Cui, Z., Ng, O., & Jong, M. S.-Y. (2023). Integration of Computational Thinking with Mathematical Problem-based Learning: Insights on Affordances for Learning. *Educational Technology & Society*, *26*(2), 131-146. https://doi.org/10.30191/ETS.202304_26(2).0010

Integration of Computational Thinking with Mathematical Problem-based Learning: Insights on Affordances for Learning

Zhihao Cui[1], Oi-lam Ng[1*] and Morris Siu-Yung Jong[1,2]

[1]Department of Curriculum and Instruction, The Chinese University of Hong Kong, Hong Kong SAR, China // [2]Centre for Learning Sciences and Technologies, The Chinese University of Hong Kong, Hong Kong SAR, China // cuizhihao@link.cuhk.edu.hk // oilamn@cuhk.edu.hk // mjong@cuhk.edu.hk

[*]Corresponding author

ABSTRACT: Grounded in problem-based learning and with respect to four mathematics domains (arithmetic, random events and counting, number theory, and geometry), we designed a series of programming-based learning tasks for middle school students to co-develop computational thinking (CT) and corresponding mathematical thinking. Various CT concepts and practices articulating the designated mathematical problems were involved in the tasks. In addition to delineating the design of these learning tasks, this paper presents a qualitative study in which we examined 74 students' learning outcomes and characterized their CT and mathematical thinking co-development as they accomplished the tasks. The research results demonstrate the co-development of both mathematics- and CT-related concepts and practices in the four mathematics domains. Two types of interactions are identified: (i) applying mathematical knowledge to construct CT artifacts and (ii) generating new mathematical knowledge with CT practice. The new insights provided by the present work are threefold. First, from a mathematical learning perspective, the nature of the solution processes of the designed problems should not be immediately obvious. Second, from a technology-enhanced learning perspective, the dynamic representations and immediate visual feedback afforded by the programming tool are beneficial to student learning. Third, from a pedagogical perspective, the room for customization offered by both the designed problems and programming tools can provide affordances for learning.

Keywords: Computational thinking, Mathematics education, Problem-based learning, Problem solving, STEM

1. Introduction

Computational thinking (CT) can be regarded as a mode of problem solving and thinking with computational tools and as a fundamental skill required in daily life (Wing, 2006; Wing, 2011). In the current development of teaching and learning with computing, much emphasis has been placed on integration with other disciplines and fields (Guzdial & Soloway, 2003); this represents a shift away from focusing on computer science education in isolation. In light of science, technology, engineering, and mathematics (STEM) education has been a global educational focus today (Jong, Song, Soloway & Norris, 2021), CT is regarded as a kind of analytical thinking that shares close connections with all four involved disciplines (Leung, 2020), and especially with mathematics (Baldwin et al., 2013). The use of programming and the application of CT to learning mathematics can be traced back to Papert (1980), who argued that CT could have a unique effect on mathematical thinking and learning because it provides learners with a medium for exploring patterns and a logical structure for modeling and investigating mathematical relationships. More recently, a systematic illustration of the connection between mathematics and CT was proposed by Weintrop and colleagues (2016), who suggested that various CT practices, including data practices, modeling and simulation practices, computational problem-solving practices, and systems-thinking practices, can play a supportive role in mathematical practices and be mutually promoted. Recent reviews have revealed considerable literature growth around the integration of CT and mathematics in recent decades (e.g., Hickmott et al., 2018; Ye et al., 2023), arguing for the multi-faceted linkage of CT and K-12 mathematics education. Although these reviews evince the reciprocal relationship between CT and mathematical concepts, the question of how CT and mathematics can be co-developed remains underexplored (Nordby et al., 2022; Ye et al., 2023), as stated by Hickmott et al. (2018) "studies that explicitly linked the learning of mathematics concepts with computational thinking were uncommon in the reviewed literature" (p. 65). Recently, there have been studies exploring CT integration for learning in specific mathematical domains, such as combinatorics (De Chenne & Lockwood, 2022), number theory and mathematical modeling (Benton et al., 2018; Ng & Cui, 2021), and geometry (Ng & Cui, 2021; Pei et al., 2018), as well as others investigating challenges that emerge when engaging in mathematical problem solving within a programming environment (Cui & Ng, 2021; DeJarnette, 2019; Ng et al., 2021; Weng et al., 2022). However, as argued by Lockwood and De Chenne (2019), while programming seems to be effective in learning mathematics for certain topics, it cannot be concluded that it would be superior to paper-and-pencil methods in all mathematics domains. Therefore, further research is needed to understand the interplay between these two modes of thinking (i.e., mathematical and CT), especially in terms of shedding light on the affordances of simultaneously using two modes of thinking, as well as on the challenges students may experience when solving mathematical problems in programming contexts.

To this end, there is still a great deal of room to investigate the integration of CT with mathematics education, especially in K–12 contexts. The two most significant remaining questions in this regard are (i) how CT and mathematics learning outcomes can be co-developed (Ye et al., 2023) and (ii) connecting specific mathematical domains for integration with programming (Lockwood & Morken, 2021). In response to these research gaps, this study addresses the characteristics of CT-based mathematics instruction and student learning in such an environment. Our goals in this study are twofold. First, we illustrate the design elements of the CT-based mathematical tasks from four mathematical domains (i.e., arithmetic, random events and counting, number theory, and geometry) and highlight their impact on students' learning outcomes. Second, we are interested in identifying the affordances and barriers brought forward by problem-based mathematics learning in the block-based programming environment, Scratch. Specifically, we aim to address the following research questions (RQs):

- How is CT co-developed with problem-based mathematics learning in designed tasks in each of the following mathematical domains: arithmetic, random events and counting, number theory, and geometry?
- How might the design of CT-based mathematical activities provide affordances for student learning in each of these domains?

2. Conceptual framework

2.1. Computational thinking, concepts, and practices

In the past decade, researchers have made efforts to develop conceptual and methodological frameworks for learning and teaching CT (e.g., Brennan & Resnick, 2012; Ho et al., 2021; Jong et al., 2020; Román-González et al., 2017; So et al., 2020). Among them, Brennan and Resnick (2012) proposed one of the most popular frameworks in which CT can be addressed from three dimensions: computational concepts, computational practices, and computational perspectives. They identified seven computational concepts (sequences, loops, parallelism, events, conditionals, operators, and data), four sets of computational practices (incremental and iterative, testing and debugging, reusing and remixing, and abstracting and modularizing), and three kinds of computational perspectives on the world and oneself as a programmer (expressing, connecting, and questioning). In this study, we considered the three dimensions of CT proposed by Brennan and Resnick as the learning goals of CT.

Conversely, another group of researchers has explored the relationship between CT and thinking practices in other disciplines, such as mathematics. For example, Sneider et al. (2014) created a Venn diagram illustrating the overlap between mathematical thinking (MT) and CT, wherein the common area included problem solving, modeling, analyzing, and interpreting data, as well as skills in statistics and probability. They explained that outside the intersection of MT and CT, more distinct MT (e.g., counting and geometry) and CT (e.g., programming and data mining) practices are found. Weintrop et al. (2016) formulated a taxonomy integrating mathematics and CT into four categories: data practices, modeling and simulation practices, computational problem-solving practices, and systems-thinking practices. This mapping enabled them to produce framework statements that reflect how CT is applied—particularly in the context of mathematics and science—as a way to support integrated instruction that mutually enriches student learning in each discipline. The taxonomy was informed by the research finding that computational problem-solving practices, such as algorithm development and creating computational abstractions, can help learners develop a deep understanding of mathematical and scientific phenomena (e.g., Wilkerson-Jerde, 2014). We believe that this CT practice taxonomy could serve as a strategy for learning and problem solving, especially within the scope of integrating CT into problem-based mathematical learning.

