INVESTIGATING UNDERGRADUATE STUDENTS' GENERALIZING ACTIVITY IN A COMPUTATIONAL SETTING

Elise Lockwood
Oregon State University
elise.lockwood@oregonstate.edu

Adaline De Chenne
Oregon State University
dechenna@oregonstate.edu

Computational activity is increasingly relevant in education and society, and researchers have investigated its role in students' mathematical thinking and activity. More work is needed within mathematics education to explore ways in which computational activity might afford development of mathematical practices. In this paper, we specifically examine the generalizing activity of undergraduate students who solved combinatorial problems in the context of Python programming. We demonstrate instances of generalizing in terms of Ellis et al.'s (2017) framework, and we argue that some opportunities were facilitated and supported by the computational setting in which the students worked.

Keywords: Generalization, Combinatorics, Computational Thinking

Introduction and Motivation

Computation is an increasingly essential aspect of science and mathematics, and researchers and policy makers within STEM broadly (Blikstein, 2018; NGSS Lead States, 2013; Weintrop, Beheshti, Horn, Orton, Jona, Trouille, & Wilensky, 2016), and mathematics education especially (e.g., Benton, Saunders, Kalas, Hoyles, & Noss, 2018; Buteau & Muller, 2017; Cetin & Dubinsky, 2018; Hoyles & Noss, 2015; Feurzeig, Papert, & Lawler, 2011), are making the case for more attention to be paid to computing in the field. Central to current interest in computing are questions related to whether and how computing might strengthen students' mathematical thinking and activity, including their engagement in mathematical practices. The question of whether such computing allows for transfer of content knowledge or practices is a source of debate, with some researchers making the case for and some against arguments that evidence of transfer exists (see Tedre & Denning (2016) for discussion). Acknowledging this debate, we investigate such questions with a qualitative exploration in which we demonstrate how computing may support students' engagement with a mathematical practice – generalization.

Our aim is to answer the following research question: *In what ways did a computational setting support undergraduate students' generalizing activity on combinatorial tasks?* We hope that by providing qualitative interview data, we can gain some insight into how students engage in generalization in a computational setting. We see the paper as serving both a specific and a broader purpose. First, the paper is meant to highlight specifically how students engage in the particular practice of generalization with the use of programming (and, even more, within the domain of combinatorics). In this way, our results shed light on generalization as a practice, and they can also illuminate implications for generalization within combinatorics. At a broader level, this paper can serve as an instance of how students can make connections to and engage in mathematical practices within a computational setting.

Relevant Literature and Theoretical Perspectives

Literature and Theoretical Perspectives on Computation

In this paper we focus on *machine-based computing*, which we take to be the practice of developing and precisely articulating algorithms that may be run on a machine. We make two distinctions in this characterization. First, while computing can encompass many kinds of activity, we focus on activity that involves developing, articulating, and implementing algorithms. Second, while such algorithm development could occur strictly by hand, we focus on activity that uses a machine. Further, we distinguish machine-based computing from engagement with technology more generally, which might include using computer algebra systems or dynamic geometry software – for us, machine-based computing involves not just using software, but engaging in algorithm design and implementation in some capacity. In our study, the specific machine-based computing in which our students engaged was programming in Python.

Mathematics education research has a history of using computers to enhance students' mathematical reasoning, beginning with Papert's (1980) introduction of Logo to help young children. Recently there seems to be an increasing amount of attention being paid to computation in education research, perhaps due in part to Wing's (2006, 2008) re-popularization of the term *computational thinking*. We have seen considerable attention paid recently toward examining the role of computing in mathematics education (e.g., Benton, et al. 2018; Buteau, Gueudet, Muller, Mgombelo, & Sacristan, 2019; Buteau & Muller, 2017; Cetin & Dubinsky, 2018; DeJarnette, 2019; Lockwood, DeJarnette, & Thomas, 2018; Lockwood & De Chenne, 2019).

