel & Papert, 1990; Papert, 1980; Papert et al., 1979). Later research consistently reported learning gains in the areas of improved geometric knowledge, logical reasoning, and spatial ability through the use of Logo (Clements, 2002; Clements et al., 2001; Subhi, 1999).

In the 21st century, teachers have increasingly been asked to teach computer science concepts and skills to younger children and integrate it into learning activities (Barr & Stephenson, 2011; Fluck et al., 2016; Google Inc. & Gallup Inc., 2016; Powers & Azhar 2020). This call was rooted in a multitude of research demonstrating various areas of impact for universal computer science education in the society: (1) economic and workforce development (2) equity and social justice, (3) competencies and literacies, (4) citizenship and civic life, (5) scientific, technological, and social innovation, (6) school improvement and reform, and (7) fun, fulfillment, and personal agency (Vogel et al., 2017). Vogel et al. (2017) underscored that computer science presents an interesting, innovative challenge, which evokes curiosity among K-12 students. As a pedagogical practice, it increases higher-order thinking skills and strengthens problem-solving capabilities and stamina (Atmatzidou & Demetriadis, 2016; Lee & Cho, 2019; Matere et al., 2021). These practice-based computer science lessons allow for hands-on, project-based learning.

Visual programming tools and environments such as Scratch, Scratch Jr., Snap!, and App Inventor are becoming increasingly popular in K-12 educational contexts (Grover & Pea 2013; Flannery et al., 2013; Morelli et al., 2011; Saritepeci, 2020; Scherer et al., 2020). Saez-Lopez et al. (2016) performed a quasi-experimental study, which highlighted

significant improvements regarding the learning of programming concepts, logic, and computational practices among 5th- and 6th-grade students utilizing the visual programming language—Scratch. Middle schoolers often have progressed to more game-based and creation activities (Garneli, 2015). When encouraged to high-school students, learning the programming concepts may support metacognition, further promoting problem-solving skills and creativity (Gim, 2021).

Computer Science Integration in Elementary Education

Despite the multiple benefits of computer science instruction, such incorporation into an existing K-12 curriculum is not without challenges. While computer science has been an increasingly common subject area in secondary education, this subject has mostly been taught in high school levels through advanced placement program (Rich et al., 2019), as computer science or programming as a skillset is not often deemed as age-appropriate for elementary learners. Therefore, a more inclusive and all-encompassing concept, *computational thinking* (CT), has progressively gained traction amongst researchers and practitioners (Lye & Koh, 2014; Rich & Hodges, 2017; Rich et al., 2019; Shute et al., 2017).

Although there has been a slew of multiple definitions describing computational thinking (Shute et al., 2017), the majority of literature on CT falls back on Jeannette Wing's definition emphasizing that CT being the way of thinking at multiple levels of abstraction involved in formulating problems and solutions applicable to all fields (Wing, 2006). Using the simplest terms, computational thinking is defined as thinking and solving problems like a computer, or to solving problems using a computational approach. Angeli et al., (2016) presented a computational thinking curriculum framework that incorporates indicators of competence for all given CT skills; namely: abstraction, generalization, decomposition, algorithmic thinking, and debugging. Teaching CT skills at a younger age denotes a way of teaching students to think about their day-to-day activities and solving problems algorithmically, which is a set of much broader fundamental skills than the specific skillset represented in the field of computer science. Researchers recommended that CT as a fundamental skill should be learned in early years of education so that application may continue throughout the student's educational journey (Barr & Stephenson, 2011; Coşar & Özdemir, 2020; Lu & Fletcher, 2009; Mladenović et al., 2021; Qualls & Sherrell, 2010).

While research shows decades of work regarding how to make programming more accessible to high school students, it is still unclear how to bring this content into elementary school classrooms (Weintrop et al., 2017). Many teachers at the elementary education level often shy themselves away from anything related to CT due to their misconceptions equating CT with coding. This in part may have been caused by administrators not preferring more coursework being added to present curricula, in fear of risking test scores and teacher apprehension (Burke, 2016). The wide varieties of technologies associated with computer science and CT (i.e., drag-and-drop type of coding, script-based programming, robotics) often make teachers uneasy and intimidated (Sadik et al., 2017; Mouza et al., 2018). Elementary teachers are reportedly facing exacerbating challenges, comprised of a lack of access to technology, inflexible curriculum, and inadequate planning time (Staples et al., 2005). Consequently, elementary teachers frequently struggle with seeing the CT and

computer science integration being connected to the subject areas that they teach in the classroom.

Although companies and organizations like ISTE, Code.org, and Google have produced numerous resources, most materials and resources are geared towards pull-out programs rather than integration; meanwhile, studies repeatedly call for integration into the existing curriculum as a more effective method of teaching CS as compared to teaching as a standalone subject course (Barr & Stephenson, 2011; Garneli et al., 2015). The multitude of coding modules and lessons available online may also seem overwhelming. This has caused confusion and frustration amongst educators about ways to create the best integration strategies for already existing standards or subject areas in order to achieve the highest levels of success (Denning, 2017; Garneli et al., 2015).

The aforementioned challenges may also be due to a perception that programming concepts are found to be difficult and abstract (Akinola, 2015; Noh & Lee, 2020; Sáez-López et al., 2016). However, learning a complex subject can be facilitated effectively through a suitable instructional approach (e.g., scaffolding) (Caglar et al., 2018). For example, elementary students—who are typically 7 to 11 years old—may struggle from learning an abstract concept as they start to develop logic reasoning (Piaget, 1964). Therefore, selecting and sequencing suitable instructional tools and programming concepts are imperative and should align with the students' context (e.g., age and grade level) (Mladenović et al., 2021).

Students can be introduced to the foundation of programming concepts through block-based coding tools (e.g., Scratch and code.org) initially. Particularly, this type of coding tools can assist students in visualizing the concept, manipulating the elements, and comprehending the programming logic through the blocks, rather than memorizing and recalling the coding script (Lambić et al., 2020; Rodríguez-Martínez et al., 2020; Vasconcelos & Kim, 2020) that may cause cognitive load for early programming learners (Lye & Koh, 2014; Vasconcelos & Kim, 2020). Essentially, these tools provide a visual representation of the programming concepts that students may otherwise perceive as difficult and abstract (Mladenović et al., 2021) and promote user engagement because they provide instant visual feedback after the code execution (Lye & Koh, 2014; Vasconcelos & Kim, 2020). As a result, learning programming can be interesting, enjoyable, and motivating, triggering students' desire to learn more (Coşar & Özdemir, 2020; Kumar, 2014; Lakanen & Kärkkäinen, 2019; Lambić et al., 2020) and promoting a career aspiration in a related discipline (Lakanen & Kärkkäinen, 2019).

Once the students have the knowledge foundation, they can try a more complex programming activity, such as by learning the programming language syntax or the textual programming. As a start, teaching the programming language may utilize Python (Mladenović et al., 2021). It is deemed appropriate to introduce the programming language to beginners such as K-12 students (Lee & Cho, 2019; Mladenović et al., 2021).

Bridging Computer Science and Mathematics Education

Prior research has suggested various correlations between students' learning of mathematical concepts and abilities and their learning of computer programming (Bubnó & Takács, 2019; Clements, 2002; Clements et al., 2001; Mladenović et al., 2021; Razak & Ismail, 2018; Relkin et al., 2021; Rich et al., 2013; Tran, 2019), which motivated educators to

use the Logo Programming Language and its connection to geometry (Burke, 2016) Burke (2016) and Gim (2021) contended that coding is in essence grounded in mathematics. Wright et al., (2013) described a full, in-school curriculum and software package called Bootstrap, which taught students how to program their own video games while making connections to algebra (Schanzer, 2015). This study suggested an increased students' understanding of algebraic functions and variables, while also presenting evidence of using Racket programming language to energize students' math learning. Other studies similarly highlighted the effect of teaching programming using Scratch. Scratch can provide students a new perspective regarding math-related concepts and processes (Benton et al., 2017; Calder, 2010; Hughes et al., 2017), potentially because it promoted the skills needed to comprehend math such as critical thinking and metacognitive skills (Rodríguez-Martínez et al., 2020). It is not a surprise some teachers may perceive the connection between programming concepts and math and science instructions (Neri, 2021; Rich et al., 2019). Basawapatna et al. (2010) emphasized that the implementation of computer science concepts while participating in video game creation activities could be used as a springboard to advancing their computational thinking. Overall, both middle-school and high-school students can take advantage of learning programming to boost their computational thinking skills (Noh & Lee, 2020), in with mathematical abilities play an imperative role in learning programming (Soboleva et al., 2021).