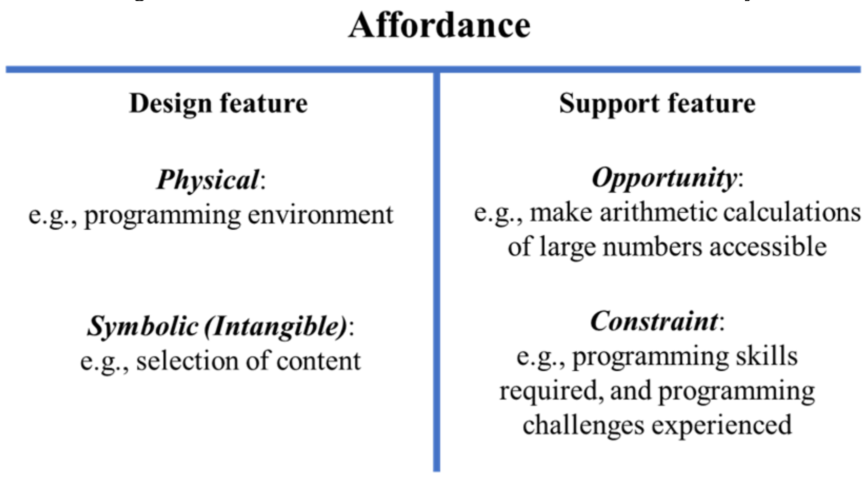
2.2. Affordance for learning

"Affordance" was introduced by Gibson (1979) to describe the relationships that exist between an object or environment and an organism. The subsidiary idea is that affordances provide both opportunities and constraints, which are not opposites but complementary. Norman (1999) proposed one of the most notable reformulations of the concept of affordance with respect to "real" affordance and "perceived" affordance, according to which real affordance refers to the physical characteristics of a device or interface that allow its operation, as described by

Gibson (1979), while perceived affordance can be defined as the apparent characteristics of a device that provide clues to its proper operation. Differing from Gibson (1979), Norman (1999) recognized that the object or environment could be both symbolically and physically designed and that the term "affordance" could be used for the purpose of design. Building on the work of Gibson (1979) and Norman (1999), Hartson (2003) further defined the term "affordance" as cognitive affordance (Norman's perceived affordance) and physical affordance (Norman's real affordance); in this scheme, affordance is a "design feature" that "aids, supports, facilitates, or enables thinking, knowing, and/or doing something" (p. 319). In the context of educational research, educational affordances are "those characteristics of an artifact (e.g., how a chosen educational paradigm is implemented) that determine if and how a particular learning behaviour could possibly be enacted within a given context (e.g., project team, distributed learning community)" (Kirschner, 2002, p. 14). In the mathematics education context, Bishop and colleagues (2014) provide another example of cognitive affordance, referring to individuals' understanding or knowledge that may lead to successful learning progress or problem solving within the targeted content.

Informed by Kirschner's educational affordance and Bishop's cognitive affordance in mathematics learning, we propose a framework for understanding the potential affordance of integrating CT with mathematics in the current study (Figure 1). First, we emphasize design features, which include physical and symbolic (or intangible) aspects (Norman, 1999). For example, in the context of the present study, the block-based programming environment could be treated as a physical design affordance for learning CT, because it was something with which the students could directly interact. Moreover, we identified the learning content (i.e., the four mathematical domains of arithmetic, random events and counting, number theory, and geometry) as an intangible design affordance for supporting learning behavior (Kirschner, 2002). For instance, certain geometrical content is suitable to accompany visual representations to support programming practices and thus to construct, explore, and verify the properties of geometric figures; this represents a case of the selection of learning content to afford students' CT-based mathematics learning. Second, we highlight the support feature of affordance in terms of providing both opportunities and constraints. This can be exemplified by the use of computing to make arithmetic calculations with ease (i.e., opportunity); however, the students must correctly program to perform the calculations (i.e., constraint). In summary, CT-based mathematics instruction may provide a unique affordance for student learning from the perspectives of design and support.

Figure 1. The two features of affordance in the current study


Affordance
Design feature
Physical: e.g., programming environment
Symbolic (Intangible): e.g., selection of content
Support feature
Opportunity: e.g., make arithmetic calculations of large numbers accessible
Constraint: e.g., programming skills required, and programming challenges experienced

3. Methodology

3.1. Research design, participants, and context

This study is situated in a series of programming-based teaching interventions that address various mathematical domains. It employed a design-based research (DBR) methodology consisting of three iterative cycles of implementations to achieve its aims. DBR is conducted "with the intent of producing new theories, artifacts and practices that account for and potentially impact learning and teaching in naturalistic settings" (Barab & Squire, 2004, p. 2). During the three cycles of implementation, the researchers designed and refined a total of eight programming-based mathematical tasks in partnership with mathematics schoolteachers and computer science experts.

A total of 74 participants (57 male and 17 female) ranging from fifth to eighth grade (ages 10 to 14) were recruited from various primary and secondary schools in Hong Kong and provided informed consent to participate in the study. According to their self-reported previous experience in programming before the study, some of the participants had experienced some very basic functions related to programming (such as motion control and simple conditions) in Scratch. They had never been engaged in using programming to solve mathematical problems or in learning more comprehensive CT concepts (e.g., variables and iteration) and practice (e.g., modeling and remixing). Hence, their prior knowledge was considered to have no significant influence on the learning outcomes of the current study.

3.2. Selected mathematical domains and tasks

Table 1 lists the mathematical domains and tasks developed and implemented in the study, labeled (1)–(8). Specifically, the first cycle of implementation employed tasks (1) and (6); the second cycle of implementation addressed tasks (1), (2), (3), and (8); and the third cycle of implementation involved tasks (4), (5), (7), and (8). All the tasks were designed with authentic contexts with real-life relevance and were open-ended in nature, which required knowledge from the respective mathematical domains to solve. In summary, the tasks can be categorized into four major mathematical domains (arithmetic, random events and counting, number theory, and geometry), thus allowing the research questions to be explored. Noting that most of the selected mathematical content (i.e., geometric sequences, probability, and fractal geometry) had not been introduced in formal lessons prior to the study, it can be inferred that both mathematical and CT concepts were developed by the students in the current study. More information about the tasks implemented in this study is provided in Appendix 1.

Table 1. Selected domains and tasks with corresponding mathematical and CT concepts

Domain	Task name	Mathematical concept involved	Expected product
Arithmetic	Two Savings problem (1)	Sequence and series	Numerical output and their visual representations
	Fibonacci Sequence (2)	Recursive sequence	
Random events and counting	Dice Rolling problem (3)	Random events with equally likely outcomes	Value output of variables/visual representation of distributions
	Dart Throwing problem (4)	Random events with unequally likely outcomes	
Number theory	Count to 21 or 100 problem (5)	Counting, inductive and deductive reasoning	A math game with inputs and computer auto-reactions
	Prime Detector (6)	Divisibility rules, factors and multiples	Text and/or list output
Geometry	Drawing Polygons (7)	Exterior and interior angles	Multiple polygons
	Drawing Fractals (8)	Fractal and recursion	Fractal geometry

3.3. Procedures

The three cycles of implementation employed a similar set of procedures. Participants attended three to five programming sessions involving various mathematical problem-based learning in increasing order of complexity. Each session took approximately two hours. In the first part of each session, the course instructor conducted whole-class instruction with the goal of scaffolding essential prerequisite mathematical and programming knowledge for solving the target problems in the respective sessions. Afterward, the students would follow demonstrations led by the course instructor, answer questions posed by the instructor, and complete some guided activities. The remainder of each session (around one hour) was devoted to students' individual and collaborative problem solving, in which teaching assistants, with a teacher–student ratio of roughly 1:6, provided the necessary assistance. Participants were encouraged to communicate with peers about their thoughts and plans to solve the problem, while this process was video-recorded. After each session, participants wrote reflections on the tasks; this included critically discussing what they had learned and the challenges they had met during the session. By the end of the implementation, selected students were invited to participate in semi-structured post-course interviews. The semi-structured questions included: What was something new you learned in the course? Were there any challenges or difficulties that you met, and how did you overcome them? How did you come up with ideas for solving the problem?