We acknowledge that there is some discussion about whether or not computing can be effective in helping students engage in other practices and skills (e.g., Tedre & Denning, 2016). However, in spite of such debates, we think it is still worth investigating the degree to which computational activity might in fact support students' thinking and engagement in mathematical practices. Part of our reason for this is that we feel there are useful frameworks within mathematics education (such as Lobato's view of actor-oriented transfer) that may bring fresh perspectives toward questions of the role of computing in mathematics education. We draw on recent work by Lockwood et al. (2019), who interviewed research mathematicians in academia about their use of computing in their work. While these mathematicians suggested many benefits of computing, they also "framed computing as allowing for some other important habits of mind or practices related to their work" (p. 9). While these mathematicians did not name the practice of generalizing specifically, we view our work as building on these findings by Lockwood et al. The mathematicians reported in Lockwood et al.'s study shared their own beliefs and experiences, and we wanted to explore and demonstrate some of their claims with student data, offering insights into how students' engagement with computation can actually give opportunities for students to connect computing to the practice of generalizing.

Literature and Theoretical Perspectives on Generalization

Generalization is an essential aspect of mathematical thinking and learning, and there has been much work that has established the importance of generalization in mathematics education. Such work has included investigations into students' generalizing within algebraic contexts (e.g., Amit & Neria, 2008; Ellis, 2007a, 2007b, Radford, 2008; Rivera & Becker, 2007, 2008), and there has also been exploration into generalizing activity among undergraduates in areas like calculus, linear algebra, and combinatorics (e.g., Dubinsky, 1991; Jones & Dorko, 2015; Kabael, 2011; Lockwood, 2011; Lockwood & Reed, 2018). Researchers have also proposed theories about the nature of generalization, providing some categories and distinctions of generalizing activity (e.g., Ellis, 2007a; Harel & Tall, 1989; Harel, 2008). Together these studies have provided rich insight into the nature of generalization in a variety of situations and contexts. We aim to contribute to this body of work by examining generalizing within the context of machine-based computing, and we hope to identify and understand specific ways that a computational setting might support students' generalizing activity.

Broadly, Ellis (2007a) followed Kaput (1999) and defined generalization as "engaging in at least one of three activities: a) identifying commonality across cases, b) extending one's reasoning beyond the range in which it originated, or c) deriving broader results about new relationships from particular cases (p. 444), and we similarly adopt that broad characterization. We also draw upon Ellis, Lockwood, Tillema, & Moore's (2017) Relating-Forming-Extending (R-F-E) framework of generalizing activity in characterizing generalization. Ellis et al. (2017) emphasize three different generalizing activities, which build upon a previous taxonomy that Ellis (2007a) had developed. In *relating,* students establish "relationships of similarity across problems or contexts" (p. 680), and so students make connections among situations they have encountered. In *forming*, students engage in "searching for and identifying similar elements, patterns, and relationships" within a single task (p. 680). Here, students may be attending to regularity and articulate some general pattern or relationship that they observe. In *extending,* "students extend established patterns and regularities to new cases" (p. 680). This might typically involve some increased abstraction (such as moving from numerical cases to arguments involving variables). Ellis et al. also discuss ways in which these generalizing activities are interrelated – for instance, relating and forming may help students start to identify some regularity, which can then facilitate their extending to more general cases. This categorization offers language by which to characterize generalizing activity that we observed in our students. We focus on instances of relating and forming in this paper.

