


Evaluating Students' Experiences of a Weekly "Hour of Code": Cookies or Cake?

Marguerite Koole, University of Saskatchewan, Canada*

 <https://orcid.org/0000-0002-0041-5615>

Kaleigh Elian, Learning Disabilities Association of Saskatchewan, Canada

ABSTRACT

In the winter semester of 2020 during a multimedia design and production class for pre-service teachers, the students were introduced to basic computer coding concepts such as variables, conditional statements, various expressions, logic, and syntax. For their final project, the students were asked to create an interactive instructional app using MIT App Inventor for their own future students in their teaching subjects (such as social studies, mathematics, science, and language arts). They were expected to integrate technical skills and knowledge of interface design, instructional design, and pedagogical strategies. The instructors examined exit tickets submitted at the end of each hour-of-code lesson and course evaluations at the end of the semester for evidence of threshold concepts, students' learning experiences, and motivation. This brief qualitative study provides a description of the course, coding and computational thinking processes, and the student evaluations. The paper concludes with commentary on lessons learned for teaching coding to pre-service teacher candidates.

KEYWORDS

Coding, Computational Literacy, Computational Thinking, Hour of Code, MIT App Inventor, Mobile Learning, Pre-Service Teachers

INTRODUCTION

In Canada and other countries around the world there has been increased pressure from governments and corporate interest groups to increase numbers of graduates in science, technology, engineering, and mathematics (STEM) subjects at all levels of the education system (Kafai & Proctor, 2021; Koole & Squires, 2020). Yet, many students in grade school are afraid to learn coding and often lose interest in STEM subjects (Laffee, 2017). To address this situation, schools are looking towards teachers to fill the gap; however, there evidence that teachers still generally lack training in computer programming—including the use of visual tools, current methods, and problem-solving strategies for coding (Dağ, 2019). Furthermore, many teachers lack the confidence, to select age-appropriate pedagogical approaches and instructional resources (Barr & Stephenson, 2011; Dağ, 2019). This is

DOI: 10.4018/IJMBL.304458

*Corresponding Author

This article, originally published under IGI Global's copyright on July 15, 2022 will proceed with publication as an Open Access article starting on March 19, 2024 in the gold Open Access journal, International Journal of Mobile and Blended Learning (IJMBL) (converted to gold Open Access January 1, 2023) and will be distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/4.0/>) which permits unrestricted use, distribution, and production in any medium, provided the author of the original work and original publication source are properly credited.

compounded by the attitudes of some teachers, albeit already overburdened by the daily demands of their teaching responsibilities, who feel that the inclusion of computer programming is just an extra activity on top of an already overlaid curriculum rather than an important way to foster valuable analysis and thinking processes in their students (Laffee, 2017). Kafai and Proctor (2021) argue that there is a tremendous need to include computing education in teacher education programs and to “develop a better understanding of what content knowledge and skills K-12 teachers actually need to have, what equitable teaching practices look [like], and how teachers can address critical dimensions of [computer science]” (Kafai & Proctor, 2021, p. 5).

With these challenges in mind, in January 2020, a group of 22 teacher-education students were offered a 4th year course called Multimedia Design and Production at the College of Education, University of Saskatchewan. As part of the course, the teacher candidates learned about designing, editing, and creating text, images, audio, and video for educational purposes. In this course, the learners were asked to create a small mobile application using MIT App Inventor (MIT App Inventor, n.d.). The students were asked to complete a small exit ticket (brief post-lesson student evaluation survey) after each weekly, hour-of-code-style programming lesson. This paper offers a brief discussion of the benefits of computer coding for students and teacher candidates. The paper then describes the context, the course, the instructional techniques, and lessons learned. There were three main goals: 1) to observe learner engagement, 2) to understand pre-service teachers’ need for skills to teach their own future students, and 3) to explore possible threshold concepts in the acquisition of computer-programming and problem-solving skills.

CODING AND LEARNING PROCESSES

Why coding? In contrast to the neoliberal argument that STEM subjects and computer programming are imperative for countries to gain economic advantage, it is unlikely that most learners will grow up to be computer programmers (Kafai & Proctor, 2021). However, there are tangible and intangible benefits that support teaching computer programming and computational thinking in Kindergarten to Grade 12 (K-12) (Ateşkan & Hart, 2021; Barr & Stephenson, 2011; Grover & Pea, 2013). For example, some knowledge of coding can help foster important thinking skills. Furthermore, as digital technologies are embedded into our day-to-day lives, lurking in our automobiles, household appliances, televisions, and even on our bodies in the form of smart devices and wearables, knowledge of programming can help to demystify how every day, digital tools and services function and articulate.

Although coding once might have been a solitary activity, it is now often a “shared social practice” (Kafai, 2016, p. 27) in which complex problems are tackled through collaborative and cooperative behaviours (Doleck et al., 2017). Indeed, Kafai and Proctor (2021) suggest that an important aspect of “authentic learning practices . . . means becoming a member of a community of practice with shared goals and values” (p. 3). In addition to developing social skills, students can also develop persistence and learn how to cope with knotty, open-ended challenges. While engaging in planning and design activities, they also need to think logically and sequentially. Computer programming might also be considered a form of literacy in which learners engage with a “symbolic representational system” replete with a vocabulary, grammar, and syntax that learners come to understand, produce, communicate, and share (Bers, 2019, p. 504). Kafai and Proctor (2021) define computer literacies as “a set of practices situated in a sociocultural context which utilize external computational media to support cognition and communication [and which] encompass phenomena at scales from the individual to the societal, as well as connections between these phenomena and the media which supports and shapes them” (pp. 3-4). Recent studies are starting to show other interesting benefits for kids learning to code. For example, a study conducted by Arfe, Vardanega, and Roconi (2020), showed that coding activities with 5- to 6-year-old children can enhance cognitive development in planning and inhibition skills. The authors concluded, “the children exposed to Coding activities increased their planning time and accuracy and decreased the rate of inhibition errors” (p. 14). Therefore, the goal in teaching

coding is not necessarily to create computer scientists, but rather to use computer tools and concepts to foster creative problem solving, criticality, communication, personal expression, and collaboration (Barr & Stephenson, 2011; Kafai & Proctor, 2021).

