# Students and Teachers Learning to Code: Two Worlds, or One?

Andrijana Burazin
University of Toronto Mississauga
a.burazin@utoronto.ca

Miroslav Lovric
McMaster University
lovric@mcmaster.ca

*The authors have recently integrated computer labs into their first-year courses. At the start of the semester, students showed high levels of anxiety, claiming that they have not done coding in high school. This obviously being a high school to university transitional issue, we designed activities to help high school teachers learn about coding as well. Our hope is that, once they remove their personal barriers, teachers might be better motivated to engage with coding activities in their math classes. Being able to compare, we are interested in possible differences in views, experiences, and actual learning when two populations (first year university students and high school teachers), with no or little prior experience, learn to code in Python? Our preliminary results point to the fact that the two groups show some differences, which will inform how we modify our teaching and learning support for them. However, these differences cannot be confidently attributed to the young-adult divide only.*

## Introduction

The authors have recently introduced coding activities ("computer labs") in Python into their first-year courses (calculus for the physical sciences at University of Toronto Mississauga (UTM) and calculus for the life sciences at McMaster University; both universities are located in Southern Ontario). Their reasons for doing so were to offer experiences to students that otherwise would not be there, or not as dominant (Lovric, 2018). For instance, even a simple code could offer creative opportunities for dynamic modelling. Modification of a computer code, for instance by changing the value of a parameter, generates an immediate mathematical reaction (trying to predict what will happen when the value of a parameter is changed is a good learning experience).

Furthermore, coding enhances one's sense of abstraction, and yet the objects involved have a "tangible feel," having been created by a user. To code a function, a user needs to know what the input is (i.e., what kinds of objects the function is going to operate on), what the rule for the transformation is, and what kinds of objects this rule generates as output. Thus, what students often perceive as abstract objects (domain and range, and the rule that defines a function) indeed become "real," and more "tangible."

The "low floor, high ceiling" approach that coding promotes invites students with little or no experience to engage; once they learn, they can move beyond the problem they are solving. It takes very little time to learn to code a simple repeating process (loop), and to see the connections between the code and the output (Where does the loop start, and where does it end? How many times does my computer repeat the loop, and what happens with each pass through the loop?). Then, through a sequence of “what if” investigations (What if I change this value? What if I add another line to the loop? What if I insert a loop into a loop?), and with the benefit of the immediate feedback, a beginner learner moves fairly quickly from elementary to more complex coding situations.

We should keep in mind a psychological phenomenon, sometimes referred to as the “fear of new” (as experienced, for instance, by our first-year calculus students when we discuss integration, which is a new topic for many). Upon learning that their course will involve coding labs, a large number of students showed high levels of anxiety, claiming that they will not be able to do it as they have not done any coding in high school (which is indeed true). In our province, the curriculum for *Computer Studies* – whose main component is coding – has been in place since 2008. However, as very few teachers feel (or are) qualified to teach, most schools do not offer classes in coding. Anticipating this, we prepared a gentle introduction to coding, both in written form, and as a video. Asking ourselves what else we could do, we decided to involve high school teachers (adults!), by helping them to learn to code as well. By offering teachers an opportunity to learn about coding, we believe we can remove barriers that they feel are massive and require large amounts of time and energy to surmount. Once this happens, teachers might be better motivated to introduce coding activities in their high school math classes.

Having two distinct populations – undergraduate students fresh out of high school, and high school grade 12 teachers – presented us with an opportunity to contrast their experiences as they learn to code. We focused on the following two questions:

- What are the differences in views, experiences, and learning when two different populations (first year university students and high school teachers), with no or little prior experience in coding, learn to code in Python?

- What kind of professional adjustment is needed to make mathematics teachers more comfortable in a high school coding classroom?

## Method

As there is little research on coding in classroom settings (Grover & Pea, 2013; Lye & Koh, 2014), we collected our own data, including: students’ replies to coding activities (including questions about their experiences); teaching evaluations; interviews with a selected group of students; teacher feedback following the Computing Workshop at UTM in 2018 and 2019; and informal conversations with teachers and high school students. We have barely started to analyze this data, using mixed methods (Creswell, 2014) and the case study approaches.

Of particular interest to us are the “Aha!” moments, or the moments of discovery, when a learner realizes that they “got it,” after what seemed to be a long and unproductive effort. “Aha!” moments have been recognized as important in mathematics because of their “transformative effect on ‘resistant’ students’ affective domains, creating positive beliefs and attitudes about mathematics as well as their abilities to do mathematics” (Liljedahl, 2005). We believe that their importance and benefits naturally transfer to learners of any age who are engaged with computer programming.

## Findings

As we are only starting to obtain results, we cannot offer a comprehensive analysis. However, preliminary results point to the fact that the two groups (first-year students and high school teachers) show some differences. Does this mean that adults learn to code in ways that are different from young(er) people's approaches? It is hard to obtain a definitive answer, and qualify it along some kind of distinction between young and adult. We believe that some differences are correlated to the experiences of learning and working with mathematics (and possibly with other school learning experiences).