Mathematical Discussion and Motivation for Focusing on Combinatorics

There are a couple of reasons that we focus on combinatorics in this paper. First, the computational setting is particularly well-suited for combinatorial problems, in the sense that some of the features of the programs (loop structures, conditional statements) serve to highlight important combinatorial concepts. We have articulated this phenomenon elsewhere, including demonstrating students' uses of conditional statements to reason about types of counting problems (Lockwood & De Chenne, 2019) and highlighting the computer's effectiveness in helping students verify solutions to counting problems (De Chenne & Lockwood, In press). We believe that combinatorial problems provide rich contexts in which students can solve mathematical problems in computational settings. In addition, combinatorial tasks are well suited to generalization, and researchers have previously explored students' generalizing activity on combinatorial problems (e.g., Lockwood, 2011; Tillema & Gatza, 2018). Our work builds on such studies by illuminating ways in which the computational setting supports generalizing within combinatorics. We thus aim to contribute both to work on generalization and work on combinatorics, building on our knowledge base of students' generalizing activity within the field of combinatorics especially. Finally, on the whole, combinatorial problems can be difficult for students to solve (e.g., Batanero, Navarro-Pelayo, & Godino, 1997; Lockwood & Gibson, 2016), and we see value in investigating ways to improve students' combinatorial experiences. In this case, by focusing on generalization within a computation setting, we gain insight into how students might understand and generalize ideas within the particular domain of combinatorics.

Methods

Data Collection

We draw on two data sources for this paper. First, we conducted a paired teaching experiment (in the sense of Steffe & Thompson, 2000) with two undergraduate students, Charlotte and Diana (all names are pseudonyms). They were chemistry majors recruited from a vector calculus class, and they participated in selection interviews, which indicated that they had not taken courses in discrete math, they were not familiar with combinatorial formulas, and they had no prior programming experience. Second, we share results from an individual interview with a computer science (CS) student, Allen. He was a CS major recruited from an introductory class in computer science. He indicated on a recruitment survey that he had not taken a class in discrete mathematics and that he had programming experience. In both cases, the students sat at a computer and worked on combinatorial tasks, writing in a Python coding environment while the interviewer asked clarifying questions. Charlotte and Diana participated in 11 total interview sessions during which they solved a variety of counting problems. Allen participated in 3 total interview sessions during which he wrote programs to list all outcomes of counting problems. In the final interview (from which the data in this paper is taken), we asked him only to solve a counting problem, and we then prompted him to explain how he would verify his solution.

Data Analysis

The data are part of a project in which we explored students' combinatorial thinking and activity within a computational setting, and we were not explicitly targeting generalization in this project. However, as we reviewed data it became clear that students were engaging in generalizing activity, and we wanted to examine that activity more systematically. For analysis, we surveyed both sets of data for instances that illustrated each form of generalization in the R-F-E framework. We particularly sought examples that would highlight the role of the computer and ways in which it supported students' generalizing activity. We identified a number of episodes of relating and forming in our data sets, and we chose the two episodes discussed in this paper as representative examples. Together we discussed additional generalizing in our data, and we articulated ways in which the computer in particular facilitated generalizing activity, which we elaborate in the Results and the Discussion and Implications sections.

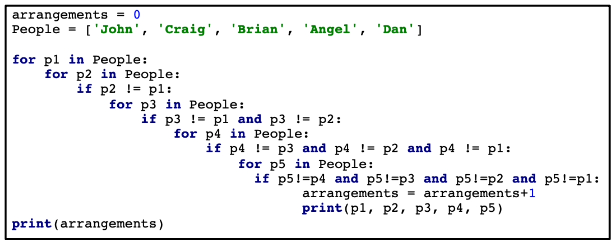
Results

We provide two examples of students engaging in generalizing activity, and we focus especially on *relating* and *forming* within the R-F-E framework. We do not provide data related to extending in part due to space, and also because we hypothesize that the computer is particularly useful in supporting relating and forming, and students can then extend ideas and relationships by hand. We elaborate this point in the Discussion Implications section.