Students were also reported to enjoy, remain motivated, and build confidence in the programming-infused learning environments (Coşar & Özdemir, 2020; Gim, 2021; Hsu et al., 2018; Lambić et al., 2020; Romero & Lepage, 2017; Scherer et al., 2020; Saritepeci, 2020; Tran, 2019). In the Basawapatna et al. (2010) study, when asked if they enjoyed designing games on the computer, 100% of surveyed students chose the response, "Strongly agree." Similarly, student participants, who were third-grade students, in Tran's (2019) study enjoyed the computational thinking learning activities due to the unique learning opportunity that was not typically found in traditional lessons. Lambert & Guiffre (2009) indicated improved confidence and interest in Computer Science and Math seen in elementary school children after conducting a computer science outreach containing a series of "unplugged" computer science lessons that did not involve a physical computer. Meyer and Batzner (2016) conducted a study consisting of 450 9- to 10-year-old students from 22 to 33 elementary schoolers. The results concluded that all of the participants enjoyed the course and would like to participate again. Belanger et al., (2018) reported a study showing an increase in student confidence in categories related to the ability to do math, the ability to give directions and the ability to someday build a computer after teaching three common lessons of computational thinking. Saez-Lopez et al. (2016) also reported students' perceived enjoyment of Scratch-supported computer science activities and enhanced motivation, in addition to their improved understanding of computational concepts and practices. Although programming lessons may be initially perceived challenging, elementary students can potentially change their misconceptions about the programming concepts and thereby enjoying learning it, feeling accomplished by the end of a lesson, and desiring to continue their learning path (Gim, 2021).

Since programming is related to mathematical concepts (Bubnó & Takács, 2019; Clements, 2002; Clements et al., 2001; Mladenović et al., 2021; Niemelä & Helevirta, 2017; Razak & Ismail, 2018; Relkin et al., 2021; Rich et al., 2013; Tran, 2019), it is imperative to help students recognize the transferable skills they gain from learning programming to

learning math. Transfer of learning is deemed essential as it can transform students' understanding of the concept to realizing the connection with other contexts, promoting their metacognition and meaningful learning (Niemelä & Helevirta, 2017). As a result, students will potentially be able to grasp the common fundamental concepts of different domains and apply them in various contexts (Niemelä & Helevirta, 2017). Racket can be utilized to teach programming and simultaneously assist students to focus on transferring knowledge between math and programming and vice versa (Niemelä et al., 2017). Racket, a math-friendly functional programming language, offers image representation allowing students to visualize and manipulate algebraic expressions, and can pique the interests of elementary students (Felleisen & Krishnamurthi, 2009; Niemelä et al., 2017). However, some teachers may find it too complex to be used for teaching programming to elementary students (Niemelä & Helevirta, 2017). Therefore, the suggested learning path, if multiple coding tools are used, is to utilize Scratch, Python, and then Racket, which would be appropriate for early programming learners like the elementary students under study (Niemelä & Helevirta, 2017).

Purpose of the study

Extant literature has painted a mixed picture documenting both the successes and challenges of teaching CT and integrating computer science and programming in elementary education contexts (Fisler et al., 2021; Franklin et al., 2017; Manches & Plowman 2017; Seiter, 2015). Despite that computer science increases motivation, attitudes, and content retention among K-12 students, little research examines such an effect on elementary learners (Lambić et al., 2020). Despite the natural link between CT and mathematics and the needs for the interdisciplinary integration (Bubnó & Takács, 2019; Fisler et al., 2021), such an integration is still rarely practiced in educational curriculum (Neri, 2021) which suggests a significant knowledge gap (Powers & Azhar, 2020). In this study, we incorporated computer science instruction to strengthen math content connections among elementary school students. This study will offer insight as to what connections among the learning of basic computer science and programming tasks may affect the retention of mainstream curricula in math in addition to possibly increasing student motivation. The study was guided by three research questions:

- RQ1: To what extent did students' knowledge acquisition of mathematical and computational concepts improve after the computing activities?

Ho: Students' knowledge acquisition scores of mathematical and computational concepts did not improve after the computing activities.

- RQ2: To what extent did students' motivation change after the computing activities?

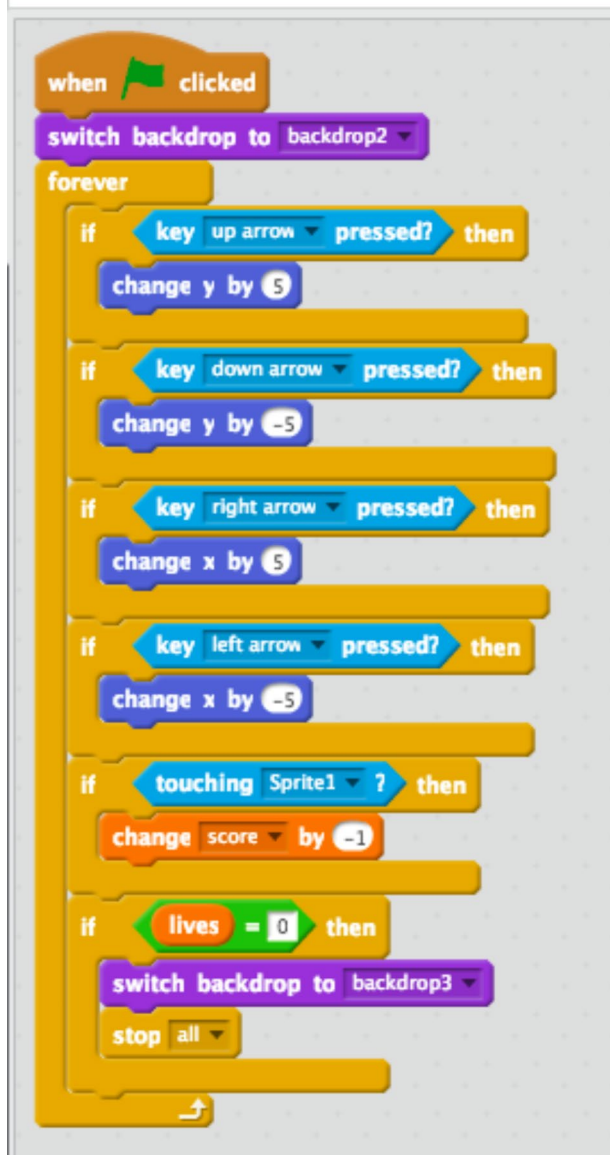
Ho: Students' motivation scores did not increase after the computing activities.

- RQ3: How did elementary students perceive these computing activities?

Methods

A mixed methods approach was conducted to glean insight into the impact of the coding activities and participants' overall experiences. The sequential explanatory strategy emerged as the qualitative data built upon the results of the quantitative data (Creswell & Clark, 2017).

Fig. 1 A screenshot showing the Scratch interface



Participants

A total of 51 students who were elementary learners belonging to two different sites of a Boys and Girls Club located in the southeastern U.S. participated in this study. The participants were drawn from a convenience sampling as one of the authors was an in-service teacher in the same school district. These participants attended the two local elementary schools, which repeatedly reported low standardized assessment scores in math. Amongst the 51 participants, 37.3% were third graders, 33.3% were in the fourth grade, and 29.4% were fifth graders. Approximately 98% of all students were considered *low-income* and received free or reduced lunch. In terms of racial makeup, the majority of the student sample population (90%) were Black, 2% Caucasian, 4% Hispanic, and 4% multiracial. Approximately 51% of students were females and 10% of students were classified as learning disabled and/or autistic.

