# Integration of Programming-based Tasks into Mathematical Problem-based Learning

**Zhihao CUI[*], Oi-Lam NG & Morris S. Y. JONG**
*Department of Curriculum and Instruction, The Chinese University of Hong Kong, Hong Kong*
*cuizhihao@link.cuhk.edu.hk

**Abstract:** In this paper, we presented four mathematical domain (arithmetic, random events and counting, number theory, and geometry) and corresponding tasks designed for several problem-based programming enrichment courses for middle school students. The courses aimed to integrate computing with mathematics to enhance mathematics teaching and learning. We examined the students' learning outcomes from each mathematical domain and task from three perspectives: cognitive, behavior, and affective with qualitative data. The results suggest that there existed affordances and challenges for learning both mathematics and programming in the tasks. We also identified two possible areas that contributed to the learning outcomes.

**Keywords:** Computational thinking, programming, mathematics education, problem solving

## 1. Introduction

Teaching with modern technologies is not only a currently popular topic but also a trend of future development. The scope of modern technologies used ranges from computers, tablets, 3D printing, and VR devices, in educational contexts ranging from college to secondary and elementary schools (Jong, 2015). Using programming and computational thinking to learning mathematical concepts can be traced back to Papert (1980) and further developed by Weintrop and his colleagues (2016), who illustrate the connection between mathematics and computational thinking. In our recent design-based research (Ng & Cui, 2020), we envisioned a computationally enhanced mathematics curriculum in local primary and secondary schools by exploring students' computational thinking development when engaging in mathematical problem solving with programming. We found that students' computational concepts, mathematical concepts, as well as problem-solving practices have been supported and developed through the designed tasks. 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. In addition, students were found to experience various challenges when solving mathematical problems in programming contexts (Cui & Ng, 2021).

To this end, there is still much room for investigation into the way computing can be integrated into mathematics education, especially in the K-12 context. One of the remaining questions in this regard was in exploring specific mathematical domains or topics that are suitable for integration with computing. In this paper, we explore four mathematical domains (arithmetic, random events and counting, number theory, and geometry) for different tasks used in a series of courses on mathematical problem-based learning via programming attended by a group of middle school students. In particular, we aim to address the research question, *what are the affordances for or barriers to learning both mathematics and computing with respect to the selected mathematical domains and tasks?*

## 2. Methodology

This study employed a research design of teaching experiments with qualitative data collection and analyses. The data reported in this paper was collected from several problem-based enrichment courses (named "digital making camps") designed for upper primary and lower secondary students as part of the study. We selected Scratch, a well-known and widely used block-based programming environment, as the computing tool from which mathematical problems are solved and presented by students.

Participants ranging from fifth- to eighth-grade (aged 10 to 14) were recruited from different primary and secondary schools in Hong Kong. We triangulated qualitative data in the form of students' artifact products, video and audio records, and self-reported surveys in reporting the results of the study.

## 3. Results

We present the results by the four mathematical domains (arithmetic, random events and counting, number theory, and geometry), respectively.

The observed learning outcomes for the tasks related to ***arithmetic*** were not as satisfactory as we had expected, despite that they were thought to be relatively simple compared to problems from other domains in a mathematical sense. The most common challenge was in the use of variables. Students had difficulties understanding the meaning of the variables they created or needed to create. Indeed, some students reported that the "bank balance problem" was the most difficult among all tasks. As commented by one student, "the numbers are too big, so a human brain is nearly impossible to do it, but I don't know how a computer thinks in this program, I use three days to complete that task […]". This suggested that the student saw the affordances of computing in dealing with large numbers; however, he struggled with solving the problem from a computational perspective.

In presenting problems with ***random events and counting*** (outcome space) in a programming environment, we observed that students had diverse performances in terms of cognitive and behavioral outcomes. This was related to the mathematics behind the problem, as we had intended that students would first program to experiment with random events using computer-generated randomized outcomes, and then observe the regularities of the frequency of outcomes to generate an outcome space. In the first part of the problem, the students achieved an average to a high degree in terms of both completion and quality. In comparison, the completion rate was significantly lower in the latter part of the problem. For example, we observe that many students wrongly used the "random" function in Scratch when dealing with the later part, which showed that the students remained at the level of experimenting with random events—a precursor to learning experimental probability. Nonetheless, for those students who did solve both parts of the problem, they demonstrated strong mathematical thinking and reasoning. One student simulated the dart-throwing situation 1000 times, and by observing the experimental outcomes of the dart's landing with the area of the dartboard, he inferred that the two quantities were proportional. Other students discovered that the outcomes ought to be symmetrical about the median when obtaining the six-dice sum (with equal likelihood). Regarding behavioral outcomes, we found that students were highly engaged in exploring and discussing the dice rolling problem. This was evident by their spontaneous discussion over the possible sums and most likely six-dice sum even before they began programming. In another incident, the students were taught to make the sprites move every time the random event generated a certain outcome. During the simulation of a large sample, the students betted on which sprite would move farther like a racing game with their pairs. It can be seen that the visual affordances of Scratch in supporting dynamic random events were positive to the students' learning experience in general.