3.4. Data analysis

We adopted a case study as the analytic methodology. Case study complements in-depth analyses of learning "given the need for extended, open, and careful consideration of data" (Parnafes & diSessa, 2013, p. 7). It takes into account the intriguing parts and significant components of the subject, which is suitable for answering research questions such as those proposed in the current study. During the DBR research, we collected data, including programming artifacts, classroom observation notes, video recordings, field notes, and student interviews. The researchers first reviewed all the artifacts constructed by the students, as well as the video and audio recordings of the class. Then, by combining these with the classroom observation field notes, the researchers selected episodes and artifacts that characterized the students' cognitive development of both CT and mathematics in each mathematical domain. For the selected episodes, we employed a constant comparative strategy (Corbin & Strauss, 2015) to narrow down the selection of episodes and artifacts so that they demonstrated and characterized the students' learning outcomes from both mathematical and CT perspectives. The student interviews served as supplemental evidence for triangulating the results. With regard to the nature of affordance according to the proposed framework, we identified how the present instructional design could provide affordances for the co-development of CT and mathematics.

To examine students' CT development in this study, we adopted two influential frameworks—those of Brennan and Resnick (2012) and Weintrop et al. (2016)—to analyze the students' CT development as encompassing a set of CT concepts and practices (Table 2). The shortened list of CT concepts and practices served as the coding criteria for demonstrating students' development of CT when reviewing the data. For example, the CT concepts in use could be identified by the programming codes used by the students, such as "if … then" and "repeat" with respect to the CT concepts of conditionals and loops, respectively. For CT practices, we referred to the students' programming processes over a period of time in terms of what kinds of subtasks they were tackling within the CT environment, e.g., modeling, testing, and debugging (Weintrop et al., 2016) or reusing and remixing (Brennan & Resnick, 2012).

Table 2. CT concepts and practices involved in the current study

CT concepts	Description
Loop	A mechanism for running the same sequence multiple times
Sequence	A particular activity or task expressed as a series of individual steps or instructions that can be executed by the computer
Condition	Make decisions based on certain conditions, which supports the expression of multiple outcomes
Iteration	The outcome of each iteration is the starting point of the next iteration
Variable	Value that contains some known or unknown quantity
Subroutine	A complete executable packaged program instruction that can be used in other programs at any time
Boolean logic	A form of algebra in which all values are either True or False. These values are used to test the conditions.
CT practices	**Description**
Modelling and simulation	Using computational models to understand a concept, to find and test solutions; assessing, designing, and constructing computational models
Algorithmic thinking	A series of steps to solve a problem
Reusing and remixing	Building on others' work (i.e., ideas and code) to create things that are much more complex
Testing and debugging	Developing strategies (e.g., by trial and error) to deal with and anticipate problems

To investigate students' mathematical development, we designed tasks targeting certain mathematical concepts in a particular domain, as illustrated in Table 1. The list served as the coding criteria to select artifacts and extract episodes to provide evidence for students' mathematical development from the data. For example, the concept of random events of equal likelihood can be identified by how they inductively infer the law of large numbers from simulating dice rolls in the reflection questions. In addition, we consistently employed transcripts to analyze the students' discourse while engaging with the mathematical problems in the programming environment to triangulate data from different sources and ensure the credibility of the qualitative results.

4. Results

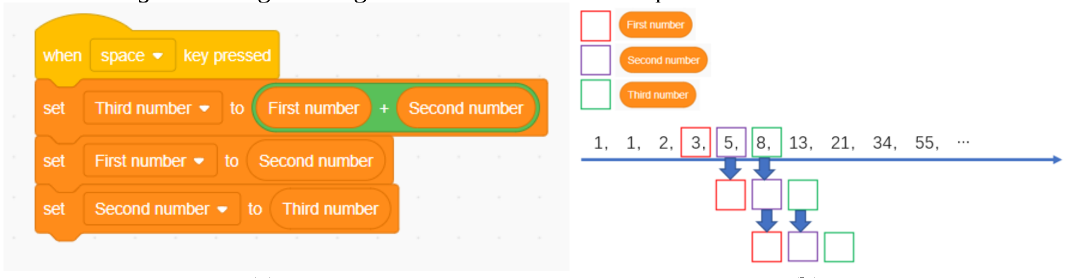
The following subsections present the results with respect to the designed tasks in four mathematical domains—(i) arithmetic, (ii) random events and counting, (iii) number theory, and (iv) geometry—with representative artifacts and episodes to detail, provide evidence for, and situate the students' development in both mathematics and CT. In each domain, to respond to RQ1, we first demonstrate how each type of mathematical content was co-developed with CT concepts and practices during the designed CT-based mathematics activities (intangible affordance). Then, in response to RQ2, we explain the importance of the design feature of the tasks that provided affordances for students' knowledge or skill acquisition.

4.1. Arithmetic

4.1.1. Co-development of CT and mathematics

Arithmetic thinking was co-developed with the CT concepts of variables and iterative operations. As the first problems tackled by the students, the CT-based arithmetic tasks were meant for the students to begin translating their arithmetic procedures, such as computing 3 + 222 + 222 = 447 and 3 + 6 + 12 = 21 in the Two Savings problem, and finding the next term in a Fibonacci sequence by adding the previous two terms using the Scratch programming language. Given the programming environment's ability to take care of the arithmetic procedures effortlessly, the respective problems stimulated the students to use effective strategies to ensure that their programs displayed the correct sequence. For this, the use of variables was called for, where (i) a variable was something that took on different values, and (ii) variables could be operated iteratively by using codes such as "set balance to balance + 222." In other words, the students linked their mathematical thinking, which involved searching for patterns and determining the next term in the sequence, with variables in a CT sense, knowing that as long as something changed, they could use a variable to represent this changing quantity. Moreover, the use of variables was complemented with iterative operations by using the [repeat] code in Scratch, which enabled a quantity to change by the repeated use of a certain rule. As shown in Figure 2a, a typical solution to the Fibonacci Sequence problem was to operate three variables by setting "the third number" equal to the sum of the "first number" and "second number." In mathematical language, this was precisely $T_n = T_{n-1} + T_{n-2}$. The concept of iterations came into play in coding the next few lines because the "second number" and "third number" become the new "first number" and "second number," respectively, generating an iteration process, as illustrated in Figure 2b. Hence, we regard the students' arithmetic thinking as co-developing with the CT concepts of variables and iterative operation.

Figure 2. Programming codes of the Fibonacci Sequence and the iteration idea



```
when space key pressed
set Third number to First number + Second number
set First number to Second number
set Second number to Third number
```

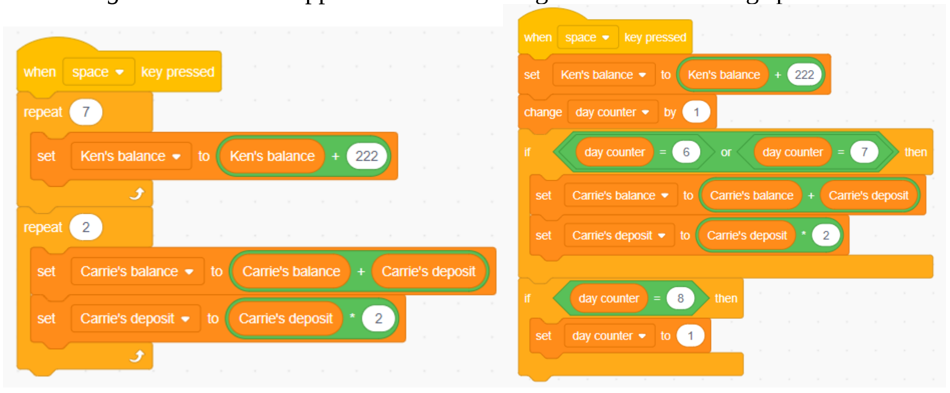
(a) (b)

4.1.2. Affordance from the design feature of the tasks

The same two CT-based mathematical activities afforded the students the opportunity to visualize and simulate the arithmetic (e.g., 3, 225, 447, …) and geometric progressions (e.g., 3, 6, 12, …) posed in the problems. Compared to a paper-and-pencil environment, which would likely have prompted the students to use a static formula, such as $3 + 222(n - 1)$ to describe the n^{th} term of the sequence, the students in this study used the programming environment to simulate each term dynamically, using the codes mentioned above to visualize the growth of various sequences one term at a time. Furthermore, some students used visual representations to show the amount of growth from one term to the next, which can be significant in improving their understanding of the differences between arithmetic and geometric sequences. On the other hand, the programming techniques required were considered constraints in solving the problems. As one student commented, "The numbers are too big, so it's nearly impossible for a human brain to do it, but I don't know how a computer thinks in this program. It took me three days to complete that task." This suggests that, although the student recognized the affordances of computing in dealing with large numbers, he struggled with solving the problem in a computational context due to being unfamiliar with the programming tools.

