Chapter 9

Learning Composite and Prime Numbers Through Developing an App: An Example of Computational Thinking Development Through Primary Mathematics Learning

Siu-Cheung Kong

Abstract Computational Thinking (CT) is a universal competence that young learners should acquire. This study illustrates the development of CT through app development for the learning of composite and prime numbers in primary school mathematics. Learners begin with inquiry activities regarding the meaning of composite numbers and prime numbers, and they are guided to find the factors of a given number by exploring an algorithm that involves the concept of modulo and can be implemented in a programming environment. Learners use a simple app to learn how the operator MOD works and then work in pairs to design an algorithm and devise a programming solution to find the factors of a given number by reusing the code of the simple app. The app will tell the users whether an inputted number is a composite or prime. Learners will make an initial connection with cryptography when large numbers such as 563,879 are explored, and they will obtain a more in-depth understanding of composite and prime numbers when 1 and 0 are tested in the app. In the process of building the app, learners will learn CT concepts such as sequence, loops, conditionals, events, operators and procedures. Learners will experience CT practices like reusing and remixing, being iterative and incremental, testing and debugging, abstracting and modularising and algorithmic thinking in developing the app. Learners will develop CT perspectives in the process of expressing, questioning and connecting with the digital world in developing the app.

S.-C. Kong (✉)

Centre for Learning, Teaching and Technology, The Education University of Hong Kong,
10 Lo Ping Road, Tai Po, Hong Kong
e-mail: sckong@eduhk.hk

© The Author(s) 2019

S.-C. Kong and H. Abelson (eds.), *Computational Thinking Education*,
https://doi.org/10.1007/978-981-13-6528-7_9

Keywords App development · Composite numbers · Computational thinking · Primary school mathematics · Prime numbers

9.1 Introduction

Researchers and practitioners in education advocate the introduction of Computational Thinking (CT) in P-12 education to nurture learners' problem-solving and creativity (Barr & Stephenson, 2011; Grover & Pea, 2013; Lye & Koh, 2014). The goal of CT development is to enable learners to draw on the fundamental concepts of computer science to formulate problems and find computational solutions to real-life problems (Wing, 2011). The usefulness of app development is not limited to the field of computer science, and it can also help students to learn in a wide range of subjects. Computational tools and skill sets are helpful in mathematics and science learning (Eisenberg, 2002; Repenning, Webb, & Ioannidou, 2010; Wilensky 1995; Wilensky, Brady, & Horn, 2014), while mathematics and science offer a meaningful context in which problems can be formulated and CT can be developed (Hambrusch, Hoffmann, Korb, Haugan, & Hosking, 2009; Jona et al., 2014; Wilensky et al., 2014). This reciprocity is the motivation to combine CT and mathematics and science learning. Many P-12 learners have started to learn mathematical concepts through app development (Kahn, Sendova, Sacristán, & Noss, 2011; Wolber, Abelson, & Friedman, 2014; Yadav, Mayfield, Zhou, Hambrusch, & Korb, 2014). To teach novice programmers, block-based programming environments, such as Scratch and App Inventor, are considered appropriate (Meerbaum-Salant, Armoni, & Ben-Ari, 2010; Price & Barnes, 2015). This study, therefore, demonstrates how mathematical concepts can be developed through the creation of an app in App Inventor using an example of learning composite and prime numbers in primary school mathematics. The experience of creating an app helps learners turn abstract pattern recognition into an algorithmic and concrete operational experience.

9.2 Background

Wing's (2006) article popularised the term 'computational thinking' and defined it as the 'thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent' (p. 1). Computational thinking relates to design, problem-solving and the understanding of human behaviour and draws on the basic concepts of computer science. Although it originates from computer science, CT is not exclusive to that field, and it goes well beyond computing. It is 'reasoning about the problems in computational terms' (Syslo & Kwiatkowska, 2014, p. 2), which means that learners are able to formulate problems in ways that help to develop computer solutions.