We noticed that teachers prefer written instructions (e.g., explanations about what a particular command does embedded into Python code as comments), whereas students prefer watching a video of an instructor explaining steps in coding. (Of course, every generalization of this sort could be easily challenged.)

Most teachers, as well as some students, used pencil and paper to make diagrams and notes as they worked on programming tasks. Working with computer code on screen and working with code on a piece of paper are two different cognitive experiences. In a pencil-and-paper thinking about an algorithm, learners have an opportunity to visualize the structure of a program. For instance, thinking about a nested loop (loop within a loop), they realize that the requirement that the loops must be nested has a useful visual representation - the arcs which join the start of each loop to its end are not supposed to cross! A conditional statement can visualized as a fork. (Of course, visual imagery can be (and is) realized on a computer as well—some programs use visual interface, which replaces typing the code.)

As well, faced with having to solve a *mathematics* problem using coding, most students start working on screen right away; very few use paper and pencil to try some algebra, draw diagrams, or otherwise reflect on the strategy they will use to approach the problem (Lovric, 2018; Mamolo & Lovric, 2018). Teachers are more reluctant to just type in the code, hit the 'run' button and see how it works. Students harvest the benefits of immediate feedback by entering "half-baked code" and seeing what the computer will tell them.

In a task given in a university class, students had to investigate Collatz Conjecture. (This conjecture concerns a sequence of positive numbers defined recursively as follows: start with a number; it if it even divide it by 2, and if it is odd, multiply it by 3 and add 1. For example: 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1. Collatz Conjecture states that no matter what number the sequence starts with, it will always end with 1.) The code which students created was able to generate sequences as the one given here - but nothing more than that. An "offline" investigation has a potential of enriching this experience. By "playing" with Collatz Conjecture on a piece of paper a student discovered a pattern: an odd number is always followed by an even number (thus, a sequence of iterations cannot have two or more odd numbers in a row).

To address teaching of coding in high school, we must accept the fact that classroom instruction with a teacher at the front needs a reconstruction. Students cannot be passive observers; instead, they have to be actively engaged in creating the computer code, its testing, interpreting results, and in making adjustments and modifications. Knowledge and skills are gained and improved through *practice*. Teaching coding requires a type of teacher who is comfortable with facilitating experimentation and trial-and-error, encouraging detours, and fostering discovery. In contrast, "Very often, when we teach math, we 'forget' how results, theorems, or definitions have been arrived at.  We present a proof, but rarely talk about how it was constructed, and in particular, we don't talk about failed attempts!" (Mamolo & Lovric, 2018)

Another important obstacle to teaching is the content. Our experiences echo the recent report from Sweden: "It is acknowledged that teachers do not often have the content expertise or confidence in teaching 'new' topics as they are assigned to curricula" (Hartell, Doyle, and Gumaelius, 2019). A body of research suggests that teachers' self-efficacy (about a particular topic, or area of mathematics) is positively correlated with their classroom work, for instance in their ability to create effective learning opportunities for their students. Here, we recognize the importance of the "low floor" affordance to coding, which, with early successes, can stimulate teachers to actively engage with coding.

## Conclusions

Common to both groups (first-year students and high school teachers) was the initial anxiety of facing something new, which, however, weakened fairly quickly. Although the two groups (might) show some differences in their approaches to coding, we do not believe that they present an obstacle to an effective high school coding classroom instruction. Perhaps the best suggestion to teachers who are considering to teach coding is to be like their students: play, experiment, be creative and explore many directions, and – do not be afraid to hit 'run'!

## References

Creswell, J. W. (2014). *Research Design.* Thousand Oaks, CA: Sage Publications.

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

Hartell, E., Doyle, A., and Gumaelius, L. (2019). Teachers' attitudes towards teaching programming in Swedish Technology education. Conference presentations. Available at https://www.researchgate.net/publication/334326760_Teachers'_attitudes_towards_teaching_programming_in_Swedish_Technology_education/citation/download

Liljedahl, P. G. (2005). Mathematical discovery and *affect*: the *effect* of AHA! experiences on undergraduate mathematics students. *International Journal of Mathematical Education in Science and Technology*, 36 (2-3), pp. 219-234.

Lovric, M. (2018). Programming and Mathematics in an Upper-Level University Problem-Solving Course. *PRIMUS*, 28(7), 683-698. doi: 10.1080/10511970.2017.1403524

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

Mamolo A. & Lovric, M. (2018). Computational Thinking in and for Undergraduate Mathematics: Perspectives of a Mathematician. In A. Weinberg, C. Rasmussen, J. Rabin, M. Wawro, & S. Brown (Eds.), *Proceedings of the 20th Annual Conference on Research in Undergraduate Mathematics Education* (pp. 723-731). San Diego, California: SIGMAA.