Relating

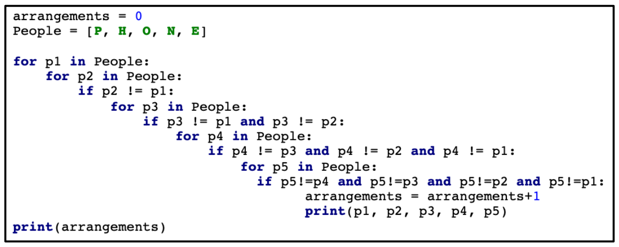
We demonstrate one particular sub-category of relating that Ellis et al. (2017) described: *relating objects*, which involves forming a relationship of similarity between two or more present mathematical objects. We demonstrate an instance of relating objects in which Charlotte and Diana connected back to work on a prior problem they had done (both sets of students engaged in more relating, but we do not have space to offer additional examples). We highlight the tendency of students to copy and paste, then edit, code from prior problems (we call this repurposing previous code), which we feel is a feature of the computer that particularly supported relating. On the one hand, repurposing code could seem just like practical, time-saving technique, but we argue that this is actually important for generalization for a couple of reasons. First, the code itself gives students a new aspect of the problem to which and from which they can relate. Because the code can be seen as encapsulating and representing a counting process, students can identify similarities between the representation of code on various problems. Our students drew on similar structures and features of code as they copied and pasted work from prior problems. For instance, at one point, Diana asked Charlotte, "Do you want me to copy this code from over here [a previous problem], since it's really similar?" This suggests that she perceived similarity between code they had written in the past and a current situation. Notably, the computer specifically facilitates that similarity by allowing such repurposing easily to occur. With very little effort, students get to duplicate and then adjust something they did previously. Such adjustments could be done by hand, but there is something about editing and adjusting real time that allows for efficiency. We also hypothesize that because copying and pasting reduces students' work load, it may incentivize their looking for similarity between solutions, thus encouraging generalizing activity.

As an example of relating, we offer an instance of Charlotte and Diana repurposing code. They had previously worked on a problem about enumerating people, *How many ways are there to rearrange 5 people: John, Craig, Brian, Angel, and Dan?*, reasoning about code in Figure 1a. This code counts arrangements of five people – the nested loops cycle iteratively through each element in the set People, and != (not equal) prevents elements from being repeated (more details about such problems are in Lockwood & De Chenne, 2019). We later asked: *Write some code to list and count the number of ways to arrange the letters in the word PHONE. How many outcomes are there? What do you think the output will look like?* As they started this problem, they had the following exchange.



```
arrangements = 0
People = ['John', 'Craig', 'Brian', 'Angel', 'Dan']

for p1 in People:
    for p2 in People:
        if p2 != p1:
            for p3 in People:
                if p3 != p1 and p3 != p2:
                    for p4 in People:
                        if p4 != p3 and p4 != p2 and p4 != p1:
                            for p5 in People:
                              if p5!=p4 and p5!=p3 and p5!=p2 and p5!=p1:
                                    arrangements = arrangements+1
                                    print(p1, p2, p3, p4, p5)
print(arrangements)
```



```
arrangements = 0
People = [P, H, O, N, E]

for p1 in People:
    for p2 in People:
        if p2 != p1:
            for p3 in People:
                if p3 != p1 and p3 != p2:
                    for p4 in People:
                        if p4 != p3 and p4 != p2 and p4 != p1:
                            for p5 in People:
                              if p5!=p4 and p5!=p3 and p5!=p2 and p5!=p1:
                                    arrangements = arrangements+1
                                    print(p1, p2, p3, p4, p5)
print(arrangements)
```

Figure 1a, 1b: Code for the People and PHONE problems

Charlotte: Okay. So, I feel like, yeah, basically just gonna be the same as this one because, I mean PHONE has five letters and this had five people. So, I feel like we can maybe just copy the same code.

Diana: Yeah, and then like edit it to be like PHONE.

The interviewer then asked her why their idea would work, and Diana said the following:

Diana: It works because there's like the same number of things that you're arranging. So, like you're arranging people here, there's five of them and then you're arranging letters here and there's five of them. And it's still the same as like you're not repeating any letter, so you keep the not equal to expressions. And, yeah.

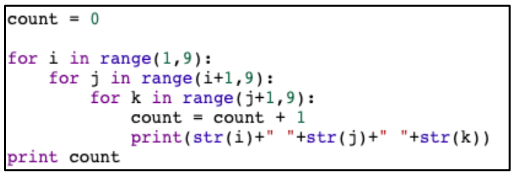
Diana's comments highlight what she perceived as similar about the situation – they were still arranging five objects, and they still did not want to be able to repeat any object. So, she noted that they still wanted to maintain the same fundamental features of the code, which was the "not equal to expressions." They edited the code from Figure 1a just to have the letters P, H, O, N, and E instead of the people (Figure 1b). They ran the code, which correctly printed all 120 arrangements of letters in the word PHONE. The fact that they left much of their code the same as the People problem is noteworthy, as it suggests that they were attuned to the fact that some features of the code were or were not essential to solving the new problem. Notably, they did not change what they perceived as features that would not change their process or output (for instance, they did not re-name the set People, and they kept and p1 through p5 as their variables).