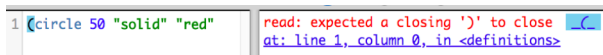
Materials

In this study, students were exposed to three programming languages: Scratch, Python, and Racket. Scratch bridges the gap between block-based programming and more advanced text-based programming languages and provides students with no programming experience an innovative and interactive environment, allowing them to create programs, stories, and games using drag-and-drop blocks (See Fig. 1). Python is a suitable language for teaching beginners. One of the programs taught in this study is shown in Fig. 2 where students created a function, which drew an angle of choice. Students had to figure out that in order to produce the correct angle, they had to write the code to subtract their chosen angle from 180, which represents a straight line. Racket is a math-friendly functional programming language, which allows students to visualize algebraic expressions and therefore helps them easily transition from visual block programming languages to more sophisticated programming languages. In this study, Racket was extremely beneficial when students were asked to create algebraic expressions using a scale factor. Additionally, when students utilized the Wescheme web-based program using Racket, any errors or “bugs” in their code are explicitly identified and the student is then able to easily make corrections (See Fig. 3). The

```
1 from turtle import *
2
3 turtle1 = Turtle()
4
5 def drawAngle(degrees):
6     turtle1.forward(100)
7     turtle1.left(180 - degrees)
8     turtle1.forward(100)
9
10
11 drawAngle(60)
12
```



Fig. 2 A screenshot showing the Python interface



```
1 | circle 50 "solid" "red"
   | read: expected a closing ')' to close
   | at: line 1, column 0, in <definitions>
```

Fig. 3 A screenshot showing the Wescheme web-based program

design of the instruction was partly guided by the Bootstrap curriculum (Schanzer, 2015) while continuing the ideas of transfer to specifically address math concept retention and connections.

The nine, one-hour session lesson was designed to promote transfer as shown in Table 1. Video tutorials were used for three out of the nine instructional lessons to facilitate and reinforce students' continuous content retention. During these lessons, students were taught how to write different examples of code to make math connections and how to create their own working computer programs. Table 1 demonstrates the learning objectives and the activities students were engaged in during the lessons.

Instrumentation

Knowledge retention. The knowledge retention test instrument was strategically crafted to mimic the 4th - and 5th -grade district-pacing guide content in addition to the Common Core State Standards for math. The 22-item test instrument included multiple choices and short answers questions targeting the following math and computer science content areas: the coordinate plane, variables, data types, syntax, algebraic expressions, word problems, angles, binary conversions, algorithms, loops, and debugging (See Table 2). The reliability coefficient of the knowledge retention test was calculated at 0.654.

Motivation. A 9-item Likert-scale survey instrument was used to measure students' motivation. Two items adapted from Papastergiou (2009) was used to measure any difference in student study motivation and feelings toward school in general. One item specifically addressed students' motivation in math. The remaining six items were adapted from Belanger et al., (2018), which addressed students' perceived usefulness of computer science and their confidence level with coding. To make the survey content more readable to elementary students, emoji images were used in combination with the text options. For example, a smiley face emoji was used for the agreeable options and a sad face was used for the disagreeable options. The reliability coefficient in the motivation survey was calculated at 0.736.

Procedures

The researchers first recruited participants from the local Boys and Girls Club based on the following criteria: (a) students who were daily Boys and Girls Club members, (b) students who attended the two local public elementary schools, (c) students recently completing the 3rd, 4th, or 5th grade. One researcher then attended a parent meeting to personally invite the students to participate in the computer science course. A pre-test of knowledge retention and pre-survey of motivation were distributed to all 51 students before the course commenced. In the next three weeks, two instructors taught a total of nine face-to-face classes and conducted activities to achieve the objectives listed in Table 1. After the course concluded, all 51 students participated in a post-test and post-survey. All students were also invited to participate in the one-on-one interviews to obtain richer information about student's percep-

Table 1 Math and Programming lesson objectives

Lesson	Learning Objectives	Activities	Concepts Learned
1	To “understand that the first number in an ordered pair indicates how far to travel from the origin in the direction of one axis and the second number indicates how far to travel in the direction of the second axis” (CCSS.Math.Content.5.G.A.1) by using Scratch to show a video game character moving on a screen	Students learned how to login to the Scratch web-based program via scratch.mit.edu . Students were taught basic character movement on x and y axis. Students learn to visualize their own video game existing on an imaginary coordinate plane.	Coordinate plane
2	To simulate a “live video game” by moving around the room on “imaginary x and y axes.”	Students completed the basic video game and discuss code meaning. Students took steps to the right to show moving in a “positive direction” on the x axis, left to represent a “negative direction” on the x axis, reached up in the air to represent “positive y” and bent down to the floor to represent “negative y.” Students added a background screen to their games in addition to writing the code for the antagonist character and the “prize character.”	Code syntax
3	To create overlapping shapes with loops in Python while typing text-based code and troubleshooting error messages	Students were introduced to Python turtle drawings. Students were then taught, through discussion, what would happen if this code repeated, but the second and third time, it changed the starting position or angle? Students were given a loop code to add to their original code. Students learned to debug erroneous codes in Python.	Loops, debugging
4	To understand variables and continue to analyze error messages while creating a program in Python where the computer acknowledges user input and displays it in the response	The concept of a variable was reviewed. Students were given the option to chance the letter “n” to any letter to represent “a number.” Students were given more opportunities to enter and debug codes. Student were presented with the concept “input.”	Data types and variables
5	To create a program in Python, which simulates the drawings of right, acute, and obtuse angles (CCSS.Math.Content.4.G.A.1)	Students were taught the code that draws an angle. Students were presented with the challenge to figure out why the turtle drew a 135 degree angle when the program showed the turtle turning 45 degrees.	Angles

Table 1 (continued)

Lesson	Learning Objectives	Activities	Concepts Learned
6	To “convert among different-sized standard measurement units within a given measurement system (binary to decimal and decimal to binary), and use these conversions in solving multi-step, real world problems” (CCSS.Math.Content.5.MD.A.1)	Students viewed a short video explaining binary numbers and then presented with numerous examples. They were then given opportunities to practice and complete converting binary numbers to their decimal representations.	Binary Conversion
7	To describe how to create a type of sandwich and analyze how this relates to computer algorithms.	The students were asked to sit in a circle while the instructor stated the first step involved in making a peanut butter and jelly sandwich in order to illustrate the concept of algorithms. Each student was then responsible for stating the next step in the process and engage in discussions to understand loops (repeated) and conditionals (if/else).	Algorithms, loops
8	To use the Racket Programming Language to “understand that shapes in different categories may share attributes. To recognize rhombuses, rectangles, and squares as examples of quadrilaterals” (CCSS.Math.Content.3.G.A.1)	Students were taught the concept of a string in computer science and experimented with the Racket programming language and the Wescheme environment. Students learned the code for a circle, and then experimented with figuring out the codes for other shapes such as rectangle and triangle and creating a national flag that has a circle.	Algorithms Debugging
9	To “write simple expressions that record calculations with numbers, and interpret numerical expressions without evaluating them. (CCSS.Math.Content.5.OA.A.2) To “compare the size of a product to the size of one factor on the basis of the size of the other factor, without performing the indicated multiplication” (CCSS.Math.Content.5.NF.B.5.A)	Before each math problem was completed, the students tested out the Racket code in the Wescheme environment to represent the math problems. They completed several math problems such as to write an algebraic expression to determine the size of the circle.	Algebraic expression, Debugging

Note. The corresponding Common Core State Standards are noted in parenthesis. A more detailed lesson description can be found at <https://tinyurl.com/yr8r3f77>

tions. The interviews were conducted after the post-test and post-survey were administered. A total of 46 students were interviewed, each of which lasted approximately 4–10 min.