9.2.1 CT Framework

Brennan and Resnick developed a CT framework (2012) that was structured in three dimensions: CT concepts that learners engage with in programming, CT practices that learners develop when solving computational problems and CT perspectives that learners gain when interacting with technologies and relate to their experience with the digital world. Regarding CT concepts, learners are expected to learn the basic ingredients of programming such as sequences, conditionals, loops, events, operators and data handling. When tackling computational problems, learners experience CT practices such as reusing and remixing, being incremental and iterative, abstracting and modularising, testing and debugging, and algorithmic thinking. In the programming process, learners have opportunities to develop CT perspectives such as expressing, questioning and connecting with the digital world. Expressing refers to learners' opportunities to use programming as a medium for self-expression. Questioning is the ability to ask questions about and with technology. Connecting is the process by which learners create with and for others in programming. It helps learners to experience working with other people and can lead to more accomplishments resulting from discussions and collaborations. Creating for others enables learners to experience positive and negative feedbacks in authentic contexts when their creations are appreciated (or not appreciated) by other people. Connecting is the most important component of CT perspectives, as it enables learners to connect CT practices with the digital world so that they feel empowered and have an identity in the digital world. All three dimensions of CT are essential for implementing CT in school education.

9.2.2 Block-Based Programming Environments

Inspired by LogoBlocks (Begel, 1996), block-based environments allow learners to create apps by dragging and dropping blocks into a scripting pane with minimal effort spent dealing with programming syntax. It minimises syntax errors because the blocks can only fit together when the code makes sense. In addition, feedback is immediate, as the results of the code blocks that the learners build are shown right in front of them when the program is executed. Such a user-friendly programming environment enables novice programmers to be more engaged in thinking about the programming process in a more systematic manner as they build the code block by block. Scratch (Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010) and App Inventor (Wolber, Abelson, Spertus, & Looney, 2011) are the most common block-based programming environments used by novice programmers (Price & Barnes, 2015). This study illustrates how App Inventor was used to design an app for learning primary mathematics and CT development.

9.2.3 Mathematics Learning and CT Development

Learners can develop CT in learning mathematics when programming is introduced as pedagogy. Learners can develop CT in learning a variety of subjects, including mathematics, science, social studies and language studies (Barr & Stephenson, 2011). CT and mathematics are particularly closely related. This study connects CT and mathematics learning explicitly to show that learners can develop both CT and mathematical concepts holistically. The National Research Council (NRC) suggested that mathematics and CT should be essential practices for the scientific and engineering aspect of the Framework for K-12 Science Education (NRC, 2012). Programming is pedagogically helpful in mathematics learning (Eisenberg, 2002; Repenning, Webb, & Ioannidou, 2010; Wilensky, 1995; Wilensky, Brady, & Horn, 2014). Syslo and Kwiatkowska (2014) advocated that traditional topics in school mathematics should be extended and enriched through the application of programming to build solutions. The literature review conducted by Lye and Koh (2014) revealed that learners learn mathematics well with the help of programming (e.g. Kahn, Sendova, Sacristán, & Noss, 2011). The study conducted by Yadav et al. (2014) found that teachers favoured the implementation of CT into as many subjects as possible. Mathematics is most preferred as CT always involves problem-solving with mathematics, such as design algorithms and heuristics in the solving process. Research studies have documented that the immersion of CT in mathematics learning can be successfully achieved by building apps. In the study conducted by Morelli et al. (2010), participating teachers and learners developed a learning app to promote the immersion of CT in traditional subjects, such as mathematics. In Ke's study (2014), a group of secondary school learners used Scratch to design mathematics games that successfully enhanced their mathematics learning outcomes. The study by Foerster (2016) documented similar outcomes in a high school that integrated programming into its mathematics curriculum by combining Scratch and geometry learning in Grades 6 and 7. Long-term assessments indicated that these education initiatives resulted in significant improvements in their learners' performance. Hsu and Hu (2017) conducted a study in which Grade 6 learners were guided by teachers to apply CT to write a block program in Scratch to solve daily-life mathematic problems. It was proved successful in enhancing learners' mathematics learning effectiveness. Evidence indicates that CT can be integrated into meaningful mathematics learning through app development. This study demonstrates how programming can serve as pedagogy in the learning and teaching of primary school mathematics for better mathematics learning and CT development.

9.2.4 Learning Composite and Prime Numbers in Primary School Mathematics

One of the traditional methods of learning composite and prime numbers is through the sieve of Eratosthenes. While it is generally taught at the tertiary level, its simplified version is introduced in primary school: ‘sifting out the composite numbers, leaving only the primes’ (Maletsky et al., 2004). Ploger and Hecht (2009) suggested using the software Chartworld to introduce the sieve of Eratosthenes to primary learners in a visualised and interactive way. Chartworld colours all multiples of a number when that number is selected. For instance, after selecting the number 3, all multiples of 3 except 3 will be highlighted. After selecting the numbers from 2 to 10, all of their multiples will be highlighted. The remaining un-highlighted numbers are prime numbers, except 1. Although this method enables young learners to learn about composite and prime numbers, it is limited in finding prime numbers in a given range. In this regard, there is a need to explore an alternative way to teach primary school learners so that they can explore larger composite and prime numbers. This study proposes an innovative pedagogy to guide primary school learners to make an app that can find the factors of an inputted number.