Regarding the Two Savings problem, it was observed that the design of the problem provided affordances that supported the students in modeling a real-life scenario that involved arithmetic operations. After successfully creating a program that simulated the two saving plans, a final challenge remained: namely, modeling the situation in which a deposit was to be made every day for the first saving plan, as opposed to making deposits only on weekends for the second saving plan. With this type of problem design, we observed that the students used various nonroutine strategies to model the situation successfully. For example, some students used different keys to denote different parameters, such as using the "D" key to denote the passing of weekdays and "W" for the passing of weekends. Other strategies included (1) using seven days or a week as a unit, that is, within each week, repeating the deposit seven times for the first plan and two times for the second plan (Figure 3a), and (2) creating a new variable (i.e., day counter) to serve as a hint regarding the day of the week, and then operating the deposit accordingly (Figure 3b). This indicated that students experienced and developed the skills of modeling and simulation in computational practice.

Figure 3. Different approaches to modeling for the Two Savings problem



```
when space key pressed
repeat 7
    set Ken's balance to Ken's balance + 222
repeat 2
    set Carrie's balance to Carrie's balance + Carrie's deposit
    set Carrie's deposit to Carrie's deposit * 2
```

```
when space key pressed
set Ken's balance to Ken's balance + 222
change day counter by 1
if day counter = 6 or day counter = 7 then
    set Carrie's balance to Carrie's balance + Carrie's deposit
    set Carrie's deposit to Carrie's deposit * 2
if day counter = 8 then
    set day counter to 1
```

(a) (b)

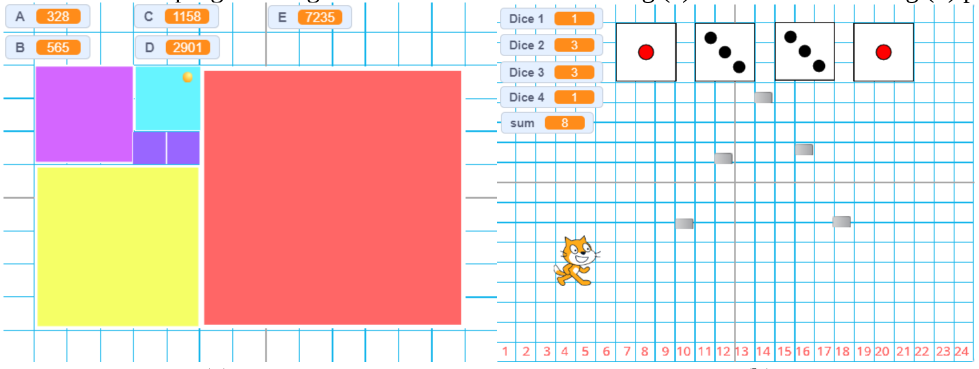
4.2. Random events and counting

4.2.1. Co-development of CT and mathematics

The concept of randomness, as appropriated by the randomize function in Scratch, was codeveloped in the students' probabilistic thinking in mathematics. In the first stage of the Dice Rolling problem, the students were guided to simulate the situation of rolling six dice at once by using the randomize function and calculating their sums. As shown in the online chat record, the students initially held certain common-sense expectations regarding the concept of randomness: "I found that Scratch's random is fake … it has a pattern." This comment was in agreement with other students' observations: "I got 22 five times (in 20 clicks)," "21 never happened for me," and "I got three 28s in a row (20 clicks)." These expressions suggest that the students had wrongly related the computer-generated randomized results to their expectations of a uniform distribution within only a small number of trials. In other words, they thought that when the dice were rolled randomly, a certain number should not appear at such a high frequency (i.e., five instances of 22 in 20 rolls), or a certain number should have appeared (i.e., 21 never happened but "1 + 2 + 3 + 4 + 5 + 6 = 21"). These conversations suggested that the students were rethinking the meaning of "randomness," both in a mathematical and computational sense: as they obtained more and more trials with the help of loops in Scratch, the students began to see that the observed frequency would mirror the expected distribution when performing random events.

At the same time, the students' concept of experimental probability was found to co-develop with the CT practice of simulation. In both the Dice Rolling problem and the Dart Throwing problem, the students were encouraged to simulate the process a large number of times to observe the distribution of the results. As illustrated in Figure 4a, in order to design a fair scoring system, one student ran the dart-throwing simulation 20,000 times to find the frequency distribution with which the dart hit the squares. As such, he proposed a scoring system that incorporated his observed frequency distribution. The square that was hit most frequently should be scored the lowest, and so on. Furthermore, using the data obtained from 20,000 simulations, the student designed a scoring system (A = 22, B = 12, C = 6, D = 2.5, E = 1) according to the ratio of the number of times each square was hit (i.e., the scoring system should be inversely proportional to the ratios). Another student simulated the dart-throwing situation 1,000 times, and by observing the experimental outcomes of the darts' landing points with the area of the dartboard, he inferred that the two quantities were proportional. A similar observation was also found in the Dice Rolling problem. By visualizing the outcomes of the targeted sums in Scratch (Figure 4b), one student discovered that the outcomes were expected to be symmetrical around the median when obtaining the four-dice sum. These examples suggest that concepts of experimental probability co-emerged with the CT practice of simulation.

Figure 4. Two students' programming artifacts for the Dart Throwing (a) and the Dice Rolling (b) problems



(a) (b)

4.2.2. Affordance from the design feature of the tasks

The most significant affordance provided by the tasks in this domain is the opportunity for students to simulate a large number of random events, which would be nearly impossible to do when performed manually. The combination of experimental probability with the programming environment was an instrumental affordance allowing students to develop their probabilistic thinking alongside computational concepts and practices. For example, the students' discourse about their expected dice rolling results reflected that they had made sense of what the frequency distribution would look like. With the ability to process a large number of trials by programming, the students tested their hypothesis, which filled the gap between experimental and theoretical probability—which is one of the main challenges for learning probability. The second stage of the Dice Rolling problem required students to generate the outcome space (e.g., 1-1-1-1-1-1, 1-1-1-1-1-2, … 6-6-6-6-6-6), as well as the frequency of the sums. We note that students who successfully programmed to find the outcome space inferred that the obtained sums would be symmetrical around the median sum (i.e., 21 in a six-dice situation) and that the median sum would appear with the greatest frequency. With the comparison of the counting results and the experimental simulation over a large number of trials, the students found the relationship between theoretical and experimental probability, which was co-developed with the CT concept and practice of loops and simulations.