Data Analysis

To investigate the effect of the computer science instruction on content retention and motivation (RQ1 and RQ2), we first used descriptive statistical analyses in SPSS to compare means and standard deviations of the test and survey items and then conducted a series of

Table 2 Structure of the Knowledge Retention Test Instrument

Category	Test Items	Topics	Short code
1	Question # 4, 5, 6	Coordinate plane	CP
2	Question # 7, 8, 9, 20	Python Data types and variables	Python
3	Question # 10,18,19	Code syntax	Code
4	Question # 11	Word Problems (Algebraic Expressions)	WP_AE
5	Question # 12, 17	Word Problems (Python turtle)	WP_PT
6	Question # 13,14, 24	Angles	Angles
7	Question #15	Binary Conversion	BC
8	Question #16	Algorithms	Algorithms
9	Question # 21, 25	Loops	Loops
10	Question #22	Debugging	Debugging
11	Question #23	Creating Algorithm in block code	CA

Note. The first three items on the test were demographic questions

paired *t*-tests to examine to identify any statistical differences from students' test scores and survey ratings between the pre- and post-conditions. Several assumptions needed to be met prior to running the *t*-test for repeated measures. The assumption that the differences between the pre- and post-tests have no outliers was tested by a visual inspection of a box plot, and no outliers were detected. The Shapiro-Wilk Test of Normality was significant ($p=.039$). Kolmogorov-Smirnov test of normality was not significant ($p=.085$) and the repeated measures *t*-test is reasonably robust against violations of normality (Wiedermann & von Eye, 2013). The nonparametric equivalent of the repeated measures *t*-test (Wilcoxon Signed Ranks Test) was also performed on each category based on the content areas due to violations of normality.

To better understand participants' perceptions of the computer science instruction, student interviews were recorded, transcribed, and analyzed using an open-coding approach (Patton, 2002). Two researchers independently extracted patterns and themes from interview transcripts and further organized the data into meaningful categories. Quotations from the interviews were extracted to provide further insight into students' perceptions of the computer science instructions. The two researchers then met and discussed about the disagreements and made decisions on the final themes to be included in the paper.

Ten interview questions were initially organized in the following categories: learning, content, and motivation. The *learning* category referred to students' general perceptions of their experience in the computing lessons and activities, including what they liked or disliked about the class. It also inquired students' learning preferences, what they learned, how well they learned, and whether they prefer typing the code or using the block-code method. If students stated they preferred typing the code, they were asked what they remembered creating after they had typed their code. The *content* category referred to several questions where students were assessed on certain content areas, such as describing an algorithm, providing a specific example of an algorithm and state what specific problem their algorithm solved, as well as explaining the relevance between coding and math. The *motivation* category asked students' motivation to code, how confident they perceived themselves in cod-

ing, and their perceived usefulness of programming in the future. The last question elicited ideas for change and encouraged personal suggestions.

A stage-by-stage analysis approach (Burnard, 1991) was used to create coding themes for qualitative data analysis. In the initial coding stage, notes and memos were created to accompany the interviews. Themes started to become identified throughout the notes and memos. Open coding where actual categories started to form occurred. Final lists of categories were produced and to guard against researcher bias, and the category lists were shared to assist with determining three specific emerging themes. Interviews were again compared to the three finalized themes that were modified from the initial categories, including: (a) experience: overall experience in the computing activities, (b) learning: how well partici-

Table 3 A Matrix displaying conceptual categories, interview questions, and excerpts

Category	Opening Question	Probing Questions	Excerpts
Experience	How was your experience during the coding classes?	Were you excited to attend the classes each day? What do you like about the coding/programmer classes? What do you dislike about the coding/programmer classes?	“I like it. It was fun. I learn how to make game like create one that I want to create.” “Well, it was actually very interesting because I already knew about coding. I already figured it out using Khan Academy. But, this was different. Because I learned a lot more about new stuff that I didn’t know about coding. Like python. Never heard of it.” “I like it. Because I made game and made the shapes.” “Great. Coding angry bird.” “It was fun. Oh, I like making the video game. I chose a unicorn [character to make].”
Learning	How well did you learn during the classes?	Did you prefer the drag and drop method of coding or did you prefer typing the code? How would you describe an algorithm? How do you think coding relates to math?”	“I really enjoyed typing and making rectangles.” “I like typing in Python because I feel like a real coder.” “It was really fun when we created shapes in Weshceme. I loved making them different colors.” “The Siri program was my favorite. I had trouble with getting the code right, but I loved doing that stuff.” “An algorithm is like the steps to make a peanut butter and jelly sandwich because you are hungry. First you...then...” “To make the character go right, the number had to be positive and when he went left, it had to be negative.” “When we had to subtract the angle we wanted from 180 to get the correct code.”
Motivation	How excited, motivated, or eager were you during the computing classes? How may the coding/programmer classes help you in the future?	Did you surprise yourself by how much you learned throughout this course? Did you believe you could ever learn how to code in this way after taking the first assessment? Do you feel smarter now that you have learned how to write computer programs?	“I wanted to come to class every day! I was so excited about it!” “It was different, so it was interesting.” “It made me feel like a real coder.” “I didn’t think I could ever do something like that!” “I impressed myself.” “Some days I didn’t want to come because it was tough.”

pants learned during the computing classes and (c) motivation: to what extent they were motivated by the computing activities (See Table 3).

Trustworthiness

Patterns, themes, and categories emerged rather than being imposed to allow for inductive analysis (Patton, 2002). The researchers used triangulation of multiple data sources and multiple methods to increase trustworthiness, validity and credibility (Patton, 2002). In this study, we used data from pre- and post-assessments, semi-structured individual face-to-face interviews, and instructor in-class observations. The in-class observations were performed in a semi-structured manner where the instructor documented students' progress and their perceptions through all the coding activities. The observation checklist contains notes in each day's lesson, organized by activity, progress made, and challenges. Additionally, constant comparative analysis (Glasser, 1965) and a thick, rich description utilizing multiple forms of data within this study was provided with the intention of assisting readers to develop their own conclusions of the results (Lincoln & Guba, 1985).

Results

RQ1. To What Extent Did the Computing Activities Impact Students' Knowledge Acquisition of Mathematical and Computational Concepts?

Table 4 shows the means and standard deviations of student scores on the knowledge retention test. Overall, students' test scores have improved in all content areas according to the descriptive statistics. The post-test results had a statistically significant increase from the pre-test results, $M = -58.61$, 95% CI [-63.38, -53.84], $t(50) = -24.687$, $p < .001$. The effect size was measured using Cohen's $d = 4.057$. This effect size is considered medium (Sawilowsky, 2009).

Table 4 Pre and Post Scores of Students' Knowledge Retention

Content area	Pre-test		Post-test	
	<i>M</i>	<i>SD</i>	<i>M</i>	<i>SD</i>
CP	19.61	32.821	67.98	32.036
Python	5.88	11.820	71.08	23.115
Code	13.06	23.216	61.49	34.310
WP_AE	0.00	0.000	15.69	36.729
WP_PT	39.22	49.309	92.16	27.152
Angles	30.06	30.829	76.65	21.329
BC	0.00	0.000	90.20	30.033
Algorithms	13.73	34.754	56.86	50.020
Loops	17.65	31.343	86.27	26.605
Debugging	5.88	23.76	80.39	40.10
CA	1.96	14.00	64.71	48.26
Mean	15.14	13.61	73.75	15.24
Sum	147.04	141.28	763.47	169.30

Table 5 includes the content areas where the results of the Wilcoxon Signed Ranked test were significant, showing a significant improvement in test scores among those categories. The remaining content area data were not appropriate for significance testing given a lack of symmetry in the differences.

RQ2. To what extent did the Computing Activities Impact Students' motivation?

Table 6 shows the means and standard deviations of student ratings on the motivation survey. Descriptive statistics reveals that students perceived motivation of learning in general and their interest in computer science more favorably after the experiment. According to the survey items that showed a significant difference, more students tended to believe computer science was fun, useful, and comprehensible even at their level being an elementary schooler.

RQ3. How did Young Learners Perceive the Computing Activities?

Overall experience in the computing activities. All but two students demonstrated a positive view toward their learning experience in the computing activities. They described their experience as "great," "fun," "interesting," "exciting," "awesome," and "surprising." When asked if they were excited to come to class each day, 80% ($n=37$) of students stated that they enjoyed attending class. When asked to elaborate, four students added that they only wanted to not attend if they were in the middle of a sports game or fun outdoor activity.