9.3 Developing an App as Pedagogy for Supporting the Conceptual Understanding of Composite and Prime Numbers

In traditional mathematics lessons, senior primary school learners are taught that prime numbers are only be divisible by 1 and themselves, while composite numbers have factors other than 1 and themselves. Yet mere rote memorisation of these definitions is not encouraged, as learners will not truly understand the concepts of composite and prime numbers. This study introduces an inquiry pedagogical approach to guide learners to develop these mathematics concepts together with the development of an app. Learners will be initially introduced to examples of and ideas underlying composite and prime numbers. Learners will then develop an app that finds the factors of an inputted number and tests whether they can distinguish between composite and prime numbers. The programming process helps learners to think more deeply about the meaning of composite and prime numbers.

9.3.1 Inquiry Activities About Composite and Prime Numbers

This study introduces the concepts of composite and prime numbers using an inquiry approach, which is a learning process whereby ‘students are involved in their learning, formulate questions, investigate widely and then build new understandings, meanings

Example: 8 = 1 × 8 = 2 × 4	4 × 2 is not needed because it is the same as 2 × 4. 8 × 1 is not needed because it is the same as 1 × 8.
The factors of 8 are 1, 2, 4 and 8.	

Fig. 9.1 Find all factors of 8 by listing out all factorisations of 8

and knowledge’ (Alberta Learning, 2004, p. 1). In this activity, learners will be asked what a composite number and a prime number are. Teachers will then explain the concepts with examples. Numbers with more than two factors are composite numbers, while numbers with only two factors—1 and themselves—are prime numbers. When finding the factors of a number, learners are reminded of the commutative properties of multiplication. For instance, 8 can be expressed as ‘1 × 8 = 8’, ‘2 × 4 = 8’, ‘4 × 2 = 8’ and ‘8 × 1 = 8’. Because the products are the same irrespective of the order of the factors, learners know that ‘4 × 2 = 8’ and ‘8 × 1 = 8’ are equivalent to ‘2 × 4 = 8’ and ‘1 × 8 = 8’, respectively. When finding all of the factors of 8, learners only need to explore ‘1 × 8 = 8’, ‘2 × 4 = 8’ and do not need to consider ‘4 × 2 = 8’ and ‘8 × 1 = 8’. The order of operation is not important in finding the factors of a number, and this is the commutative property of multiplication. This study encourages learners to generalise composite and prime numbers from examples. The concepts of composite and prime numbers can also be better understood by asking young learners to explore the factors of a number through visualisation activities, such as exploring with the online activity ‘Find the factorisations of the number’ in the Illumination Project (National Council of Teachers of Mathematics [NCTM], n.d.). By choosing a particular number, such as 8, learners are invited to represent all of the factorisations of 8 by selecting the appropriate rectangular planes.

Learners are guided to conclude that composite numbers can be represented by more than one rectangular plane, while prime numbers can only be represented by a single rectangular plane. This indicates that composite numbers have more than one pair of factorisations, while prime numbers have only a single pair of factors: 1 and themselves. Figure 9.1 shows a way to find all of the factors of 8 by listing all of its factorisations. Using this form of representation, young primary school learners can be easily guided to categorise numbers as composite or prime by counting the factors of a selected number. Figure 9.2 lists examples of numbers that have only two factors and numbers that have more than two factors in a range from 2 to 12.

9.3.2 *Developing an App as Pedagogy*

After introducing composite and prime numbers by counting the number of factors of a number, teachers can guide learners to develop an algorithm to generalise a method

The numbers that only have two factors from 2 to 12	The numbers that have more than two factors from 2 to 12
2, 3, 5, 7, 11	4, 6, 8, 9, 10, 12

Fig. 9.2 Categorising composite and prime numbers by counting the number of factors of a number in the range from 2 to 12

Fig. 9.3 The sample screens display the results of testing the inputted number 16 when **a** the ‘Prime Number’ button is pressed and **b** the ‘Composite Number’ button is pressed

to differentiate whether a number is composite or prime. Learners can input any integral numbers greater than or equal to one and ask the program to identify and list all factors of the given number. Learners are asked to decide whether it is composite or prime based on the number of factors found and listed by the app. They can check the answer by pressing the ‘Composite Number’ button or the ‘Prime Number’ button. They are asked to explore with the app by inputting different numbers before starting the programming task. Figure 9.3 shows sample screens displaying the results of testing an inputted number 16 when (a) the ‘Prime Number’ button is pressed and (b) the ‘Composite Number’ button is pressed.