Attributed to Wing (2006), the term computational thinking refers to “creative thinking, algorithmic thinking, critical thinking, problem-solving, cooperative learning, and communication skills” (Dağ, 2019, p. 281). Computational thinking refers to “a kind of problem solving and a readiness to move between different levels of abstraction and decomposition, transformation and simulation” (Pöllänen & Pöllänen K, 2019, p. 15). Arfe, Vardanega, and Roconi (2020) break computational thinking into four parts: 1) analyzing the problem, 2) reducing/decomposing the problem into smaller units of analysis, 3) creating a sequence of steps towards a solution (i.e., an algorithm), and 4) a verifying the solution. Teaching computational thinking and programming can be challenging at any age or educational level. For this reason, it is important to consider pedagogical strategies carefully. Some educators recommend starting with real world problems, learning lower-level skills (i.e., single lines of code) first and slowly gaining in complexity (i.e., moving towards nested structures or conditional syntax). Others use a spiral approach starting with concrete objects, engaging in computational thinking about the objects, then moving into the creation of algorithms, and then finally, coding (Pöllänen & Pöllänen, 2019).

Because coding and computational thinking are now recognized as beneficial for children, it is important to better understand how to support in-service and pre-service teachers in developing instruction for their classes. Menekse (2015) conducted a systematic review of in-service teachers’ training in computer science of which they found only 21 articles with only seven that mentioned pre-service teachers. Since then, more studies are starting to emerge and are now providing insights into the needs and challenges teachers face. Zha, Moore, and Gaston (2020), for example, conducted a study with 15 pre-service teachers for whom a computational thinking/coding module was added into an educational technology course. Using a block-style programming interface called Hopscotch, the pre-service teachers learned coding concepts such as “sequence, conditional logic, and algorithms” (p. 20). Of the 15 study participants, only one had some limited prior knowledge of programming. The researchers noted that early in the module, the participants “expressed confusions, anxiety, and resistance, while a couple of them showed curiosity at the beginning stage. As they paced along with the step-by-step instructions in the online module, their resistance and anxiety were reduced. They became interested in learning and practicing the coding activities” (p. 24). In another small qualitative study, some pre-service teachers had the opportunity to learn to program with Scratch (block-style) and Python (Gok & Kwon, 2020). Of the 12 participants observed, four semi-structured interviews were conducted. The researchers noted that the participants struggled with understanding programming concepts, algorithmic thinking, syntax, and debugging. Gok and Kwon also noted the ways in which the participants sought solutions: increased planning, asking for assistance from tutors, guessing potential solutions, and testing their attempts (i.e., running the programs). Interestingly, they noted that the teachers appreciated learning through sample code provided by the instructors and were most likely to seek support from their peers before asking their instructors.

There are many potential platforms for teaching coding in primary school and for training pre-service teachers including Scratch, Creative Hybrid Environment for Robotics Programming (CHERP), JEM Inventor, and directly coding in Python and other languages (Arfé et al., 2020; Bers, 2019). There are organizations that specialize in the teaching and learning of code. The “hour of code” movement and website, Code.org (Code.org, 2021), was founded by Hadi Partovi and Ali Partovi. In 2013, they launched an hour-long course to introduce grade-school students to computer science. From that time, Code.org has provided computer courses and has become involved in teacher training. In the multimedia course discussed in this paper, the instructors designed their own lessons and teaching resources.

So, why ask pre-service teachers to design a mobile application? Creating mobile applications offers a concrete experience in that the learners can create an interactive app that can be

‘physically’ downloaded onto an actual phone and shared with classmates, friends, and family inside and outside of the classroom. The mobile environment also offers unique constraints that complement instructional design considerations discussed in the multimedia class such as file size, layout, and instructional strategies.

METHODOLOGY

This section begins by describing the study goals, context, and the coding platform. It then offers a brief description and depiction of the procedures for collecting the data.

Research Goals: Learner Engagement, Need, and Threshold Concepts

This small qualitative study used exit-ticket-style evaluations of the hour of code component of a class on multimedia. On these exit tickets, the students wrote answers to the questions: What did you learn today? What did you enjoy today? Did you have an ah-ha moment? Was anything confusing today? These questions correspond to the goals of the study. To reiterate, the first goal was to observe the level of learner engagement as they learned to code in MIT App Inventor. The instructors hoped that the ability to create, test, share, and ‘play’ with apps on real devices both inside and outside the classroom would add a sense of tangibility, motivation, ownership, and fun. The second goal was to get a sense of how the teaching and learning of coding might support students as soon-to-be teachers. Grover and Pea (2013) recommend “computing as a medium for teaching other subjects—dovetailing the introduction of [computational thinking] at K-12 with transfer of problem-solving skills in other domains” (p. 42). In this course, the coding tasks complemented the multimedia content and the instructional design content. Lastly, the instructors wanted to explore whether there were any threshold concepts or special needs for pre-service teachers who might wish to teach coding to children in the Canadian K-12 system. According to Cousin (2010), there are five characteristics of threshold concepts:

1. They are significantly transformative in how students perceive a topic or how they see themselves.
2. They are usually irreversible.
3. They help learners understand relationships between different concepts.
4. They may help learners identify boundaries between disciplines.
5. They are initially troublesome because they are complex, tacitly acquired, or counterintuitive.