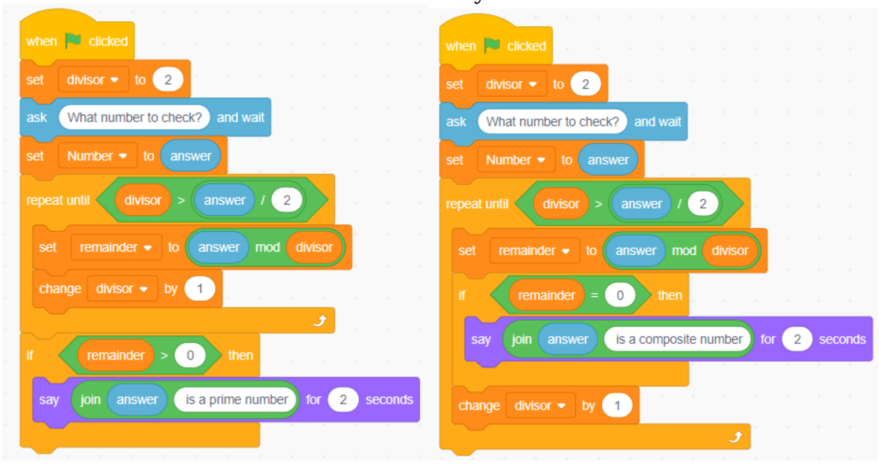
4.3. Number theory

4.3.1. Co-development of CT and mathematics

The mathematical concepts of divisibility, factors, and multiples were co-developed with the CT concept of conditions and Boolean logic. In the Prime Detector problem, the students used the code "mod" to determine whether a number was divisible by another number (i.e., A mod B results in the remainder of A divided by B). Starting from the definition of a prime as a whole number greater than 1 and divisible only by the number 1 and itself, a typical model in computational language was "check the mod of the target number (N) repeatedly from 2 to N – 1; if all the mod results are non-zero, then the target number is a prime," as illustrated in Figure 5a. The codes used by some students seemed to meet the logic of "checking all the divisors starting from 2 and then using the condition 'if' to determine whether all the remainders were greater than zero." However, this was incorrect because the computer did not store the results within the loop. The condition "if" only checked the result of the last divisor. We observed that the students who experienced this unsuccessful attempt turned to the alternative model of "checking for a composite number" (Figure 5b). In other words, whenever the remainder yielded a result of zero, this indicated that the number had a factor other than 1 or itself and was therefore a composite number. Here, the students needed to combine Boolean logic, which returned a value of true or false and the conditional statement of "if … then …" to conclude that a number was a prime "if all remainders were non-zero."

We note that the question of when to stop the loop came into play for some students, who questioned how they might optimize the condition in the program to make it “more efficient.” The students knew that based on the definition of a prime number, one could perform divisions from N/2 to N/(N – 1) to check for factors within the interval [2, N – 1]. However, some students found that if they changed the stopping condition to “repeat until the divisor > target number/2” (Figure 5b), the results would remain the same. We suggest that this particular meaning of primes was situated in the students’ CT practice in that the students were made aware of how to think like a computer as well as of ways to make the program more efficient (i.e., by taking half of the calculation time).

Figure 5. Two students’ programming artifacts illustrating how to set the condition (a) incorrectly and (b) correctly



```
when 🏁 clicked
set divisor ▾ to 2
ask What number to check? and wait
set Number ▾ to answer
repeat until divisor > answer / 2
    set remainder ▾ to answer mod divisor
    change divisor ▾ by 1
if remainder > 0 then
    say join answer is a prime number for 2 seconds
```

```
when 🏁 clicked
set divisor ▾ to 2
ask What number to check? and wait
set Number ▾ to answer
repeat until divisor > answer / 2
    set remainder ▾ to answer mod divisor
    if remainder = 0 then
        say join answer is a composite number for 2 seconds
    change divisor ▾ by 1
```

(a) (b)

4.3.2. Affordance from the design feature of the tasks

The task provided an affordance supporting students in systematically testing and debugging their programs. For example, the student who programmed the codes in Figure 5a claimed that his program worked because he had tested the numbers 14, 16, and 110, and the program had returned the result of a composite number. Meanwhile, when he tested the numbers 5, 7, and 11, the program returned the result of a prime number. Consequently, he incorrectly claimed that the number 7,081 was a prime number because he had failed to test the program with an odd composite number, such as 15, which the program would incorrectly detect as a prime. This phenomenon raised the question of how to test the program effectively. Unlike the Two Savings problem, in which the students could compare their program output with their hand calculations to test whether the program had worked as desired, the prime detector problem was more sophisticated in that it prompted the students to decide what numbers to use to test their program. Without adequate consideration of the properties of primes and composites, their choice of testing numbers potentially limited their judgment of the correctness of the program. Therefore, the task was considered meaningful for developing the CT practice of testing and debugging in tandem with number properties.

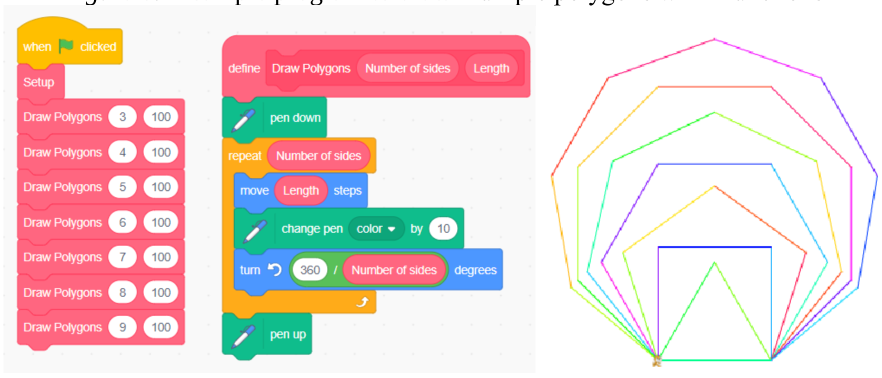
4.4. Geometry

4.4.1. Co-development of CT and mathematics

We observed the co-development of the students’ geometrical thinking along with the CT concept of sequence, as well as the use of parameters. The mechanism of drawing in the Scratch program requires making a path for the Pen tool to move along. In this case, the programming sequence becomes critical because it differs from spontaneous drawing with paper and pencil. For example, drawing a line and then coming back to the starting point could be easily done by hand, but it would require the sequence of “move X steps – turn 180 degrees – move X steps” in Scratch, and the direction of the Pen would remain reversed. In the problem of Drawing Polygons, students were first guided to draw a triangle. Initially, some students encountered difficulties due to confusing the Pen tool’s movement with the turning angle in Scratch. Using trial and error, they then successfully determined how to turn properly and went on to draw various (regular) polygons. In doing so, they also inferred that the sum of the exterior angle of any polygon would be 360 degrees (as reflected in the code “turn [360/number of sides] degrees” in Figure 6a) because the Pen tool would have rotated exactly one round after drawing the polygon. The use of the parameter "number of sides" was especially pertinent in helping the students observe this relationship, which demonstrated that the students' geometrical thinking was closely supported by the CT environment.

The mathematical concept of the limit was co-developed with the CT concept of subroutines and the CT practices of reusing and remixing as the students continued to advance their usage of parameters. Unlike other tasks in the course in which the output was in numerical form rather than strictly visual, the students needed to think about the size and aesthetics of their drawings. As such, the students learned to use subroutines to duplicate drawings with varied sizes or customizable features using parameters. In drawing multiple polygons, such as the ones in Figure 6a, the students learned to create a function with two parameters (i.e., the number of sides and the lengths of sides). The "draw polygons" function now served as a subroutine that students could reuse repeatedly, which was a significantly different experience from drawing with paper and pencil. As one student commented regarding using subroutines: "The main code will be shorter and more efficient, and if you wanted to change some parts of the codes, you would only have to change it one time." Meanwhile, in the process of drawing polygons with varying numbers of sides (Figure 6b), one student asked, "If I draw a 360-sided polygon, will I get a circle?" This comment was derived from his observation that a polygon with many sides resembles the shape of a circle, so he set 360 as the parameter for the number of sides, which is a relatively large number of sides, making each turn angle 1 degree (very small). Regardless of what he believed to be the limit value, his exploratory thinking could be considered a limiting process that supported the construction of the limit concept. Importantly, this was uniquely contextualized in the CT environment, particularly with subroutines and the practices of reusing and remixing.