9.3.3 Problem Decomposition and Algorithmic Thinking

There is a need to guide learners to figure out the sub-tasks in building the app. Decomposing the problems into sub-tasks is a high-level abstraction process. The four sub-tasks are finding the factors of the inputted number, showing the factors on the screen, deciding whether the number is a prime number and deciding whether the number is a composite number. Figure 9.4 shows the high-level abstraction involved in decomposing the problem into sub-tasks.

The most challenging task is to facilitate learners to find ways to check whether a number is a factor of an inputted number. The use of examples is the best way to guide learners. For example, learners can be asked to fill in the columns ‘Modulo’ and ‘Factor of 8?’ to check whether 1, 2, 3, 4, 5, 6, 7 and 8 are factors of 8 when it is divided by 1, 2, 3, 4, 5, 6, 7 and 8, respectively, as listed in Table 9.1. Table 9.1 shows all of the results in columns ‘Modulo’ and ‘Factor of eight?’ when eight is divided by 1, 2, 3, 4, 5, 6, 7 and 8, respectively. Learners are encouraged to discuss in pairs to identify the pattern of zero modulo in identifying the factors of the given number.

The operator of modulo in programming needs to be introduced to learners to prepare them to find an algorithm to check whether a number is a factor of a given

Fig. 9.4 A high-level abstraction of decomposing the problem into four sub-tasks

Table 9.1 A table with columns ‘Modulo’ and ‘Factor of 8?’ to be filled in by learners identifying patterns to find the factors of a given number

Division	Divisor	Modulo	Factor of 8?
8 ÷ 1	1	0	✓
8 ÷ 2	2	0	✓
8 ÷ 3	3	2	
8 ÷ 4	4	0	✓
8 ÷ 5	5	3	
8 ÷ 6	6	2	
8 ÷ 7	7	1	
8 ÷ 8	8	0	✓

Fig. 9.5 The interface and code for learners to understand the operator of modulo in the programming environment

Fig. 9.6 An algorithm for finding all of the factors of an inputted number

number. Figure 9.5 shows the interface and the code for learners to understand the operator of modulo in the programming environment.

Learners are then asked to work in pairs to design an algorithm to find all factors of an inputted number. Figure 9.6 shows an algorithm for finding all factors of an inputted number.

9.3.4 Reusing Code from a Simple App to Build an App to Find Factors

Following the algorithmic design, learners can start to build an app to find all of the factors of a given number by using the operator of modulo. Figure 9.7 shows the code of a program for finding all of the factors of a positive integral number inputted by a learner.

Figure 9.8 shows the code executed when the ‘Prime Number’ button and the ‘Composite Number’ button are clicked, respectively. The code in ‘primeButton’ and ‘compButton’ counts the number of factors of the input number to check whether it is a prime or composite number.

Fig. 9.7 The code of a program for finding all of the factors of a positive integral number

Fig. 9.8 The code executed when the ‘Prime Number’ button and the ‘Composite Number’ button are clicked, respectively

9.3.5 *Testing the App and Connecting the Tasks with the Digital World*

Learners can then test the program by checking whether the input number is a prime number or a composite number. The possible answers are ‘This is a prime number’ or ‘This is not a prime number’ when the ‘Prime Number’ button is clicked and ‘This is a composite number’ or ‘This is not a composite number’ when the ‘Composite Number’ button is pressed. Figure 9.9 shows the sample screens displaying the results

Fig. 9.9 Sample screens displaying the results of testing the inputted number 953 when **a** the ‘Composite Number’ button and **b** the ‘Prime Number’ button are pressed

of testing the inputted number 953 when (a) the ‘Composite Number’ button and (b) the ‘Prime Number’ button are pressed, respectively.