Cousin’s criteria were used to detect threshold experiences in the students’ exit-ticket responses and final course evaluations.

The Context and Course

The Multimedia Design and Production course is normally taught for 3 hours a week over 13 weeks. It is an elective for undergraduate students enrolled in the Bachelor of Education program at the College of Education. The course was held in a computer laboratory on Mac computers with 22 students enrolled. The course covered elements of visual design, instructional design, and various technical skills to produce interactive instruction and multi-mediated forms of learning resources. The students needed to learn how to create and edit specifically for mobile access. For example, lessons focused on the use and design of text, images, audio, and video for optimal access on mobile output while also considering low bandwidth conditions (which is still a reality in some parts of Canada). The students also needed to consider how to design their apps for effective learning—presuming they would use them with their own students after graduation. To prepare for the final project, the students received one hour-of-code lesson for each week for eight weeks. During the final three classes after the eight-week series of hour-of-code lessons, students could ask the instructors for assistance, hold group meetings, and work on their final projects. (Note: two of the final classes were remote because of the pandemic; therefore, collaboration was limited.)

The instructors varied their instructional style from week to week. Early in the semester, the instructors would walk the students step-by-step through the process of programming a simple app comprising simple code snippets. The programming tasks slowly became larger and more complex. At times, the students were asked to read and trace code. In code tracing, a learner visualizes or simulates how a block of code works. Some studies suggest that learners should first engage in tracing code before they can learn to write code (Harrington & Cheng, 2018; Hertz & Jump, 2013). At other times, the learners were organized into small groups and were given a type of jigsaw activity (also known as Parsons problems (Grey, 2021)) in which the learners would discuss the function of code snippets, organize them into blocks, and ultimately type them into the MIT App Inventor interface for testing. In one lesson, the students were given an incomplete app in which the code was jumbled. The students were asked to upload the incomplete app to MIT App Inventor and unjumble the code to make it work. Studies have found that offering students concrete program comprehension tasks can assist learners in forming mental models while they engage in programming tasks (Izu et al., 2019). (A weekly breakdown of the lessons and strategies is presented in the data collection section below.)

Throughout the semester, the instructors incorporated a sociocultural aspect through group work in order to foster a community of practice within the class. As the semester progressed, the instructors encouraged increasing interaction and autonomy among small groups in which they would engage in programming tasks in dyads (two people) or triads (three people). ‘Pair programming’ has been shown to benefit female learners and may help to scaffold learners towards greater autonomy and increased persistence (Hanks et al., 2011; Werner & Denning, 2009). The final project required the students to work in small groups and integrate the multimedia skills they acquired during the semester into a mobile application in MIT App Inventor.

The Coding Platform

MIT App Inventor (n.d.-a) was deemed appropriate as the main platform because it allows block-style coding (Figure 1) which is currently popular in Kindergarten to Grade 12 (K-12) settings. Similar to Scratch (another programming interface for children), MIT App Inventor offers scripts and sprites. ‘Scripts’ refers to the code that appears in the blocks (children can also view the code without blocks). The ‘sprites’ refer to the images that appear on the screen (Chandra & Lloyd, 2020).

MIT’s App Inventor offers tools for layout, integration of media, drawing and animation, sensors, telephony, Bluetooth connectivity, databasing, and even integration with LEGO® MINDSTORMS®. Although the creators of App Inventor indicated they were developing an iOS-friendly version, the apps could only be used on Android devices at the time this course was offered. For this reason, the instructors acquired five Android devices connected to the university WIFI. The students could also test their creations using the MIT App Inventor emulator installed on the laboratory computers. This emulator is shown in Figure 2, showing the output of the current code. This same code is shown running on a physical Android device in Figure 3.

Figure 1. Block-style coding

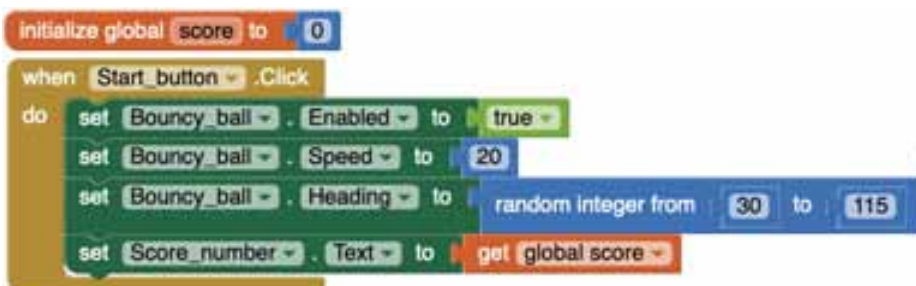


Figure 2. MIT App Inventor interface (sample application designed by S. Thuringer 2020 shared with permission)