Figure 6. A sample program to draw multiple polygons with "functions"



```
when ⚑ clicked
Setup
Draw Polygons 3 100
Draw Polygons 4 100
Draw Polygons 5 100
Draw Polygons 6 100
Draw Polygons 7 100
Draw Polygons 8 100
Draw Polygons 9 100
```

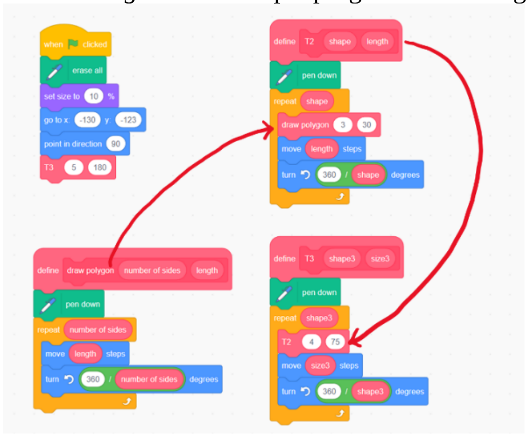
```
define Draw Polygons Number of sides Length
pen down
repeat Number of sides
    move Length steps
    change pen color ▾ by 10
    turn ↺ 360 / Number of sides degrees
pen up
```

(a) (b)

4.4.2. Affordance from the design feature of the tasks

The design features of "multiple" and "regular differences" were the key affordances supporting students' development of both geometrical concepts and the CT practice of reusing and remixing. For example, in this task, the students were encouraged to draw multiple polygons of different sizes and shapes, which would be difficult to perform with paper and pencil. To complete it efficiently, the task prompted students to observe and think about the similarities and differences among these polygons—that is, how the number of sides related to the exterior angles. In addition, during the process of exploring and drawing multiple polygons, the students came to appreciate the use of "functions" to demonstrate reusing and remixing. Figure 7 illustrates one student's work in drawing complex figures. After creating and using the "draw polygon" function as a basic element repeatedly, the student created a new function "T2" based on it, and then used "T2" to create another function "T3" (Figure 7a). The reusing and remixing of existing functions finally yielded a complex drawing (Figure 7b).

Figure 7. A sample program of drawing complex figures with “functions”



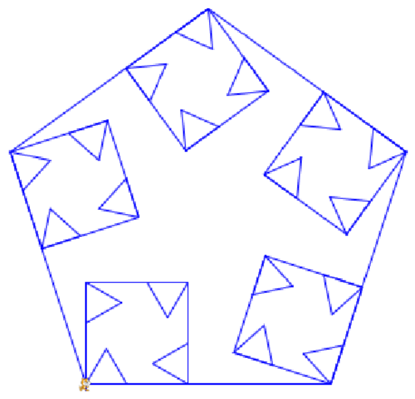
```
when [flag] clicked
erase all
set size to (10) %
go to x: (-130) y: (-123)
point in direction (90)
T3 (5) (180)
```

```
define draw polygon (number of sides) (length)
pen down
repeat (number of sides)
    move (length) steps
    turn ⟲ ((360) / (number of sides)) degrees
```

```
define T2 (shape) (length)
pen down
repeat (shape)
    draw polygon (3) (30)
    move (length) steps
    turn ⟲ ((360) / (shape)) degrees
```

```
define T3 (shape3) (size3)
pen down
repeat (shape3)
    T2 (4) (75)
    move (size3) steps
    turn ⟲ ((360) / (shape3)) degrees
```

(a)



(b)

5. Discussion and conclusion

Based on the four mathematical domains, we presented the findings for the teaching interventions in which the learning tasks were designed and articulated with mathematical problem-based learning in programming environments. The results suggest that these domains, as exemplified by the eight tasks used in the study, provided affordances for the co-development of computational and mathematical aspects of learning. In the following, we explicate some factors that are critical for future research and practice in K–12 mathematics and programming education.

Apart from the intangible affordance provided by the selected content, the use of technology (i.e., Scratch) should be noted, especially in the context of integration with mathematical problems. First, we highlight the functions of “operators” built into the programming tools. The operators contain blocks that provide support for mathematical, logical, and string expressions, enabling the programmer to perform numerical and string manipulations. Some participants had experience using Scratch for programming; however, most only used functions focusing on interactive or narrative projects, such as animations and games (e.g., Kafai & Peppler, 2011). With the codes in the operators, students were able to perform the necessary calculations in order to solve mathematical problems they had not encountered in formal classes. For instance, with the code “mod,” a concept that most students knew about but which was new to them in the programming context, students were able to explore advanced mathematical domains, such as number theory. Thus, this study demonstrated that applying mathematical knowledge to construct CT artifacts plays an important role in solving CT-based mathematical tasks (Bouck & Yadav, 2020; Grizioti & Kynigos, 2021; Miller, 2019).

The second technological aspect we wish to spotlight is the “sensing” function. The capabilities of the sensors included detecting the position of the sprite and mouse and any key input, as well as providing the affordance of human–computer interaction (HCI) via the “ask and answer” code. As suggested by Kafai and Burke (2014), the programming environment should move “beyond the computer screen to meld the digital with the tangible” (p. 91), thus providing additional sensory input and output. Although Scratch’s sensors and HCI functions are virtual, they allow learners to see the actual movement and outcomes of the program visually, similar to the physical world (Ching et al., 2018). Based on our observations, the students showed enthusiasm for making something that was “more like a real product” by using the sensing function, while many students attempted to change the sprite and background to customize the problem’s context. The students’ pursuit of making products is in line with how “learning as making” (Ng & Chan, 2019) pedagogy supports mathematics learning as hands-on and goal-oriented. Making allows learners to actively construct knowledge instead of passively receiving information.

Third, we would like to highlight the feature of the stage area in Scratch. Unlike other text-based programming tools with value or text outputs, many codes were specifically intended to produce visual output in the Scratch stage area. In the current study context, the stage allowed the students to visualize abstract mathematical problems as “authentic.” For instance, in the Two Savings problem, students could program the sprite to report their deposit and even change the images of the sprites to match the context of the problem; in the Dice Rolling problem, the students created sprites to determine the frequency of a target sum and various dynamic representations to visualize the frequency. Importantly, visualization became both the product and process of the students' creation, and this afforded opportunities for the students to engage in mathematical thinking because visualization is one of the vital processes in mathematics learning (Barmby et al., 2007). It is argued that the stage area in Scratch is able not only to prompt the co-development of mathematics and CT knowledge but also to improve the students' ability to visualize in general.

Regarding the selection of mathematical domains and tasks, we identified two types of interaction in the co-development of mathematics and CT: (i) applying mathematical knowledge to construct CT artifacts and (ii) generating new mathematical knowledge along with CT practice (see also Ye et al., 2023). For the first type, students developed CT concepts, including variables, loops, iterations, and conditions, supported by existing mathematical knowledge. Previous research has suggested that the concept of looping in programming is difficult for students to master (Grover et al., 2015; Robins et al., 2003; Zur-Bargury et al., 2013). Also, students experience challenges in creating multiple variables and applying conditions (Cui & Ng, 2021). Given the mathematical context of the current study, the students were prompted to draft an algorithmic solution or create pseudocodes using their mathematical knowledge, which was considered helpful in overcoming difficulties in programming, thus supporting the development of CT concepts (Futschek, 2006; Grover et al., 2014). In addition, the mathematical ideas or relationships related to the problems were constructed based on the students' reflections on the CT outputs. As suggested by Wilkerson-Jerde (2014), students "explore[d] important mathematical properties of [fractal] structures, and offered more ways to construct fractals with particular mathematical properties" (p. 118) by observing a collection of fractals produced by a computer. Pei and colleagues (2018) also found that students could reason and generalize regarding patterns from the data of a number of polygons. The current study provides empirical evidence that the students developed new mathematical ideas, such as experimental probability and limits, through CT practice and outputs. They also constructed mathematical ideas and relationships by working with and reflecting on the CT outputs they created with Scratch.

The current study has both theoretical and practical implications for the integration of CT and mathematics. Theoretically, we clarified and highlighted the meaning of "affordance" in the instructional design. Moreover, we argue that the intangible affordance is even more important in technology-rich learning environments, given that the settings of these environments can vary for different programming languages and hardware. According to the results, the four mathematical domains provided unique opportunities for different CT concepts and practices (i.e., arithmetic – variables; random events – loops; number theory – testing and debugging; geometry – reusing and remixing). Meanwhile, within the given domains, the design features of the tasks played an important role. For example, the design of drawing multiple and different polygons in the domain of geometry prompted higher-order thinking in students regarding the mathematical concept of the limit. On the other hand, the constraints of programming-rich environments for mathematics learning should be acknowledged. In line with the findings of Cui and Ng (2021), the students encountered two types of constraints of affordance, namely mathematics-related constraints (e.g., the challenge of extracting the mathematical rules under the problem) and programming-related constraints (e.g., the challenge of using appropriate code to perform certain functions). We suggest that programming-related constraints could influence students' learning outcomes in significant ways, and they should be minimized when teaching and learning mathematics in programming-rich environments.