When asked what they liked the most about the class, the vast majority of students ($n=44$) stated that they enjoyed being able to code and create their own games. They also

Table 5 The results of wilcoxon signed ranked test

Content area	Number of pairs	Increases	Decreases	Ties	Z	p-value
CP	48	41	1	6	5.50	0.000*
Python	51	51	--	--	6.31	0.000*
Code	48	38	4	6	5.34	0.000*
Angles	51	40	2	9	5.62	0.000*

Note. *Significant at the 0.05 level

Table 6 Pre and post scores of students' motivation

Item	Pre-survey		Post-survey		t	p
	M	SD	M	SD		
I believe I am a very smart kid.	3.69	1.06	4.39	0.65	5.01	0.001
I am excited to come to school every day.	2.70	1.75	3.84	1.20	5.15	0.001
I am good at solving word problems in math.	3.78	1.15	4.12	0.97	1.82	0.074
Computer science with coding is fun.	3.94	1.19	4.57	0.83	3.46	0.001
Computer science with coding is used in the real world.	4.25	1.04	4.57	0.81	1.88	0.066
I feel confident in my ability to create a computer program.	4.08	0.96	4.37	0.99	1.58	0.121
Computer science with coding is not too difficult for me to learn.	3.80	1.00	4.16	0.97	1.82	0.074
Computer science will be useful to me in the future.	3.86	1.13	4.35	0.80	2.57	0.013
I understand stuff about computer science.	3.73	1.13	4.41	0.83	3.67	0.001

Note. 1=No, never, 2=Usually no, 3=Neutral, 4=Yes, sometimes 5=Yes, always

liked the characters in Scratch, the different activities such as the moving angry bird, creating their own “Siri-like” experiences, and coding in a text-based environment. A student who favored the text-based programming said that “I might like to be into programming. I’d like to make my own game.” Another student expressed, “I like the Python because I’d like to learn a little bit more. I like how you learn how to program.” When asked what they disliked about the class, all students stated that there is nothing that they did not like. A few ($n=4$) expressed their desire to learn more. As one student said, “I just wanted it to be longer so we can learn more, even about designing a video game.” Two students even asked the class to be offered again in the future because they would like to attend again.

Learning in the computing activities. When asked how well they learned during the class, 13 students (28%) stated that they learned very easily, 39% ($n=18$) expressed it was just right, and eight students (17%) stated that they learned with some difficulties. Learning was evident in this study as the majority of students preferred the actual typing process of the text-based programming languages—Python and Racket—over the block-based, drag-and-drop programming activities like Scratch. When asked if the drag-and-drop block coding or the typing of text-based coding was preferred, 33 students (72%) preferred the text-based coding even though it was considered the more difficult method. Five students (11%) favored the Scratch Lessons because they were able to “choose their own game characters.” The remaining students ($n=5$) favored creating loops with Python. Overall, there were only two students (out of 46) who disliked using text-based programming.

When asked how they would describe an algorithm, twenty-five students (54%) successfully demonstrated a “real-world example” of an algorithm such as the steps to making a sandwich and 18 students (39%) gave a coding-process example, such as the Python code for creating a rectangle or the direction where a character should move. One student gave an example, “Algorithm is a code. Algorithm is a loop. Algorithm is like steps. It’s like the peanut butter and cereal things. You have to take cereals out. You have to pour the cereals into bowl. You have to put in spoon. You’re making it so you aren’t hungry.” Another student gave a different example, “An algorithm is when you make the character do what you want to do. When your character goes up, down, and side, and side. You make the character go around so that [the] thing doesn’t catch it.” Only two students were unable to answer the question. Although the question on algorithms was answered poorly in the post-test, most students were able to answer it correctly in the interviews.

When asked about the connection between coding and math, all students stated that they were closely related. Thirty-two students (69%) were able to accurately state relationships including connections to: coordinate plane, coordinates, x and y values, variables, algebra, numbers, addition, subtraction, multiplication, division, and angles. For example, one student said, “I saw the X - and Y -axis. And, that’s math. My character was a robot. He was moving up, down, and right [using hand gestures to additionally articulate the connection with X - and Y -axis].” Another student stated, “I think coding [is] related to math because you have to solve problems.”

Observations occurred during each session with the instructor keeping tallies as documentation while students progressed through the coding activities. It was evident that verbal frustrations most frequently occurred during the following lessons: Python (variables, data types) and Python loops and debugging. Students expressed frustration as they were asked to type the code using the computer keyboard. Verbal celebrations such as, “Yes!” “I got

it!” and positive cheering occurred more frequently during the following lessons: Scratch block-based coding and Racket (strings, shapes, and coordinate position).

Motivation. The vast majority of students ($n=37$) appeared to be very motivated by what they learned in the computing class. When asked if students surprised themselves and ever thought they would be able to learn how to code in this way, 61% of students ($n=28$) stated that they surprised themselves and they never thought they could code in this way. The remaining students stated that they always knew they could complete this task. Fourteen students commented on various tasks that they did not know they could do, such as the ability to be able to debug. One student commented, “I like to debug problem. I felt good [after being able to debug].” Another student commented, “Yeah, when we were doing the Python, I got one [bug.] [...] I know I can do it because I had to learn it well.” There was also another comment: “Yup, I didn’t give up [debugging]. It’s like I was fighting the bad guy and won.”

When asked if students felt smarter now that they have learned how to write computer programs, 16 students (35%) stated “Yes.” When asked how the coding classes may help them in the future, 20 students stated that they could help them “make video games,” and “do more with computer science.” One student commented, “It helped me a lot because it’s like I want to make game.” Another student commented, “When I get older, I actually want to have experience to know how to actually make a game and probably teach how to make a game.” Other students ($n=8$) noted that coding skills could help them “find a job” and “make money” because such skills would be needed in their future job market.

Discussion

As there has been rising hype about incorporating CT, computer science and coding surrounding the K-12 community with regard to both research and practice (State of Computer Science Education, 2021; Manches & Plowman 2017; Rich & Hodges, 2017; Rich et al., 2019; Shute et al., 2017; Vogel et al., 2017), we developed this study aiming to provide more empirical evidence showing the impact of computing activities primarily amongst younger-aged learners. The statistical findings provided insights into students’ learning achievement and conveyed significant increases in computer science and mathematical content retention, as well as motivation. The qualitative findings echoed the quantitative findings, presenting a variety of means through which students expressed a positive learning experience. These positive findings concur with previous studies where engagement, motivation and learning were seen amongst K-12 students involved in programming (Coşar & Özdemir, 2020; Gim, 2021; Hsu et al., 2018; Lambić et al., 2020; Meyer & Batzner, 2016; Lambert & Guiffre 2009; Saez-Lopez et al., 2016; Saritepeci 2020; Scherer et al., 2020).

Our study suggested evidence for the successful integration of CT and coding into math curriculum. The computing lessons and assessments were designed according to Common Core State Standards on math. By linking math content to hands-on, project-based coding activities, it enabled students to explore math content in a more meaningful and relevant way. Although in previous studies where students demonstrated difficulties in the comprehension of loops and algorithms (Belanger et al., 2018), our findings suggested that the learning potential of elementary school students should not be underestimated. With the visual stimuli and hands-on activities, the majority of students in the interviews were able

to verbalize an abstract concept (i.e., algorithms) and some provided concrete examples. Their knowledge retention of math increased significantly, and they were able to perceive the relevance between math and coding that further deepened their understanding of said math concepts while helping them realize the value of math in problem solving. This finding reconfirmed the potential of CT integration to invigorate math instruction, as evidenced in prior literature (Hickmott et al., 2018; Niemelä & Helevirta, 2017; Rodríguez-Martínez et al., 2020; Weintrop et al., 2016; Wright et al., 2013).

It is equally worth noting that the coding activities were not only fun and enjoyable, but they were also challenging to the students. Although students expressed the most difficulties generating the text-based code instead of dragging and dropping block-based code, most admitted that they enjoyed the text-based code. This may contradict previous literature suggesting visual-based programming environments were more age-appropriate for young learners (Bers & Horn, 2010; Lambić et al., 2020; Mioduser et al., 2009; Noh & Lee, 2020; Rodríguez-Martínez et al., 2020; Strawhacker & Bers, 2019; Tran, 2019; Vasconcelos & Kim, 2020). As students in this study were presented with three programming environments, they experienced both block-based and text-based coding. This may have been a representation of students voluntarily seeking a greater challenge, which resulted in deeper learning and further accomplishments. Therefore, students were motivated to learn not because they were fun, but they were also engaged in an adequate amount of challenge prompting them to pursue even more.