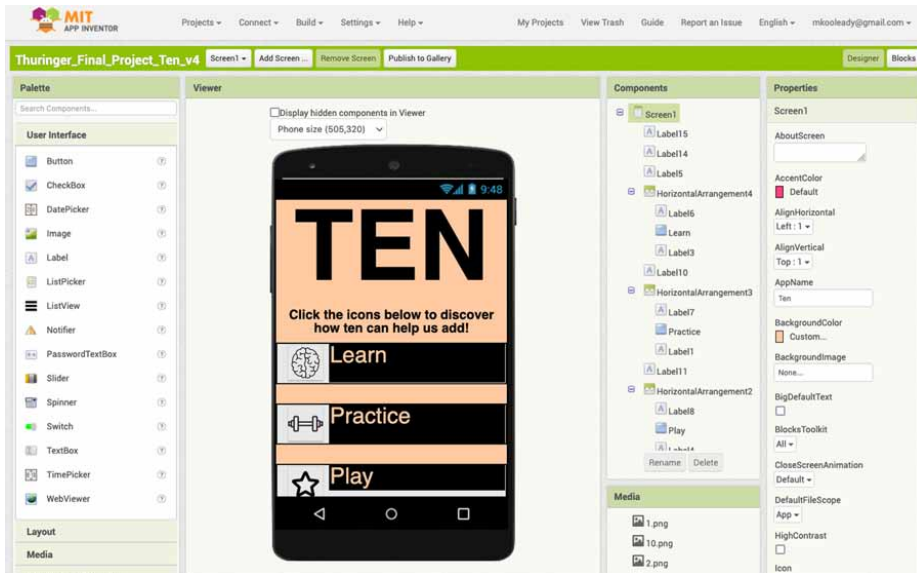


Figure 3. Sample of a student application on an Android device (sample application designed by S. Thuringer 2020 shared with permission)

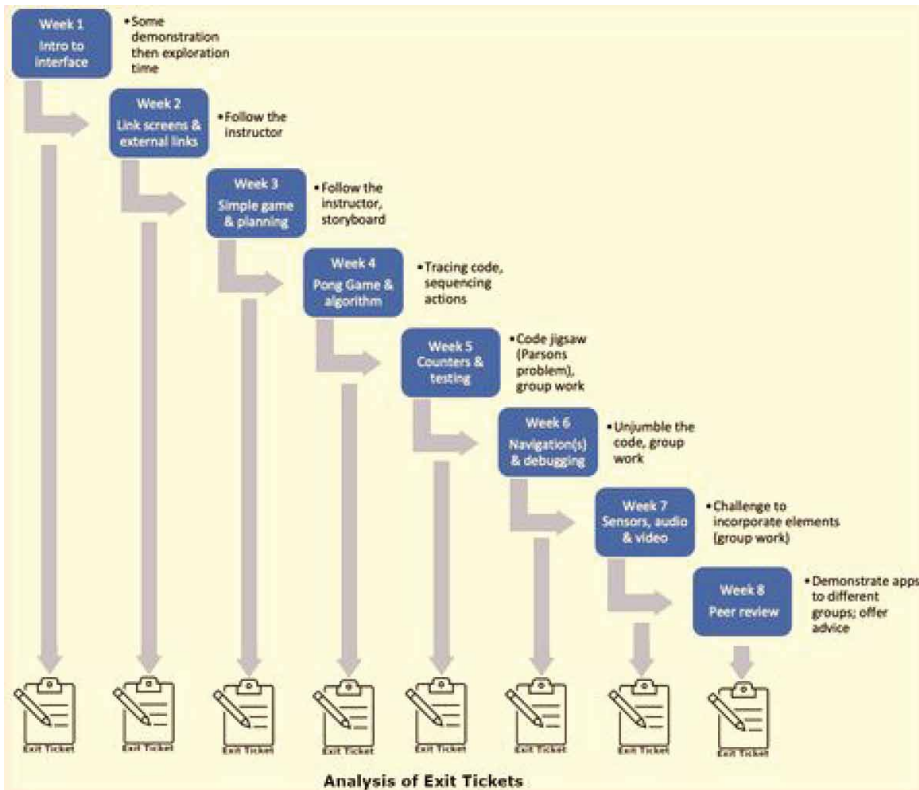


Data Collection

Each week for eight weeks of the thirteen-week semester, the instructors facilitated a lesson on how to program with MIT App inventor. Figure 4 outlines the weekly lessons, topics, and pedagogical strategies.

In the first week, the instructors introduced the MIT App Inventor interface and allowed the students free time to explore the interface for the last half hour of class. During the second week, the instructors directly demonstrated how to create new screens, how to link them, and how to link to external web pages; the students followed along clicking when they were asked to click. In week three, the students again clicked along with the instructors to learn how to incorporate some simple actions (building skills towards a ‘pong’ game). At the end of class, the students were asked to think about how to plan a more elaborate pong game (at approximately this point in the course, the students were encouraged to begin storyboarding and structuring their final, group projects). In week four, the students were given more elaborate code for a pong game in which they engaged in code tracing and considered what should happen first, second, and so forth. In week five, the students were asked to add a counter to their pong game. Working in small groups, they were asked to arrange paper pieces (a Parsons problem) with block coding in order to determine how to code the counter. In weeks six and seven, the students were asked to unjumble code as well as determine how to add mobile sensor features, add audio, and add video. In week eight, the students had an opportunity to show each other their in-progress final projects and offer each other assistance in solving coding problems. After each hour-of-code lesson, the students were asked to complete an online exit ticket with four

Figure 4. Lessons and collection of exit tickets



questions: What did you learn today? What did you enjoy today? Did you have an ah-ha moment? Was anything confusing today?