This app enables learners to test whether large positive numbers are prime or composite numbers, and they will thus come to understand the importance of the application of such testing in cryptography. This mathematical topic is selected as the theme of this computational task because of the significant role of prime numbers in cryptography. Email communication, instant messaging and satellite television, which are ubiquitous in everyday life, often require data encryption for the safe delivery of information. Encryption methodologies such as RSA use trapdoor one-way functions to encrypt messages. These functions are easy to compute in the forward direction, but difficult to compute in reverse without special information. RSA utilises some properties of prime number multiplication to design a function that satisfies the definition of the trapdoor one-way function (Salomaa, 1996). To enable learners to experience a one-way function, multiplication and factorisation are used as examples in this study to demonstrate the forward and reverse operations, respectively. The app developed in this study enables learners to test whether a large positive number is a composite or prime number. For example, the app can provide the factors of a relatively large number, such as 563,879, from the perspective of

Fig. 9.10 The factors of a relatively large number are displayed after a time gap

young primary learners. As shown in Fig. 9.10, the factors of 563879 are 569 and 991, which are prime. Learners have to wait several minutes before they can see the outcomes, during which time they can discuss and connect the function with the work of encryption. The larger the number, the more time will be needed to identify its factors. It is important for us to find sufficiently large numbers for the encryption work so that our messages can be transmitted safely in all digital communications.

9.3.6 Using ‘1’ and ‘0’ to Trigger In-depth Discussion of Composite and Prime Numbers

Learners are then guided to find the number of factors of the number 1, which is one. The outcomes will be ‘This is not a prime number’ and ‘This is not a composite number’. At this stage, teachers can emphasise the definition of a prime number and explain why 1 is not considered prime, which is rarely discussed in teaching. This app can thus help teachers to trigger discussion among learners in classrooms. Figure 9.11 shows the sample screens displaying the results of testing an inputted number 1 when (a) the ‘Composite Number’ button and (b) the ‘Prime Number’ button are pressed.

Learners are then asked to find number of factors of the number 0. The app will return an empty list of factors because it is not designed to handle this case. There are infinitely many factors for 0 when it is checked by the rule ‘the inputted number can be divided by a positive integer and it is a factor when the modulo is zero’. When 0 is divided by 1, the modulo is 0. Similarly, when 0 is divided by 2 or any

Fig. 9.11 The sample screens displaying the results of testing an inputted number 1 when **a** the ‘Composite Number’ button and **b** the ‘Prime Number’ button are pressed

other positive number, the modulo is 0. Therefore, 0 has infinitely many factors, and the app will report ‘There are infinitely many factors’ when 0 is inputted for testing. Thus, learners can understand the limitations of the computer program regarding this case, which is worthy of discussion among learning peers and teachers. Figure 9.12 shows the message after clicking the ‘Show All Factors of the Number’ button when 0 is inputted.

9.3.7 Adding a Conditional Statement to the App to Handle the Case of Inputting 0

Learners are then asked to write a conditional statement to respond to the input of 0. Figure 9.13 shows the conditional statement for handling the special case when 0 is inputted. This is an opportunity for learners to understand program debugging. Learners will know that programming an app is an iterative process and that the program needs to be completed incrementally. Learners can reuse the code for the ‘findFactors’ function shown in Fig. 9.13 to search for factors.

With the categorisation of composite and prime numbers taking place in the above manner, this study proposed four categories of numbers: composite numbers, prime

Fig. 9.12 The message ‘There are infinitely many factors’ after clicking the ‘Show All Factors of the Number’ button when 0 is inputted

Fig. 9.13 The conditional statement for handling the case in which 0 is inputted in this app

numbers, 1 and 0. In this regard, the definition of a composite number should be revised from the original definition, ‘a number with more than two factors’, to ‘a number with more than two factors but a finite number of factors’.