Implications for practitioners

We offered the following suggestions for educators interested in incorporating computing activities into their existing curriculum. First, what differentiates these lessons utilized in this study with others is that it provided teachers with a manageable amount of materials they may feel confident in implementing. These lessons were intended to supplant bits and pieces of existing curricula instead of supplement full lessons, which often cause more work for the teacher. In order to provide additional scaffolding and lessen the workload of the classroom teachers who taught these lessons, in this study video tutorials were also made available to the students as supplemental materials. This suggestion can address some of the challenges that teachers have expressed regarding time limitation, workload, and inadequate resources (Rich et al., 2019; Tran, 2019).

When considering the integration of computing activities, we recommend teachers to attend to content alignment within their existing curriculum. While extant literature suggests that CT can be integrated in a wide variety of content areas (i.e., math, science, social studies, language arts) (Barr & Stephenson, 2011; Berland & Wilensky, 2015; Jenkins, 2015; Shute et al., 2017), it may be more feasible to begin the integration with a subject area where the relevancy is more easily perceived by both teachers and students, such as math. Our findings also suggested that computing activities may be applicable to a wide array of mathematical concepts that can carry over across multiple grade levels (Gim, 2021; Niemelä et al., 2017; Niemelä & Helevirta, 2017).

We also recommend that it might be worthy of introducing a variety of programming environments simultaneously, including purely text-based environments that may be pre-conceived as intimidating by teachers. As our findings suggested, the majority of partici-

pants in this study were open and excited about the opportunity to code in a text-based environment. Despite the challenges imposed by the programming script, working in the text-based programming environment made learners feel like actual programmers, which largely enhanced their motivation to learn and practice in such environment. As most programming courses are limited to one language, by utilizing a combination of Scratch, Python, and Racket Programming Languages, students will be exposed to multiple programming environments and syntax (Niemelä & Helevirta, 2017), which may help them to view computer science in a less daunting way. Additionally, Racket, in combination with the Wescheme Environment, offers a math-friendly experience for both students and teachers (Schanzer, 2015).

Limitations and Future Research Recommendations

Several limitations exist within this research. This research study utilized a pre- and post-design as splitting students into two groups was not an option at the time of the study. Having both control and experimental groups would have been more effective when determining the effect of this coding-integrated instruction. One threat to internal validity is that some of the changes among student motivation may have been due to the novelty effect (Leedy & Ormrod, 2016) involving their excitement towards a new educational technology and not necessarily the coding instruction. We are aware that while the pre-post test design sheds light on the impact of the CT instruction, it did not directly measure the impact of the instruction and therefore failed to speak to the magnitude of the outcome as well as whether the outcomes are due to the instruction or alternative factors. Other factors such as prior knowledge or experiences of computer knowledge may have an influence on the increase of mathematical and computational knowledge acquisition and motivation that were not accounted for in the study. The responses from the participants may be subject to social-desirability bias, especially since the instructor also undertook the researcher role conducting the interviews. Inter-rater reliability could have been calculated to improve the reliability of qualitative data analysis. The sample size was small, as only 51 students from a Boys and Girls Club participated in this study. Additionally, the duration of the study was relatively brief with only nine one-hour sessions; greater insights would have been gained if the course were longer.

Given these limitations, we recommend the following future research directions. First, future researchers could employ a true or quasi-experimental design to directly examine the impact of these computing activities. Additional variables, such as differences in grade-level, prior knowledge of coding as well as computer science concepts, and gender can be further factored in to determine how these variables may influence student learning and motivation. Future studies using a larger sample drawn from a traditional classroom setting will improve the generalizability of the study. Future researchers may also specifically delve into the differences of the three programming languages to provide further insights into the unique attributes of each. Additionally, to evaluate how may the coding instruction influence student learning of math, further research is needed to examine the relationship between students' standardized exam results in math and the coding-infused instruction. Future research would be advantageous to further shed light on the role of computer programming in the directions specified.

Funding This study was not funded by any agency.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Compliance with Ethical Standards This research project has received IRB approval from Old Dominion University.

References

- State of Computer Science Education (2021). Retrieved from <https://advocacy.code.org/>
- Akinola, S. O. (2015). Computer programming skill and gender difference: An empirical study. *American Journal of Scientific and Industrial Research*, 7(1), 1–9. <https://doi.org/10.5251/ajsir.2016.7.1.1.9>
- Angeli, C., Voogt, J., Fluck, A., Webb, M., Cox, M., Malyn-Smith, J., & Zagami, J. (2016). A K-6 computational thinking curriculum framework: Implications for teacher knowledge. *Journal of Educational Technology & Society*, 19(3), 47–57
- Armoni, M. (2012). Teaching CS in kindergarten: How early can the pipeline begin? *ACM Inroads*, 3(4), 18–19. <https://doi.org/10.1145/2381083.2381091>
- Atmatzidou, S., & Demetriadis, S. (2016). Advancing students' computational thinking skills through educational robotics: A study on age and gender relevant differences. *Robotics and Autonomous Systems*, 75, 661–670. <https://doi.org/10.1016/j.robot.2015.10.008>
- 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, 48–54. <https://doi.org/10.1145/1929887.1929905>
- Basawapatna, A. R., Koh, K. H., & Reppenning, A. (2010, June). Using scalable game design to teach computer science from middle school to graduate school. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education* (pp. 224–228). ACM. <https://doi.org/10.1145/1822090.1822154>
- Belanger, C., Christenson, H., & Lopac, K. (2018). *Confidence and common challenges: The effects of teaching computational thinking to students ages 10–16* [Master's thesis, St. Catherine University]. SOPHIA Repository. <https://sophia.stkate.edu/maed/267>
- Benton, L., Hoyles, C., Kalas, I., & Noss, R. (2017). Bridging primary programming and mathematics: Some findings of design research in England. *Digital Experiences in Mathematics Education*, 3(2), 115–138. <https://doi.org/10.1007/s40751-017-0028-x>
- Berland, M., & Wilensky, U. (2015). Comparing virtual and physical robotics environments for supporting complex systems and computational thinking. *Journal of Science Education and Technology*, 24(5), 628–647. <https://doi.org/10.1007/s10956-015-9552-x>
- Bers, M., & Horn, M. (2010). Tangible programming in early childhood: Revisiting developmental assumptions through new technologies. In I. Berson, & M. Berson (Eds.), *High-tech tots: Childhood in a digital world* (pp. 49–70). Information Age Publishing
- Bubnó, K., & Takács, V. L. (2019). Cognitive aspects of mathematics-aided computer science teaching. *Acta Polytechnica Hungarica*, 16(6), 73–93. http://acta.uni-obuda.hu/Bubno_Takacs_93.pdf
- Burke, Q. (2016). Mind the metaphor: Charting the rhetoric about introductory programming in K-12 schools. *On the Horizon*, 24(3), 210–220. <https://doi.org/10.1108/OTH-03-2016-0010>
- Burnard, P. (1991). A method of analysing interview transcripts in qualitative research. *Nurse Education Today*, 11(6), 461–466. [https://doi.org/10.1016/0260-6917\(91\)90009-Y](https://doi.org/10.1016/0260-6917(91)90009-Y)
- Caglar, F., Shekhar, S., Gokhale, A., Basu, S., Rafi, T., Kinnebrew, J., & Biswas, G. (2018). Simulation modelling practice and theory cloudhosted simulation-as-a-service for high school STEM education. *Simulation Modelling Practice and Theory*, 58(2015), 255–273. <https://doi.org/10.1016/j.simpat.2015.06.006>
- Calder, N. (2010). Using Scratch: An integrated problem-solving approach to mathematical thinking. *Australian Primary Mathematics Classroom*, 15(4), 9–14. <https://doi.org/10.1007/s10857-012-9226-z>
- Clements, D. H. (2002). Computers in early childhood mathematics. *Contemporary Issues in Early Childhood*, 3(2), 160–181
- Clements, D. H., Battista, M. T., & Sarama, J. (2001). *Logo and geometry*. National Council of Teachers of Mathematics. <https://doi.org/10.2307/749924>