STUDENT EVALUATION RESULTS

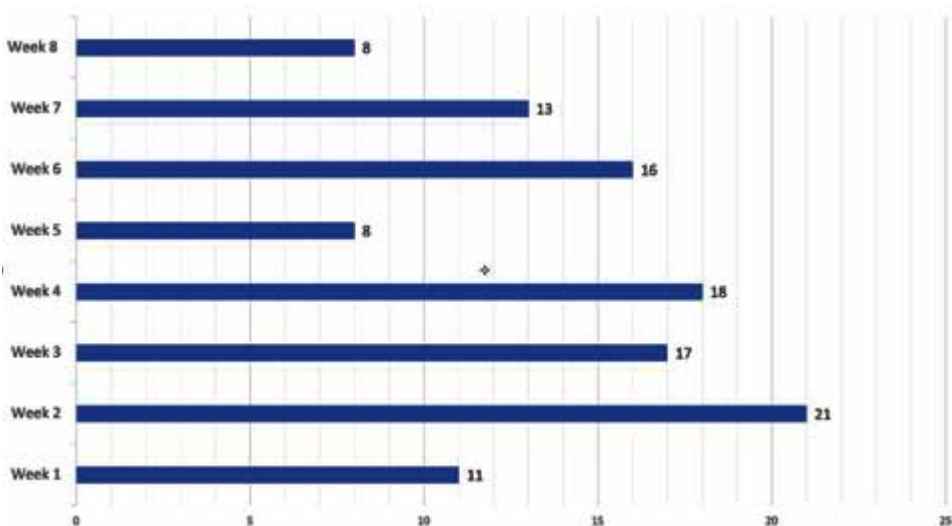
Students were asked to complete the exit tickets at the end of each class, but completion was voluntary. The number of exit tickets submitted each of the eight weeks varied from 8 to 21 out of a possible 22 (Figure 5). Identities of the participants were removed to preserve confidentiality.

The results presented here are organized according to the four questions on the exit-ticket evaluation questions. In this way, the instructors could monitor the students' perceptions of what they learned, what they liked, if they had experienced any threshold concepts, and which aspects of instruction might have needed improvement. For this small study, we examined the students' answers looking for patterns in their responses and clues as to how we might improve our pedagogical strategies.

What Did You Learn Today?

After the first hour of code, comments ranged from confused and concerned (“I don’t understand the program or what we are really supposed to be doing, or how I am supposed to be able to create an app myself in this short period of time”) to being intrigued by the block-style coding and designer interfaces, to having successfully created their first small application (“Using the tutorial, I learned how to code the Hello Codi [activity]”). Some students indicated they already had experience with block-style coding through Scratch and other systems. As the semester progressed, the students listed the different skills and MIT App Inventor tools they learned (i.e., creating hyperlinks, creating menus, making sprites move on screen, etc.) and commented on the programming techniques taught (i.e., programming a counter, creating variables, creating lists, and using sensor components). Classroom interactions resulted in some unexpected successes. In one case, there was evidence that some students picked up on behaviours modeled by the instructors. For example, when asked a question during a demonstration in class, an instructor searched the MIT App Inventor forum for an answer. One student commented on the exit ticket that they had learned “. . . problem solving: combing through forums to find a solution”. Another interesting comment involved the importance of debugging which the instructors also modeled by testing on an emulator, correcting code, and retesting on the emulator.

Figure 5. Number of exit tickets submitted



What Did You Enjoy Today?

After the first hour of code, many students indicated that they enjoyed having free time to explore the interface at their own pace. Two other students commented that they liked building their first app by following the instructors step-by-step and then testing their code on the emulator (which suggests that the seeing their app on an actual mobile device offered a sense of concreteness, possibly increasing motivation or enjoyment). After the second hour of code, students commented that they were excited to create hyperlinks and learn how to link screens together. One suggested that it was fun “seeing things coming together” into an actual app. At the end of the third hour of code, learners had the opportunity to show each other their first assignment. Multiple students commented that they enjoyed “looking at other peers’ apps”, “getting constructive feedback” for their own apps, and working with a partner to improve their coding. For the next class, the students started creating a pong game. One commented, “The hour of code that was the most exciting was when we made the pong ball bounce off the paddle.”

Surprisingly, the paper coding activity (jigsaw puzzle-style) during the fifth class was a hit: “I really enjoyed the group work and trying to problem solve and create [put into the correct order] blocks [code] for the score aspect of the pong game.” By the sixth hour of code, the instructors decided to change the way they scaffolded the lessons; a student commented, “I really liked downloading the app and having all of the coding in pieces. That way you were able to demonstrate the coding without simply walking us through it, and it gave us the support that some of us needed while also letting us move at our own pace. I thought that was definitely an amazing idea.” The students also responded positively when encouraged to customize the code for the pong game by changing the buttons, adding sound effects, and even changing behaviours. By the eighth hour of code students were incorporating light sensors and testing their apps on real Android phones. Several students indicated that they enjoyed the increased autonomy: “I really liked being given free rein to do it” and “I like the opportunity to figure things out for myself as always.” There was still some evidence of frustration which needs additional analysis: “[I enjoyed it] when my app actually worked. That’s a bit rare in this class [for me].”

Did You Have an “Ah-ha!” Moment?

Early in the semester, the main ah-ha moments were related to the skills being taught such as how to add sound, how to import media, understanding how the blocks and design interface elements articulated, how the MIT App Inventor components list works, how to use the emulator, setting up x and y axes, the importance of spelling variable names, and so on. Really, many of the comments were reflections on the skills taught rather than indicators of threshold concepts. Indeed, one student astutely noted at the end of the fourth session, “At this point it’s hard to have an “ah-ha” moment since we are still primarily following your tutorial. This will change once we have a firm understanding of the program and can discover on our own.” There were some comments throughout the semester of students realizing that they “could do it” (suggesting increased sense of confidence).