In terms of practical contributions, we suggest three areas in which computational problem solving may enrich mathematics learning, as informed by the findings. The first is that the problem should be stated such that the solution and/or the solution process are not immediately known. From this perspective, some of the mathematics-related problems motivated students more than others when presented in the programming context. For example, when first presented, the solution process (e.g., strategies for counting to 21), the solution itself (e.g., a prime detector for large numbers), or both (e.g., experimental probability and fractal geometry) were not immediately obvious to the students. This element of the unknown provided opportunities for students to explore and inquire about new concepts, both in mathematics and programming. Second, computing with screen-based artifacts afforded dynamic visual representation and immediate feedback (such as the movement of the sprite, outputting a certain number, and a figure to be drawn), which significantly engaged the students who participated in this study. As such, the students were more likely to continue, regardless of the complexity and difficulty; thus, the visualization also prompted higher-order thinking toward the CT and mathematics concepts involved (Barmby et al., 2007). It is also worth noting that feedback enables students to manage their learning and mental processes through metacognition (Hesse et al., 2015). Conversely, a task requiring a long procedure with no outputs or feedback should be avoided because students might easily become stuck at some point, as was the case in the "fractal geometry" problem. Third, we consider room for autonomy to be an important characteristic to engage students by providing problems with choices and the customization of solutions. This might be explained by the fulfillment of autonomy, which is regarded as one of the key psychological requirements to support students' motivation to engage in learning tasks (Hsu et al., 2019).

To conclude, this paper described and discussed affordances provided by activities for teaching and learning mathematics in computationally enhanced ways, drawing on selected mathematical domains and tasks. This initial research is highly promising, but more research is warranted to further investigate the design of learning materials for computationally enhanced mathematical teaching and learning.

Acknowledgment

The work described in this paper was fully supported by the Research Grants Council (Hong Kong), General Research Fund (Ref. No. 14603720).

References

Baldwin, D., Walker, H. M., & Henderson, P. B. (2013). The Roles of mathematics in computer science. *ACM Inroads*, *4*(4), 74–80. https://doi.org/10.1145/2537753.2537777

Barab, S., & Squire, K. (2004). Design-based research: Putting a stake in the ground. *Journal of the Learning Sciences*, *13*(1), 1–14. https://doi.org/10.1207/s15327809jls1301_1

Barmby, P., Harries, T., Higgins, S., & Suggate, J. (2007). How can we assess mathematical understanding? In *Proceedings of the 31st Conference of the International Group for the Psychology of Mathematics Education* (Vol. 2, pp. 41–48).

Benton, L., Saunders, P., Kalas, I., Hoyles, C., & Noss, R. (2018). Designing for learning mathematics through programming: A case study of pupils engaging with place value. *International Journal of Child-Computer Interaction, 16*, 68–76. https://doi.org/10.1016/j.ijcci.2017.12.004

Bishop, J. P., Lamb, L. L., Philipp, R. A., Whitacre, I., Schappelle, B. P., & Lewis, M. L. (2014). Obstacles and affordances for integer reasoning: An analysis of children’s thinking and the history of Mathematics. *Journal for Research in Mathematics Education*, *45*(1), 19–61. https://doi.org/10.5951/jresematheduc.45.1.0019

Bouck, E. C., & Yadav, A. (2020). Providing access and opportunity for computational thinking and computer science to support mathematics for students with disabilities. *Journal of Special Education Technology*, *37*(1), 151–160. https://doi.org/10.1177/0162643420978564

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American educational research association, Vancouver, Canada* (Vol. 1, p. 25). http://scratched.gse.harvard.edu/ct/files/AERA2012.pdf

Ching, Y.-H., Hsu, Y.-C., & Baldwin, S. (2018). Developing computational thinking with educational technologies for Young Learners. *TechTrends*, *62*(6), 563–573. https://doi.org/10.1007/s11528-018-0292-7

Corbin, J., & Strauss, A. (2015). *Basics of qualitative research: Techniques and procedures for developing*. Sage.

Cui, Z., & Ng, O.-L. (2021). The Interplay between mathematical and computational thinking in primary school students’ mathematical problem-solving within a programming environment. *Journal of Educational Computing Research*, *59*(5), 988–1012. https://doi.org/10.1177/0735633120979930

De Chenne, A., & Lockwood, E. (2022). A Task to connect counting processes to lists of outcomes in combinatorics. *The Journal of Mathematical Behavior*, *65*, 100932. https://doi.org/10.1016/j.jmathb.2021.100932

DeJarnette, A. F. (2019). Students’ challenges with symbols and diagrams when using a programming environment in mathematics. *Digital Experiences in Mathematics Education*, *5*(1), 36–58. https://doi.org/10.1007/s40751-018-0044-5

Futschek, G. (2006). Algorithmic thinking: the key for understanding computer science. In *Informatics Education–The Bridge between Using and Understanding Computers: International Conference in Informatics in Secondary Schools–Evolution and Perspectives, ISSEP 2006, Vilnius, Lithuania, November 7-11, 2006. Proceedings* (pp. 159–168). Springer.

Gibson, J. J. (1979). *The Ecological approach to visual perception.* Houghton Mifflin Co.

Grizioti, M., & Kynigos, C. (2021). Code the mime: A 3D programmable charades game for computational thinking in MaLT2. *British Journal of Educational Technology, 52*(3), 1004–1023. https://doi.org/10.1111/bjet.13085

Grover, S., Cooper, S., & Pea, R. (2014). Assessing computational learning in K-12. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education* (pp. 57–62). ACM.

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

Guzdial, M., & Soloway, E. (2003). Computer science is more important than calculus. *ACM SIGCSE Bulletin, 35*(2), 5–8. https://doi.org/10.1145/782941.782943

Hartson, R. (2003). Cognitive, physical, sensory, and functional affordances in interaction design. *Behaviour & Information Technology*, *22*(5), 315–338. https://doi.org/10.1080/01449290310001592587

Hesse, F., Care, E., Buder, J., Sassenberg, K., & Griffin, P. (2015). A Framework for teachable collaborative problem-solving skills. In P. Griffin & E. Care (Eds.), *Assessment and teaching of 21st century skills: Methods and approach* (pp. 37–56). Springer.

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

Ho, W. K., Lool, C. K., Huang, W., Seow, P., & Wu, L. (2021). Computational thinking in mathematics: To be or not to be, that is the question. In *Mathematics—Connection and Beyond: Yearbook 2020 Association of Mathematics Educators* (pp. 205–234). https://doi.org/10.1142/9789811236983_0011

Hsu, H., Wang, C., & Levesque-Bristol, C. (2019). Reexamining the impact of self-determination theory on learning outcomes in the online learning environment. *Education and Information Technologies*, *24*(3), 2159–2174.

Jong, M. S. Y., Geng, J., Chai, C. S., & Lin, P. Y. (2020). Development and predictive validity of the computational thinking disposition questionnaire. *Sustainability, 12*(11), 4459. https://doi.org/10.3390/su12114459

Jong, M. S. Y., Song, Y., Soloway, E., & Norris, C. (2021). Teacher professional development in STEM education. *Educational Technology & Society, 24*(4), 81–85.

Kafai, Y. B., & Burke, Q. (2014). *Connected code: Why children need to learn programming*. MIT Press.

Kafai, Y. B., & Peppler, K. (2011). Youth, technology, and DIY. *Review of Research in Education*, *35*(1), 89–119. https://doi.org/10.3102/0091732x10383211

Kirschner P. A. (2002). Can we support CCSL? Educational, social and technological affordances for learning. In *Three worlds of CSCL: Can we support CSCL?* (pp. 7–34). The Open Universiteit Nederland.