- Coşar, M., & Özdemir, S. (2020). The effects of computer programming on elementary school students' academic achievement and attitudes towards computer. *Elementary Education Online*, 19(3), 1509–1522. <https://doi.org/10.17051/ilkonline.2020.732794>
- Creswell, J. W., & Clark, V. L. P. (2017). *Designing and conducting mixed methods research*. Sage publications.
- Denning, P. J. (2017). Remaining trouble spots with computational thinking. *Communications of the ACM*, 60(6), 33–39. <https://doi.org/10.1145/2998438>
- Felleisen, M., & Krishnamurthi, S. (2009). Viewpoint: Why computer science doesn't matter. *Communication of the ACM*, 52(7), 37–40. <https://doi.org/10.1145/1538788.1538803>
- Fisler, K., Schanzer, E., Weimar, S., Fetter, A., Renninger, K. A., Krishnamurthi, S. ... Koerner, C. (2021, March). Evolving a K-12 curriculum for integrating computer science into mathematics. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (pp. 59–65). Association for Computing Machinery. <https://doi.org/10.1145/3408877.3432546>
- Flannery, L. P., Silverman, B., Kazakoff, E. R., Bers, M. U., Bontá, P., & Resnick, M. (2013). Designing ScratchJr: Support for early childhood learning through computer programming. In *Proceedings of the 12th International Conference on Interaction Design and Children* (pp. 1–10). ACM. <https://doi.org/10.1145/2485760.2485785>
- Fluck, A., Webb, M., Cox, M., Angeli, C., Malyn-Smith, J., Voogt, J., & Zagami, J. (2016). Arguing for computer science in the school curriculum. *Educational Technology and Society*, 19(3), 38–46
- Garneli, V., & Giannakos, M. N. (2015). Computing education in K-12 schools: A review of the literature. In *Proceedings of 2015 IEEE Global Engineering Education Conference (EDUCON)*, p. 543–551. <https://doi.org/10.1109/EDUCON.2015.7096023>
- Gim, N. G. (2021). Development of life skills program for primary school students: Focus on entry programming. *Computers*, 10(5), 1–17. <https://doi.org/10.3390/computers10050056>
- Google Inc. & Gallup Inc (2016). *Trends in the state of computer science in U.S. K-12 schools*. [http://google.j291E0](http://google/j291E0)
- Grover, S., & Pea, R. (2013). Using a discourse-intensive pedagogy and android's app inventor for introducing computational concepts to middle school students. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (pp. 723–728). ACM. <https://doi.org/10.1145/2445196.2445404>
- Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education*, 25(2), 199–237. <https://doi.org/10.1080/08993408.2015.1033142>
- Gutierrez, F. J., Simmonds, J., Hitschfeld, N., Casanova, C., Sotomayor, C., & Peña-Araya, V. (2018). Assessing software development skills among K-6 learners in a project-based workshop with Scratch. *Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training* (pp. 98–107). IEEE Xplore
- Harel, I., & Papert, S. (1990). Software design as a learning environment. *Interactive Learning Environments*, 1(1), 1–32. <https://doi.org/10.1080/1049482900010102>
- Hickmott, D., Prieto-Rodriguez, E., & Holmes, K. (2018). A scoping review of studies on computational thinking in K–12 mathematics classrooms. *Digital Experiences in Mathematics Education*, 4(1), 48–69. <https://doi.org/10.1007/s40751-017-0038-8>
- Hsu, T. C., Chang, S. C., & Hung, Y. T. (2018). How to learn and how to teach computational thinking: Suggestions based on a review of the literature. *Computers & Education*, 126, 296–310. <https://doi.org/10.1016/j.compedu.2018.07.004>
- Hughes, J., Gadanidis, G., & Yiu, C. (2017). Digital making in elementary mathematics education. *Digital Experiences in Mathematics Education*, 3(2), 139–153. <https://doi.org/10.1007/s40751-016-0020-x>
- Jenkins, C. (2015). A work in progress paper: Evaluating a microworlds-based learning approach for developing literacy and computational thinking in cross-curricular contexts. *Proceedings of the Workshop in Primary and Secondary Computing Education* (pp. 61–64). ACM. <https://doi.org/10.1145/2818314.2818316>
- Kumar, D. (2014). Digital playgrounds for early computing education. *ACM Inroads*, 5(1), 20–21. <https://doi.org/10.1145/2568195.2568200>
- Lakanen, A. J., & Kärkkäinen, T. (2019). Identifying pathways to computer science: The long-term impact of short-term game programming outreach interventions. *ACM Transactions on Computing Education (TOCE)*, 19(3), 1–30. <https://doi.org/10.1145/3283070>
- Lambert, L., & Guiffre, H. (2009). Computer science outreach in an elementary school. *Journal of Computing Sciences in Colleges*, 24(3), 118–124
- Lambić, D., Đorić, B., & Ivakić, S. (2020). Investigating the effect of the use of code.org on younger elementary school students' attitudes towards programming. *Behaviour and Information Technology*. Advance online publication. <https://doi.org/10.1080/0144929X.2020.1781931>

- Lee, Y., & Cho, J. (2019). Quantifying the effects of programming education on students' knowledge representation and perception in computational thinking. *International Journal of Innovation, Creativity and Change*, 9(4), 27–38
- Leedy, P. D., & Ormrod, J. E. (2016). *Practical research: Planning and design*. Pearson
- Lewis, C. M. (2010). How programming environment shapes perception, learning and goals: Logo vs. Scratch. *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (pp. 346–350). ACM. <https://doi.org/10.1145/1734263.1734383>
- Lincoln, Y. S., & Guba, E. G. (1985). *Naturalistic inquiry*. Sage Publications.
- Lu, J. J., & Fletcher, G. H. (2009). Thinking about computational thinking. *Proceedings of Proceedings of the 40th ACM Technical Symposium on Computer Science Education* (pp. 260–264). ACM. <https://doi.org/10.1145/1508865.1508959>
- 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. <https://doi.org/10.1016/j.chb.2014.09.012>
- Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008). Programming by choice: Urban youth learning programming with Scratch. *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education* (pp. 367–371). ACM. <https://doi.org/10.1145/1352135.1352260>
- Maloney, J. H., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch programming language and environment. *ACM Transactions on Computing Education*, 10(4), 16. <https://doi.org/10.1145/1868358.1868363>
- Manches, A., & Plowman, L. (2017). Computing education in children's early years: A call for debate. *British Journal of Educational Technology*, 48(1), 191–201. <https://doi.org/10.1111/bjet.12355>
- Matero, I. M., Weng, C., Astatke, M., Hsia, C. H., & Fan, C. G. (2021). Effect of design-based learning on elementary students computational thinking skills in visual programming maker course. *Interactive Learning Environments*. Advance online publication. <https://doi.org/10.1080/10494820.2021.1938612>
- Meyer, D., & Batzner, A. (2016, November). Engaging computer science non-majors by teaching K-12 pupils programming: first experiences with a large-scale voluntary program. *Proceedings of the 16th Koli Calling International Conference on Computing Education Research* (pp. 174–175). ACM. <https://doi.org/10.1145/2999541.2999563>
- Mioduser, D., Levy, S., & Talis, V. (2009). Episodes to scripts to rules: Concrete abstractions in kindergarten children's explanations of a robot's behaviors. *International Journal of Technology and Design Education*, 19(1), 15–36. <https://doi.org/10.1007/s10798-007-9040-6>
- Mladenović, M., Žanko, Ž., & Aglič Čuvić, M. (2021). The impact of using program visualization techniques on learning basic programming concepts at the K–12 level. *Computer Applications in Engineering Education*, 29(1), 145–159. <https://doi.org/10.1002/cae.22315>
- Morelli, R., De Lanerolle, T., Lake, P., Limardo, N., Tamotsu, E., & Uche, C. (2011). Can android app inventor bring computational thinking to K-12. *Proceedings. 42nd ACM Technical Symposium on Computer Science Education (SIGCSE'11)* (pp. 1–6). ACM
- Mouza, C., Yadav, A., & Ottenbreit-Leftwich, A. (2018). Developing computationally literate teachers: Current perspectives and future directions for teacher preparation in computing education. *Journal of Technology and Teacher Education*, 26(3), 333–352
- Namukasa, I. K., Kotsopoulos, D., Floyd, L., Weber, J., Kafai, Y. B., Khan, S., et al. (2015). From computational thinking to computational participation: Towards achieving excellence through coding in elementary schools. In G. Gadanidis (Ed.), *Math + coding symposium*. Western University
- Neri, F. (2021). Teaching mathematics to computer scientists: Reflections and a case study. *SN Computer Science*, 2(2), <https://doi.org/10.1007/s42979-021-00461-7>
- Niemelä, P. S., & Helevirta, M. (2017). K-12 curriculum research: The chicken and the egg of math-aided ICT teaching. *International Journal of Modern Education and Computer Science*, 9(1), 1–14. <https://doi.org/10.5815/ijmecs.2017.01.01>
- Niemelä, P., Partanen, T., Harsu, M., Leppänen, L., & Ihantola, P. (2017). Computational thinking as an emergent learning trajectory of mathematics. *ACM International Conference Proceeding Series*, 70–79. <https://doi.org/10.1145/3141880.3141885>
- Noh, J., & Lee, J. (2020). Effects of robotics programming on the computational thinking and creativity of elementary school students. *Educational Technology Research and Development*, 68(1), 463–484. <https://doi.org/10.1007/s11423-019-09708-w>
- Papastergiou, M. (2009). Digital game-based learning in high-school computer science education: Impact on educational effectiveness and student motivation. *Computers and Education*, 52(1), 1–12. <https://doi.org/10.1016/j.compedu.2008.06.004>
- Patton, M. Q. (2002). *Qualitative research and evaluation methods* (3rd ed.). Sage Publications
- Papert, S., Watt, D., diSessa, A., & Weir, S. (1979). *Final report of the Brookline Logo Project: Project summary and data analysis (Logo Memo 53)*. MIT Logo Group