9.4 Computational Thinking Development

9.4.1 CT Concepts Development

In the process of building the app, learners learn CT concepts such as sequence, loops, conditionals, events, operators, data handling and procedures (see Fig. 9.14). The concept of sequences can be developed in building the ‘when showButton.Click’ program. Learners need to think carefully about the order of finding the factors after considering the special case of 0. When building the ‘when showButton.Click’ program, learners will first consider whether the input number is 0. If so, the app should tell users that it has an infinite number of factors instead of listing all of its factors. If the input number is not 0, the program should call the ‘findFactors’ procedure, as all numbers except 0 have a finite number of factors. Learners will thus understand the concept of sequence in greater depth in this case if they understand that the order of instructions for finding factors or assigning the message ‘There are infinitely many factors’ to the factorList should always come before showing the factors or the message. The concept of repetition is also developed when building the ‘findFactors’ codes. Using the ‘for each from to’ block, the program repeatedly runs the blocks in the do section, testing the numbers ranging from 1 to the input number in finding all of the factors. Learners can understand the concept of events when using the ‘when showButton.Click’ block. They will observe when an event occurs (i.e. clicking the ‘show’ button), and the app will trigger a series of tasks for execution until completion. They can also learn the concept of naming/variables when they create a factor list with the ‘initialise global name to’ block. The concept of conditionals is embedded in the use of modulo to find the factors of the input number. Using the ‘if-then’ block, the program will add the number to the factor list if the modulo is 0 after the input number is divided by the number. Learners can also develop the concept of operators when using the modulo to check the remainders of the divisions and using ‘greater than’ and ‘equals’ blocks in the ‘when primeButton.Click’ and ‘when compButton.Click’ programs. The concept of data is introduced in the use of lists. After the program finds the factors by the modulo, learners need to store all of the factors in the list before showing them. Learners can also understand the use of procedures in a complex computational task like this one. After the ‘findFactors’ program is built, the app will show the list of factors on the screen. Therefore, a procedure that shows factors should be created. To show the factor list, the ‘showButton’ is made. When the ‘showButton’ is clicked, the program calls the ‘findFactors’ and ‘showFactors’ procedures that were created previously. Learners thus know that using procedures can avoid bulky programs.

Fig. 9.14 The CT concept learned from the factor app

9.4.2 CT Practices Development

Learners can also experience CT practices, such as reusing and remixing, being iterative and incremental, testing and debugging, abstracting and modularising, and algorithmic thinking, in developing the app. They experience the practice of reusing and remixing when they use the code block of the modulo from the simple app (see Fig. 9.5) to create the ‘findFactor’ program (see Fig. 9.15). To reuse the code block of the modulo from the simple app in making the Factor app, learners can save the code in the backpack, and it can then be extracted when building the ‘findFactors’ program. The backpack is a copy-and-paste feature in App Inventor that allows users to copy blocks from a previous workspace and paste them into the current workspace.

Learners can also experience the practice of being incremental and iterative in the process of app development (see Fig. 9.16). Learners can be guided to initially build a program that finds and shows the factors of an input number and then put the program into testing and debugging. Learners can develop the program further to check whether the inputted number is a prime or a composite number. Thus, learners can experience the process of developing the app incrementally and iteratively.

Learners learn also about testing and debugging throughout the app development process in this example. One particular point concerns the testing of the case of inputting 1 and 0 in this example. When the user presses either of the buttons to test whether 1 is a prime or composite number, the app will give no response (see Fig. 9.17), as the original app only functions when the inputted number has two or more factors. When the inadequacy is identified, learners have to modify the program

Fig. 9.15 Reuse and remix the modulo code from the simple app using the backpack feature in App Inventor

Fig. 9.16 Learners develop the app incrementally and iteratively

to handle the special case of inputting 1. Moreover, an error will be found if 0 is input. Because 0 has an infinite number of factors, the app cannot list them; thus, the app needs to be enhanced to inform the user that 0 has an unlimited number of factors, as shown in Fig. 9.17. From these examples, learners will understand the importance of testing and debugging in app development.

It is not realistic to expect primary school learners are able to conduct high-level abstraction in programming design; however, this app development exposes learners to the high-level abstraction process. The four main sub-tasks highlighted at the beginning of programming design (i.e. finding factors and showing factors of the input number, checking whether the input number is a prime number and checking whether the input number is a composite number) can be shown to learners if they are unable to generate the idea after discussion. Teachers should not be frustrated if learners are unable to decompose problems into sub-tasks at this stage, but they should expose them to this process so that they appreciate that it is an important part of the programming process. Before building the code, learners should be motivated to conduct an algorithmic design that delineates the detailed instructions required to solve the problem. This process facilitates the development of algorithmic thinking, which is an essential cognitive skill in CT development (Angeli et al., 2016; Barr & Stephenson, 2011; Wing, 2006). Again, we should not expect learners to be able to work this out on their own; however, they should be guided and exposed to this process so that the seeds are sown for later development and growth. Individuals with algorithmic thinking skills can help them to precisely state a problem, divide the problem into well-defined sub-tasks and work out a step-by-step solution to tackle

Fig. 9.17 Learners experience testing and debugging after 1 and 0 are inputted

each sub-task (Cooper, Dann, & Pausch, 2000). Figure 9.6 shows the algorithm for finding the factors with modulo.