Already towards the middle of the course, students began to comment on a shift in their autonomy: “My ah-ha moment happened on the weekend when I was planning and designing my pages. I was able to watch a YouTube video and follow the steps and program my blocks. Then with the professor’s help and guidance I was able to work out all the kinks and have a fully functioning app.” In week six, one learner noted a shift in agency: “My ah-ha moment is when I shared my new knowledge with the whole class. Being able to teach others has helped me learn and retain the new in-formation.” But the most exciting comment was left at the end of the eighth class:

I finally had an “ah-ha” moment!!!!!!!!!! It took the whole semester but finally I am starting to understand how to create the code on the MIT app! I was trying to create a game and I wanted a part of it to do something different from the tutorial I was following, and I was able to do it!!!! I think I

wanted a sprite to react differently or something like that. So, that was really exciting since I have felt very confused about how to create the code myself. . . It was kind of like my brain just opened and was like here is all of the missing links and confusion about code and now it all (or most) makes sense, it was very odd, but woohoo!!!!!! So, woohoo for an ah-ha!!!!!!!

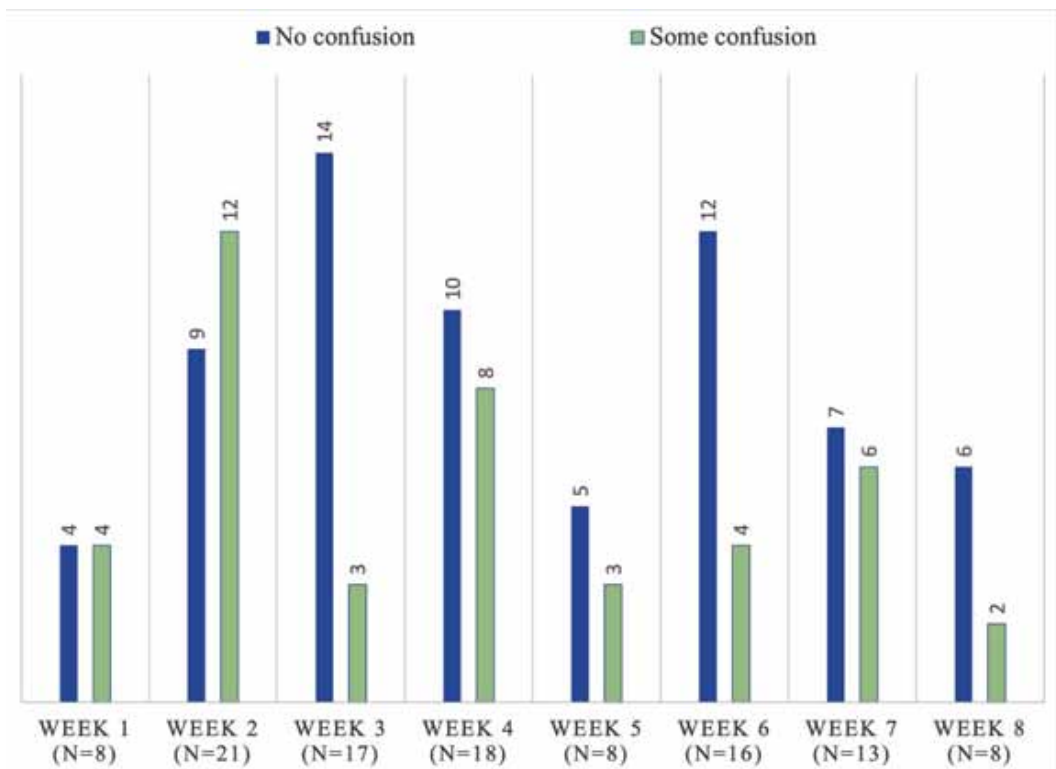
Please Describe What Was Confusing

For this category, the instructors examined students' comments to gain a sense of the degree to which the learners felt confused at the end of each lesson. If students left a question or expressed concern or anxiety, it was marked as "some confusion" (Figure 6). It is important to remember that the complexity of the activities increased as the semester progressed.

For most of the students, the interface and blocks-style were new for them, but they expected that it would become clearer as the semester progressed: "With more practice, I am sure that I will get the hang of it." Some students indicated they missed a step when following along, which suggests that more repetition or increased opportunities for one-on-one tuition would be helpful. By the end of the fourth lesson, one learner expressed a desire to have more autonomy to figure out the coding on their own and added, "I feel like we need a few people to volunteer to be 'demonstrators' and maybe do a showcase of cool stuff we figured out how to do and how we got there." This comment suggests an excellent way to motivate students who might be 'ahead' of the class in some areas and, in this way, bring the rest of the class along with them.

By the end of week eight, some students began posting comments on how they would teach coding in their classes: "If I was [substitute teaching] I would teach the students and demonstrate how to

Figure 6. Expressions of confusion



design and code a simple game. It would be an easy code to create and teach because I have already done it, and the students would be amazed by creating an interactive game.” Another commented:

I liked using the [paper] puzzle pieces for the coding. It allowed you to talk about what the pieces are used for which actually helped me understand the coding which I usually don't at [the usual class] pace. If I was [substitute teaching] for a computer science teacher, I would explain that coding is a lot like baking. You take a lot of individual pieces, and you have to combine them in a certain way in order to get your desired result. If you combine the same ingredients one way, you'll end up with a cake, and if you do it another way you'll end up with cookies.

And, yet another commented, “In the hour of code, I think it would work well to do what we did last week and this week together. First, show the app on the Android phone (like last week) and then hand out the scrambled code for us to figure out (like this week).”

DISCUSSION

Responses to the question about what they learned during a particular hour-of-code lesson showed little deviation in terms of the instructors' understanding of the teaching objectives and in-class activities. This suggests that the lesson outcomes were generally achieved. Our study participants' experiences of initial frustration, curiosity, growing realization that they were able to learn how to code, and desire for greater autonomy appears similar to the reactions noted in the findings of Zha, Moore, and Gaston (2020). The easing of anxiety for the majority of the learners suggest that careful pacing of the introduction of new programming concepts and processes was appropriate. And, like Zha, Moore, and Gaston (2020), we also recommend step-by-step scaffolding in which small coding successes (easier tasks) are interspersed.