In general, the quality and completion rate was unsatisfactory in the two tasks related to ***number theory***. Most students could program some artifacts, but few could solve the problems correctly with their programs. Regarding skill acquisition, some affordances of the programming environment were worth pointing out, including the possibility to test and debug one's solution to the problem. Since students were already familiar with the property of prime numbers, the students mainly engaged in testing their programs to make their "prime number detectors" work. The students knew that they needed to use conditions ("if… then") and repeat loops ("repeat… times" or "repeat until…"); however, they struggled to combine the two codes to repeatedly check the division statements, $N \div n$, where $N$ is the given number and $n$ is its possible factors. In such an open problem, the students had trouble ensuring the correctness of the program, especially without thoroughly and systematically testing. For example, one student claimed that his program worked, but in fact, it did not. It was because he had only used even numbers to test his problem. When the instructor asked him to test an odd composite number, the program detected the number as a prime. Therefore, it seemed that this task would be meaningful for developing testing and debugging practices in mathematical problem solving.

The students were most productive in programming ***geometry***-related solutions regardless of whether their solutions were correct or not. This can be evident in students programming various types of geometric shapes beyond what was required in the problem. For example, although we only instructed the students to draw triangles and squares, many students tried to extend their programs to drawing pentagon, hexagon, heptagon, etc. In doing so, they experienced functions and parameters, which were not typical for teaching and learning geometry with paper-and-pencil but were two computational concepts relevant for the lesson. That is, they named a function, "drawing polygon" with two parameters (i.e., number of sides), and then used the functions with different parameter inputs to draw various polygons. Creativity was observed when a student conveniently drew a 360-sided polygon using his program, which he called "a circle". Besides, using different and gradient changing colors was also common among students regardless of their level of programming skills. Where the target drawings were fractal geometry, i.e., Sierpinski Triangle, those students who failed to complete the tasks did create unique fractal geometry figures while exploring the solution. Without knowing why the program would output these particular figures, they continued to being engaged in generating them through trial and error. Many considered their programmable solutions "beautiful" figures, described as "flowers", "window grille", "carpet", "black holes", etc. The students were excited to see their programmed drawings, especially when their programs worked as designed. Most students reported in the post-camp surveys that the most enjoyable moment was to see their programs worked.

## 4. Discussion and conclusion

As informed by the findings, we suggest two areas in which programming and computational problem solving can afford enrichment to mathematics learning. The first is that the problem should be stated such that either the solution or the solution process is not immediately known. From this perspective, some mathematics-related problems were more effective when presented in a programming context. For example, the solution process (e.g., strategies for counting to 21), the solution itself (e.g., the prime detector for large number), or both (e.g., experimental probability and fractal geometry) were not known to the students immediately when it was first presented. This element of unknown provided opportunities for students to explore and inquire about new concepts, both in mathematics and programming. Secondly, computing with screen-based artifacts afforded dynamic visual representation and immediate feedback (such as movement of the sprite, outputting a certain number, a figure to be drawn, etc.), which has been shown to highly engage the students who participated in this study. As such, the students were more likely to continue regardless of the complexity and difficulty.

To conclude, this paper described and discussed some affordances and barriers to teaching and learning mathematics in computationally enhanced ways, drawing on selected mathematical domains and tasks. More research is warranted to further designing and studying learning materials for computationally enhanced mathematical teaching and learning.

## References

Cui, Z., & Ng, O. (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.

Jong, M. S. (2015). Does online game-based learning work in formal education at school? A case study of VISOLE. *Curriculum Journal, 26*(2), 249-267.

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.

Ng, O., & Cui, Z. (2020). Examining primary students' mathematical problem-solving in a programming context: towards computationally enhanced mathematics education. *ZDM – Mathematics Education, 53*(4), 847-860.

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

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.