Leung, A. (2020). Boundary crossing pedagogy in STEM education. *International Journal of STEM Education*, *7*(1). https://doi.org/10.1186/s40594-020-00212-9

Lockwood, E., & De Chenne, A. (2019). Enriching students' combinatorial reasoning through the use of loops and conditional statements in Python. *International Journal of Research in Undergraduate Mathematics Education, 6*(3), 303–346. https://doi.org/10.1007/s40753-019-00108-2

Lockwood, E., & Mørken, K. (2021). A call for research that explores relationships between computing and mathematical thinking and activity in Rume. *International Journal of Research in Undergraduate Mathematics Education. 7*(3), 404–416. https://doi.org/10.1007/s40753-020-00129-2

Miller, J. (2019). STEM education in the primary years to support mathematical thinking: Using coding to identify mathematical structures and patterns. *ZDM*, *51*(6), 915–927. https://doi.org/10.1007/s11858-019-01096-y

Ng, O.-L., & Chan, T. (2019). Learning as making: Using 3D computer-aided design to enhance the learning of shape and space in STEM-integrated ways. *British Journal of Educational Technology*, *50*(1), 294–308. https://doi.org/10.1111/bjet.12643

Ng, O.-L., & Cui, Z. (2021). Examining primary students' mathematical problem-solving in a programming context: Towards computationally enhanced mathematics education. *ZDM – Mathematics Education*, *53*(4), 847–860. https://doi.org/10.1007/s11858-020-01200-7

Ng, O.-L., Liu, M., & Cui, Z. (2021). Students' in-moment challenges and developing maker perspectives during problem-based digital making. *Journal of Research on Technology in Education*, 1–15. https://doi.org/10.1080/15391523.2021.1967817

Nordby, S. K., Bjerke, A. H., & Mifsud, L. (2022). Computational thinking in the primary mathematics classroom: A Systematic review. *Digital Experiences in Mathematics Education*, *8*(1), 27–49. https://doi.org/10.1007/s40751-022-00102-5

Norman, D. A. (1999). Affordances, conventions, and design. *Interactions, 6*(3) 38–43. https://doi.org/10.1145/301153.301168

Papert, S. A. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic books.

Parnafes, O., & diSessa, A. A. (2013). Microgenetic learning analysis: A Methodology for studying knowledge in transition. *Human Development*, *56*(1), 5–37. https://doi.org/10.1159/000342945

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. https://doi.org/10.1080/10986065.2018.1403543

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, *13*(2), 137–172. https://doi.org/10.1076/csed.13.2.137.14200

Román-González, M., Pérez-González, J. C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? criterion validity of the computational thinking test. *Computers in Human Behavior*, *72*, 678–691. https://doi.org/10.1016/J.CHB.2016.08.047

Sneider, C., Stephenson, C., Schafer, B., & Flick, L. (2014). Exploring the science framework and NGSS: Computational thinking in the science classroom. *Science Scope*, *38*(3), 10–15. https://doi.org/10.2505/4/ss14_038_03_10

So, H. J., Jong, M. S. Y., & Liu, C. C. (2020). Computational thinking education in the Asian Pacific region. *The Asia-Pacific Education Researcher, 29*(1), 1–8.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology, 25*(1), 127–147. https://doi.org/10.1007/s10956-015-9581-5

Weng, X., Cui, Z., Ng, O. L., Jong, M. S. Y., & Chiu, T. K. F. (2022). Characterizing students' 4C skills development during problem-based digital making. *Journal of Science Education and Technology, 31*(3), 372–385. https://doi.org/10.1007/s10956-022-09961-4

Wilkerson-Jerde, M. (2014). Construction, categorization, and consensus: Student generated computational artifacts as a context for disciplinary reflection. *Educational Technology Research and Development, 62*(1), 99–121. https://doi.org/10.1007/s11423-013-9327-0

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

Wing, J. M. (2011). Research notebook: Computational thinking—What and why. *The Link Magazine, 6,* 20–23.

Ye, H., Liang, B., Ng, O.-L., & Chai, C. S. (2023). Integration of computational thinking in K-12 mathematics education: A systematic review on CT-based mathematics instruction and student learning. *International Journal of STEM Education, 10*(1). https://doi.org/10.1186/s40594-023-00396-w

Zur-Bargury, I., Parv, B., & Lanzberg, D. (2013). A Nationwide exam as a tool for improving a new curriculum. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education* (pp. 267–272). ACM.

Appendix 1. Brief description of the tasks included in the study

Two Savings problem. There are two money-saving strategies. The first plan starts with \$3 and then deposits \$222 every day thereafter. The second plan starts with \$3 and then deposits double the amount of the last deposit (i.e., \$6, \$12) on each subsequent weekend (i.e., Saturday and Sunday). The problem asks students to determine which saving strategy is more optimal given different saving periods.

The mathematical concepts involved were arithmetic and geometric sequences. In solving the problem, the students were asked first to sketch a graph of the bank balance over time for each saving plan. Then, they were to model the respective saving processes with programming.

Fibonacci Sequence Generator. Observe the following number sequence: 1, 1, 2, 3, 5, 8, 13, …, what is the next term? What is the 50th term? Create a program that could find any term in this sequence.

The mathematical concept involved was sequences. The students were prompted to create a program that included input and output. The input indicates the number of items in the sequence, and the program outputs the value of the corresponding term.

Dice Rolling problem. When we roll six dice together, we can obtain the sum of the results. If we have to guess the sum of the six dice given four choices, 19, 20, 21, or 22, which one should we choose?

This was a two-part problem. The first part was programming a dice-rolling simulator with computer-generated random numbers and observing the sum upon a certain number of simulations. The second part was generating the outcome space (e.g., 1-1-1-1-1-1, 1-1-1-1-1-2, … 6-6-6-6-6-6), as well as the frequency of obtaining a given sum theoretically. For example, to obtain the sum of 6, the frequency was only one, that is, 1-1-1-1-1-1; but to obtain the sum of 7, the frequency was six (1-1-1-1-1-2, 1-1-1-1-2-1, …, 2-1-1-1-1-1). The mathematical concept involved in the task was classic probability. However, because the students had not yet learned the concept of probability, we avoided using the term and let the students experiment with the concept on their own.

Dart Throwing problem. There is a rectangular dartboard made of square regions of different sizes (i.e., side lengths of 1, 1, 2, 3, 5, and 8). When throwing darts at the board, the darts will either hit one of the square regions or miss the board completely. Design a fair game scoring system to indicate the number of points a player should gain when the darts land on the different squares.

The mathematical concept involved was unequal likelihood outcome space. To solve the problem, the students would simulate throwing darts at this specific board a large number of times and record the frequencies of the darts hitting the various squares using programming.

Prime Detector. Is 7081 a prime number? Create a device that could determine whether a number, such as 7081, is a prime or composite number.

The mathematical concepts involved were divisibility rules, factors, and multiples. The students were introduced to the code "mod" to determine the remainder of a division operation.

Counting to 21 (or 100). This is a game with two players. The players take turns calling either 1 or 2 (or 1 to 9 in the game of counting to 100), and the program will record and add all the numbers being called. For example, at the beginning, if Player A calls 2, the program will show 2; then, if Player B calls 1, the program will show 3, and so on. The player who gets the program to show 21 wins the game. The students were invited to, first, create the program and, then, play the game with their partners. Then, they would design a program with a computerized player in which the human player goes first such that the computerized player will always win.

In this two-part task, the mathematics involved observing that a winning strategy was to ensure that, upon the human opponent taking the first turn to call a number, the computer will call a number such that the sum is a multiple of three.

Drawing Polygons. Draw different regular polygons with the Pen function in Scratch.

The mathematical concepts involved were exterior and interior angles and the number of sides. The students were first shown how to draw an equilateral triangle, and then, they were to explore drawing various (regular) polygons.

Drawing Fractals. Observe the following geometric figures: a Sierpinski triangle, a fractal tree, and a Koch curve. What are some common features between them? Create a program to draw one of these fractal geometries and, then, design your own fractal geometry.

The mathematical concept involved in this task was recursion. The students were not introduced to the mathematical definition of recursion (for example, using a typical factorial example, which can be represented by the recursive formula, $f(n) = f(n-1) \times n$); rather, they were instructed to conceptualize a recursion as "a function calling the function itself" in programming.