As mentioned in Chap. 8, problem formulation is an essential component of CT practices. With the experience gained from developing the factor app, learners should be encouraged to formulate problems using their pattern recognition experience. Learners can be guided to reflect again on the pattern-identifying factors of a number by considering the outcome of the modulo when the number is divided by a divisor (Table 9.1). Teachers can ask learners to look for other patterns for formulating problems, for instance, learners may be able to discover patterns such as those shown in Table 9.2, which presents a pattern of identifying odd numbers using a similar strategy to that in Table 9.1.

Table 9.2 A table with the columns ‘Modulo’ and ‘Odd number?’ to be filled in by learners to identify patterns in finding an odd or even number

Division	Divisor	Modulo	Odd number?
$1 \div 2$	2	1	✓
$2 \div 2$	2	0	
$3 \div 2$	2	1	✓
$4 \div 2$	2	0	
$5 \div 2$	2	1	✓

9.4.3 CT Perspectives Development

In the process of developing the app, learners have opportunities to develop their CT perspectives, expressing themselves and connecting with and questioning the digital world. An important issue in the digital world is the security of data transmission. To encode and decode messages in real-life protocols, two sufficiently large prime numbers are used to compose a composite number. This app enables learners to obtain an initial idea of the speed of computing when they use the app to test whether a given relatively large number is a prime number or not. Through the app development, learners cannot only express what they understand about prime and composite numbers in the coding process, but also connect these mathematical concepts with encryption technology in the digital world and to think of different ways to encrypt messages. This encourages them to learn more mathematical concepts for encryption. Learners will be better able to raise questions about the digital world and privacy issues. Are the messages that they send in the digital world safe? Are there any other methods for data encryption? Should they learn more about programming in the digital world? They will also gain a sense of programming empowerment and will feel autonomous and competent in contributing to society by developing apps that can help them learn. We hope that in the long term, with the accumulation of more programming experience, learners will develop their computational identity and regard themselves as members of the computational community.

9.5 Conclusion

This study demonstrates how primary school learners develop an in-depth understanding of composite and prime numbers by building an app using App Inventor that uses the number of factors to formulate the concepts of composite and prime numbers. Learners can develop their CT concepts and experience CT practices in a series of programming tasks, including high-level planning to find and show factors and ground-level implementation of the modularised blocks to find and show the factors and to check whether the input number is a prime or composite number. They can also experience logical thinking and gain an in-depth understanding of composite and prime numbers in the process of building and testing the app. This example shows that building an app cannot only deepen their mathematics understanding but

also develop their CT in terms of concepts, practices and perspectives. Future work is to evaluate the effectiveness of this pedagogy in classroom teaching. CT should not be limited to programming lessons, but should be developed in young learners in other subjects. It is also necessary to extend this pedagogy to other subjects such as science, language learning and visual arts.

References

- Alberta Learning. (2004). *Focus on inquiry: A teacher's guide to implementing inquiry-based learning*. Edmonton, AB: Alberta Learning, Learning and Teaching Resources Branch.
- Angeli, C., Fluck, A., Webb, M., Cox, M., Malyn-Smith, J., & Zagami, J. (2016). A K-6 computational thinking curriculum framework: Implication for teacher knowledge. *Educational Technology & Society*, 19(3), 47–57.
- Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads*, 2(1), 48–54.
- Begel, A. (1996). *LogoBlocks: A graphical programming language for interacting with the world*. Cambridge, MA: Massachusetts Institute of Technology.
- Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the computational thinking. In *Annual American educational research association meeting*. Vancouver, BC, Canada.
- Cooper, S., Dann, W., & Pausch, R. (2000). Developing algorithmic thinking with Alice. In *Proceedings of the information systems educators conference* (pp. 506–539). Philadelphia, PA: AITP.
- Eisenberg, M. (2002). Output devices, computation, and the future of mathematical crafts. *International Journal of Computers for Mathematical Learning*, 7(1), 1–44.
- Foerster, K. (2016). Integrating programming into the mathematics curriculum: Combining scratch and geometry in grades 6 and 7. In *Proceedings of the 17th annual conference on information technology education* (pp. 91–96). New York, NY: ACM.
- Grover, S., & Pea, R. (2013). Computational thinking in K-12: A review of the state of the field. *Educational Researcher*, 42(1), 38–43.
- Hambrusch, S., Hoffmann, C., Korb, J. T., Haugan, M., & Hosking, A. L. (2009). A multidisciplinary approach towards computational thinking for science majors. *ACM SIGCSE Bulletin*, 41(1), 183–187.
- Hsu, T., & Hu, H. (2017). Application of the four phases of computational thinking and integration of blocky programming in a sixth-grade mathematics course. In S. C. Kong, J. Sheldon, & K. Y. Li (Eds.), *Proceedings of international conference on computational thinking education 2017* (pp. 73–76). Hong Kong: The Education University of Hong Kong.
- Jona, K., Wilensky, U., Trouille, L., Horn, M. S., Orton, K., Weintrop, D., & Beheshti, E. (2014, January). *Embedding computational thinking in science, technology, engineering, and math (CT-STEM)*. Paper presented at the future directions in computer science education summit meeting, Orlando, FL.
- Kahn, K., Sendova, E., Sacristán, A. I., & Noss, R. (2011). Young students exploring cardinality by constructing infinite processes. *Technology, Knowledge and Learning*, 16(1), 3–34.
- Ke, F. (2014). An implementation of design-based learning through creating educational computer games: A case study on mathematics learning during design and computing. *Computers & Education*, 73(1), 26–39.
- Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*, 41(1), 51–61.
- Maletsky, E., Roby, T., Andrews, A., Bennett, J., Burton, G., Luckie, L., McLeod, J., Newman, V., ..., Scheer, J. (2004). *Harcourt math* (Grade 4, Florida Edition). Orlando, FL: Harcourt.
- Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The programming language and environment. *ACM Transactions on Computing Education*, 10(4), 16.

- Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. M. (2010). Learning computer science concepts with Scratch. In *Proceedings of the sixth international workshop on computing education research* (pp. 69–76). New York, NY: ACM.
- Morelli, R., De Lanerolle, T., Lake, P., Limardo, N., Tamotsu, E., & Uche, C. (2010). Can android app inventor bring computational thinking to K-12. *Proceeding of 42nd ACM technical symposium on computer science education* (pp. 1–6). New York, NY: ACM.
- National Council of Teachers of Mathematics. (n.d.). *Illuminations: Factorize*. <https://www.nctm.org/Classroom-Resources/Illuminations/Interactives/Factorize>.
- National Research Council. (2012). *A framework for K-12 science education: Practices, crosscutting concepts, and core ideas*. Washington, DC: The National Academies Press.
- Ploger, D., & Hecht, S. (2009). Enhancing children's conceptual understanding of mathematics through Chartworld Software. *Journal of Research in Childhood Education*, 23(3), 267–277.
- Price, T., & Barnes, T. (2015). Comparing textual and block interfaces in a novice programming environment. In *Proceedings of the eleventh annual international conference on international computing education research* (pp. 91–99). Omaha, USA: ACM.
- Repenning, A., Webb, D., & Ioannidou, A. (2010). Scalable game design and the development of a checklist for getting computational thinking into public schools. In *Proceedings of the 41st ACM technical symposium on computer science education* (pp. 265–269). New York, NY: Association for Computing Machinery.
- Salomaa, A. (1996). *Public-key cryptography*. Berlin, Germany: Springer.
- Syslo, M. M., & Kwiatkowska, A. B. (2014). Learning mathematics supported by computational thinking. Constructionism and creativity. In G. Futschek, & C. Kynigos (Eds.), *Constructionism and creativity: Proceedings of the 3rd international constructionism conference 2014* (pp. 367–377). Vienna, Austria: Austrian Computer Society.
- Wilensky, U. (1995). Paradox, programming, and learning probability: A case study in a connected mathematics framework. *Journal of Mathematical Behavior*, 14(2), 253–280.
- Wilensky, U., Brady, C., & Horn, M. (2014). Fostering computational literacy in science classrooms. *Communications of the ACM*, 57(8), 24–28.
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33–35.
- Wing, J. M. (2011). Research notebook: Computational thinking—What and why? *The link magazine*, Carnegie Mellon University, Pittsburgh. <http://link.cs.cmu.edu/article.php?a=600>.
- Wolber, D., Abelson, H., & Friedman, M. (2014). Democratizing computing with App Inventor. *GetMobile: Mobile Computing and Communications*, 18(4), 53–58.
- Wolber, D., Abelson, H., Spertus, E., & Looney, L. (2011). *App Inventor: Create your own android apps*. Sebastopol, CA: O'Reilly Media.
- Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., & Korb, J. T. (2014). Computational thinking in elementary and secondary teacher education. *ACM Transactions on Computing Education (TOCE)*, 14(1), 1–16.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.