- Powers, J., & Azhar, M. (2020). Preparing teachers to engage students in computational thinking through an introductory robot design activity. *Journal of Computers in Mathematics and Science Teaching*, 39(1), 49–70
- Prottsman, K. (2014). Computer science for the elementary classroom. *ACM Inroads*, 5(4), 60–63
- Qualls, J. A., & Sherrell, L. B. (2010). Why computational thinking should be integrated into the curriculum. *Journal of Computing Sciences in Colleges*, 25(5), 66–71
- Razak, M. R. B., & Ismail, N. Z. B. (2018). Influence of mathematics in programming subjects. In *American Institute Physics Conference Proceedings, 1974*, Article 050011. <https://doi.org/10.1063/1.5041711>
- Relkin, E., de Ruiter, L. E., & Bers, M. U. (2021). Learning to code and the acquisition of computational thinking by young children. *Computers and Education*, 169, 104222. <https://doi.org/10.1016/j.compedu.2021.104222>
- Rich, P. J., Browning, S. F., Perkins, M., et al. (2019). Coding in K-8: International trends in teaching elementary/primary computing. *TechTrends*, 63, 311–329. <https://doi.org/10.1007/s11528-018-0295-4>
- Rich, P. J., & Hodges, C. (2017). *Emerging research, practice, and policy on Computational Thinking*. Springer. <https://doi.org/10.1007/978-3-319-52691-1>
- Rich, P. J., Leatham, K. R., & Wright, G. A. (2013). Convergent cognition. *Instructional Science*, 41(2), 431–453. <https://doi.org/10.1007/s11251-012-9240-7>
- Rich, K. M., Yadav, A., & Schwarz, C. V. (2019). Computational thinking, Mathematics, and Science: Elementary teachers' perspectives on integration. *Journal of Technology and Teacher Education*, 27(2), 165–205
- Rodríguez-Martínez, J. A., González-Calero, J. A., & Sáez-López, J. M. (2020). Computational thinking and mathematics using Scratch: an experiment with sixth-grade students. *Interactive Learning Environments*, 28(3), 316–327. <https://doi.org/10.1080/10494820.2019.1612448>
- Schanzer, E. T. (2015). *Algebraic functions, computer programming, and the challenge of transfer* (Doctoral dissertation). Retrieved from <http://nrs.harvard.edu/urn-3:HUL.InstRepos:16461037>
- Sadik, O., Ottenbreit-Leftwich, A., & Nadiruzzaman, H. (2017). Computational thinking conceptions and misconceptions: Progression of preservice teacher thinking during computer science lesson planning. In P. J. Rich, & C. Hodges (Eds.), *Computational Thinking: Research and Practice* (pp. 221–238). Springer. https://doi.org/10.1007/978-3-319-52691-1_14
- Sáez-López, J. M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using “Scratch” in five schools. *Computers & Education*, 97, 129–141. <https://doi.org/10.1016/j.compedu.2016.03.003>
- Saritepeci, M. (2020). Developing Computational Thinking Skills of High School Students: Design-Based Learning Activities and Programming Tasks. *The Asia-Pacific Education Researcher*, 29(1), 35–54. <https://doi.org/10.1007/s40299-019-00480-2>
- Scherer, R., Siddiq, F., & Sánchez Viveros, B. (2020). A meta-analysis of teaching and learning computer programming: Effective instructional approaches and conditions. *Computers in Human Behavior*, 109, 1–18. <https://doi.org/10.1016/j.chb.2020.106349>
- Seiter, L. (2015). Using solo to classify the programming responses of primary grade students. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (pp. 540–545). New York, NY, USA: ACM. <https://doi.org/10.1145/2676723.2677244>
- Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, 22, 142–158. <https://doi.org/10.1016/j.edurev.2017.09.003>
- Soboleva, E. V., Sabirova, E. G., Babieva, N. S., Sergeeva, M. G., & Torkunova, J. V. (2021). Formation of computational thinking skills using computer games in teaching mathematics. *Eurasia Journal of Mathematics, Science and Technology Education*, 17(10), Article em2012. <https://doi.org/10.29333/ejmste/11177>
- Staples, A., Pugach, M. C., & Himes, D. J. (2005). Rethinking the technology integration challenge: Cases from three urban elementary schools. *Journal of Research on Technology in Education*, 37(3), 285–311. <https://doi.org/10.1080/15391523.2005.10782438>
- Strawhacker, A., & Bers, M. A. (2019). What they learn when they learn coding: Investigating cognitive domains and computer programming knowledge in young children. *Educational Technology Research and Development*, 67, 541–575. <https://doi.org/10.1007/s11423-018-9622-x>
- Subhi, T. (1999). The impact of LOGO on gifted children's achievement and creativity. *Journal of Computer Assisted Learning*, 15(2), 98–108. <https://doi.org/10.1046/j.1365-2729.1999.152082.x>
- Tran, Y. (2019). Computational thinking equity in elementary classrooms: What third-grade students know and can do. *Journal of Educational Computing Research*, 57(1), 3–31. <https://doi.org/10.1177/0735633117743918>
- Vasconcelos, L., & Kim, C. (2020). Coding in scientific modeling lessons (CS-Model). *Educational Technology Research and Development*, 68, 1247–1273. <https://doi.org/10.1007/s11423-019-09724-w>

- Weintrop, D., & Wilensky, U. (2017). Comparing block-based and text-based programming in high school computer science classrooms. *ACM Transactions on Computing Education*, 18(1), 1–25. <https://doi.org/10.1145/3089799>
- Wiedermann, W., & von Eye, A. (2013). Robustness and power of the parametric t test and the nonparametric Wilcoxon test under non-independence of observations. *Psychological Test and Assessment Modeling*, 55(1), 39–61
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33–35
- Wright, G., Rich, P., & Lee, R. (2013). The influence of teaching programming on learning mathematics. Proceedings of *Society for Information Technology & Teacher Education International Conference* (pp. 4612–4615). New Orleans, Louisiana, United States: Association for the Advancement of Computing in Education. <https://www.learntechlib.org/primary/p/48851/>

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Tian Luo is an Associate Professor of Instructional Design and Technology at Old Dominion University. She currently serves as an associate Editor-in-Chief for *Journal of Information Technology Education: Research*.

Jilian Reynolds is a doctoral student from Old Dominion University. She is a former K-12 teacher.

Pauline S. Muljana is a PhD candidate in Instructional Design and Technology at Old Dominion University. She is a former instructional designer.