Interestingly, the students' comments indicated that they had absorbed strategies that were unplanned such as learning how to solve problems by watching the instructors search through help forums. The students indicated that they enjoyed the introduction of different instructional techniques such as direct, follow-the-leader coding, arranging paper coding snippets, modifying and/or correcting code already available within a downloaded application, and debugging code with an emulator. There were comments that the students enjoyed the Parsons problems as well as having access to sample code and having the opportunity to fix incorrect code. Gok and Kwon (2020) also noted in their study that one of their pre-service-teacher participants used “example codes as her strategy” (p. 4). Furthermore, the effectiveness of concrete objects and samples to teach programming is supported in the literature (Pöllänen & Pöllänen, 2019). There are clues in the students' comments that the use of physical paper-code snippets and testing one's work on real mobile devices added to the sense of concreteness and increased student satisfaction.

Similar to the findings of Zha, Moore, and Gaston (2020), we observed that the pre-service teachers in our study were very active and engaged during the pair and group programming activities. The students' comments on the exit tickets also suggested that they appreciated the social aspects of learning. Some noted that discussions inspired new ideas and creative ways to solve problems. In fact, Gok and Kwon (2020) found that their study participants “perceived peers' support as the most valuable and the quickest way to overcome the difficulties” they faced, and only if their peers could not assist would they go to the instructors (p. 4). While our study was just one course with a limited number of enrolments (n=22), there was evidence of growth in computational thinking about problem-solving strategies and collaborative learning—or at the very least, in an appreciation for collaborative learning—which as suggested earlier can create a more comfortable place for female learners and can help to scaffold learners towards greater autonomy (Hanks et al., 2011; Werner & Denning, 2009).

The main challenge for the evaluation was in the identification of threshold concepts. The actual question was phrased as “Did you have an “ah-ha!” moment (i.e., a moment when something became clear or a moment in which you were surprised by something) during hour of code? If so, please describe it.” In response, many students simply listed the new skills they had learned or that they were surprised when their app worked during testing. By end of the second hour-of-code lesson, the students started commenting on how they started to feel like they were able to effectively learn how to code. One student called this an “I can do this AHA moment.” Such *can do* comments suggest increases in students’ sense of self-efficacy (i.e., the sense that they are able to perform the coding task(s)).

The instructors also noted comments suggesting increased personal autonomy in the students’ own learning efforts. The instructors were unable to identify any other threshold concepts in the students’ comments. Furthermore, there is no way to know the degree to which the students’ self-perception, autonomy, or understanding was transformative or long-lasting. The extent to which the coding experience in this class led to irreversible transformation would require a more longitudinal study or, at the very least, a follow-up with the students. Finally, in future classes, it might be helpful to discuss in more depth what threshold concepts are in order to assist the students in responding to the question on the exit ticket.

CONCLUSION

Naturally, students varied in their computer comfort levels from the outset, and these variations persisted throughout the semester. While some learners expressed a sense of success early in the course, there were others who indicated ongoing struggles with the programming interface and computational concepts. Similar to reports from other researchers, meeting individual student needs in computer programming classes remains a challenge (Yadav et al., 2016). There were some comments in the student evaluations that a highly scaffolded, follow-me approach to coding with a gradual shift away from direct instruction towards greater discovery and learner self-direction is an effective way to meet the needs of learners with different comfort and comprehensions levels. The students in this multimedia class appreciated seeing the end-product demonstrated on an actual phone, arranging code snippets (both on paper and already in an application) like a jigsaw puzzle, and having opportunities and challenges to personalize code. This supports Pöllänen and Pöllänen’s (2019) description of the spiral approach to teaching in which one starts with more concrete objects (examples) and moves towards creation of algorithms and more complex coding. To be sure, the instructors will continue to experiment with different pedagogical approaches in future offerings of the multimedia course to assist the teacher candidates in learning code and developing strategies for teaching code. Ultimately, the goal is to help them in understanding which configurations might result in cookies and which might result in cake.

FUNDING AGENCY

This research received no specific grant from any funding body in the public, commercial, or not-for-profit sectors.

Conflicts of Interest

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

Process Dates:

Received: November 20, 2021, Revision: March 4, 2022, Accepted: March 4, 2022

Corresponding Author:

Correspondence should be addressed to Marguerite Koole, m.koole@usask.ca

REFERENCES

- Arfé, B., Vardanega, T., & Ronconi, L. (2020). The effects of coding on children's planning and inhibition skills. *Computers & Education*, 148, 103807. doi:10.1016/j.compedu.2020.103807
- Ateşkan, A., & Hart, D. O. (2021). Demystifying computational thinking for teacher candidates: A case study on Turkish secondary school pre-service teachers. *Education and Information Technologies*, 26(5), 6383–6399. doi:10.1007/s10639-021-10626-9
- Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12. *ACM Inroads*, 2(1), 48–54. doi:10.1145/1929887.1929905
- Bers, M. U. (2019). Coding as another language: A pedagogical approach for teaching computer science in early childhood. *Journal of Computers in Education*, 6(4), 499–528. doi:10.1007/s40692-019-00147-3
- Chandra, V., & Lloyd, M. (2020). Lessons in persistence: Investigating the challenges faced by preservice teachers in teaching coding and computational thinking in an unfamiliar context. *The Australian Journal of Teacher Education*, 45(9), 1–23. doi:10.14221/ajte.2020v45n9.1
- Code.org. (2021). <https://code.org/>
- Cousin, G. (2010). Neither teacher-centred nor student-centred: Threshold concepts and research partnerships. *Journal of Learning Development in Higher Education*, 2(2), 1–9. doi:10.47408/jldhe.v0i2.64
- Dağ, F. (2019). Prepare pre-service teachers to teach computer programming skills at K-12 level: Experiences in a course. *Journal of Computers in Education*, 6(2), 277–313. doi:10.1007/s40692-019-00137-5
- Doleck, T., Bazelais, P., Lemay, D. J., Saxena, A., & Basnet, R. B. (2017). Algorithmic thinking, cooperativity, creativity, critical thinking, and problem solving: Exploring the relationship between computational thinking skills and academic performance. *Journal of Computers in Education*, 4(4), 355–369. doi:10.1007/s40692-017-0090-9
- Gok, F., & Kwon, K. (2020). A Case Study Exploring Pre-Service Teachers' Programming Difficulties and Strategies when Learning Programming Languages. *Psychology and Cognitive Sciences: Open Journal*, 6(1), 1–6. doi:10.17140/PCSOJ-6-152
- Grey, W. (2021, September 24). How to support your students to write code. *Hello World! The Big Book of Computing Pedagogy*, 82–83.
- Grover, S., & Pea, R. (2013). Computational thinking in K–12. *Educational Researcher*, 42(1), 38–43. doi:10.3102/0013189X12463051
- Hanks, B., Fitzgerald, S., McCauley, R., Murphy, L., & Zander, C. (2011). Pair programming in education: A literature review. *Computer Science Education*, 21(2), 135–173. doi:10.1080/08993408.2011.579808
- Harrington, B., & Cheng, N. (2018). Tracing vs. writing code: Beyond the learning hierarchy. *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, 423–428. doi:10.1145/3159450.3159530
- Hertz, M., & Jump, M. (2013). Trace-based teaching in early programming courses. *Proceedings of the 44th ACM Technical Symposium on Computer Science Education*, 561–566. doi:10.1145/2445196.2445364
- Izu, C., Schulte, C., Aggarwal, A., Cutts, Q., Duran, R., Gutica, M., Heinemann, B., Kraemer, E., Lonati, V., Mirolo, C., & Weeda, R. (2019). Fostering program comprehension in novice programmers - Learning activities and learning trajectories. *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*, 27–52. doi:10.1145/3344429.3372501
- Kafai, Y. (2016). From computational thinking to computational participation in K-12 education. *Communications of the ACM*, 59(8), 26–27. doi:10.1145/2955114
- Kafai, Y. B., & Proctor, C. (2021). A reevaluation of computational thinking in K–12 education: Moving toward computational literacies. *Educational Researcher*. 10.3102/0013189X211057904
- Koole, M., & Squires, V. (2020). The education system of Canada: ICT and STEM balancing economics with social justice. In S. Jornitz & M. do Amaral (Eds.), *The Education Systems of the Americas* (pp. 1–22). Springer International Publishing., doi:10.1007/978-3-319-93443-3_39-1

- Laffee, S. (2017). Coding: The new 21st-century literacy? *Education Digest*, 83(2), 25–32. <http://cyber.usask.ca/login?url=https://search.proquest.com/docview/1932045732?accountid=14739>
- Menekse, M. (2015). Computer science teacher professional development in the United States: A review of studies published between 2004 and 2014. *Computer Science Education*, 25(4), 325–350. doi:10.1080/08993408.2015.1111645
- MIT App Inventor*. (n.d.). <http://appinventor.mit.edu>
- Pöllänen, S., & Pöllänen, K. (2019). Beyond programming and crafts: Towards computational thinking in basic education. *Design and Technology Education: An International Journal*, 24(1), 13–32. <https://ojs.lboro.ac.uk/DATE/article/view/2566>
- Werner, L., & Denning, J. (2009). Pair programming in middle school. *Journal of Research on Technology in Education*, 42(1), 29–49. doi:10.1080/15391523.2009.10782540
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33–35. doi:10.1145/1118178.1118215
- Yadav, A., Gretter, S., Hambrusch, S., & Phil, S. (2016). Expanding computer science education in schools: Understanding teacher experiences and challenges. *Computer Science Education*, 26(4), 235–254. doi:10.1080/08993408.2016.1257418
- Zha, S., Jin, Y., Moore, P., & Gaston, J. (2020). Hopscotch into coding: Introducing pre-service teachers computational thinking. *TechTrends*, 64(1), 17–28. doi:10.1007/s11528-019-00423-0

Marguerite Koole completed her PhD in E-Research and Technology-Enhanced Learning at Lancaster University UK in 2013. Her thesis is entitled "Identity Positioning of Doctoral Students in Networked Learning Environments". She also holds a Masters of Education in Distance Education (MEd) through the Centre for Distance Education at Athabasca University. Her focus was on mobile learning. Dr. Koole has a BA in Modern Languages and has studied French, Spanish, German, Blackfoot, Cree, Latin, Mandarin, ancient Mayan hieroglyphics, and linguistics. She completed a college diploma in Multimedia Production with training in web development, audio, video, animation, 3D animation, marketing, and business. Through the years, Dr. Koole has been involved in teaching, instructional design, multimedia programming, content management, e-portfolios, and social software. She has designed interactive, online learning activities for various learning purposes and platforms—including print, web, and mobile devices.

Kaleigh Elian is an Educator, Disability Advisor, Academic Strategist, and ADHD Coach with the Learning Disabilities Association of Saskatchewan. She is also continuing her education in Computer Science and Software Development at the University of Saskatchewan. Ms. Elian graduated with a Bachelor of Arts in History and holds a Bachelor of Education from the University of Saskatchewan. She has experience in teaching, computer programming, and robotics.