That is, they recognized that the underlying structure was the same between the problems, but some other features, like the names of the set and variables, did not matter. This gives insight into what they deemed as relevant similarities or differences among the two problems.

To summarize, the computational setting afforded students with opportunities to repurpose code, and by doing so they related current situations to prior work. The computational representation of the code signified a particular counting process, and the computer's capacity to allow such code to be repurposed facilitated the students' generalizing activity of making connections among problems. We demonstrated one instance of this, but there were multiple examples throughout the teaching experiments of such activity. In this way, this example of relating within the computational setting lets us see one way in which computational settings could afford some unique opportunities for engagement in the practice of generalization.

Forming

Next, we offer an instance of forming, and Ellis et al. (2017) distinguish between multiple types of forming. We focus on two of these, *searching for similarity or regularity* (searching to find a stable pattern, regularity, or element of similarity across cases, numbers, or figures), and *identifying a regularity* (identification of a regularity or pattern across cases, numbers, or figures). In this example we highlight Allen, who was working on the Books problem, which states *Suppose you have 8 books and you want to take three of them with you on vacation. How many ways are there to do this?* Allen originally solved this problem incorrectly by finding the number of ways to arrange three of the eight books, 8*7*6 = P(8,3), rather than selecting three of the eight books, 8*7*6/6 = C(8,3). (We will refer to this correct answer as C(8,3), even though Allen did not yet know a closed form for it, and he could only find the value on the computer using his program.) After writing code that listed the outcomes of arranging the books, he noticed that outcome 132 appeared in the output after the outcome 123, and he realized that he had not been correct, stating "that would not be a good combo because it already appeared up here; it's just in a different order." He thus realized he needed to correct his solution, and to do so Allen wrote the code in Figure 2. When run, this code prints all three number combinations in ascending order, thus ensuring that each combination is printed exactly once, and it correctly outputs 56 as the total number of combinations.

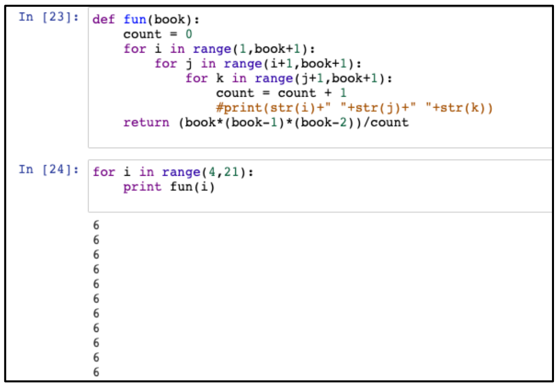


```
count = 0

for i in range(1,9):
    for j in range(i+1,9):
        for k in range(j+1,9):
            count = count + 1
            print(str(i)+" "+str(j)+" "+str(k))
print count
```

Figure 2: Allen's code for the Books problem

While Allen's code correctly counted the number of outcomes of the Books problem, he did not initially offer any justification for a counting process or mathematical expression. However, he remarked that it was interesting that 56 = 8*7, which was his original answer of 8*7*6 divided by 6. So, he realized that a ratio of his answer over the correct answer was 6 (that is, P(8,3)/C(8,3) = 6). He wondered what would happen if the problem were selecting from only 7 books instead of 8. After extending his original (and incorrect) solution to seven books, P(7,3) = 7*6*5, he predicted that the ratio of P(7,3)/C(7,3) would be 5. (One potential rationale for this prediction is that he reduced the total number of books by 1 (from 8 to 7), and so he reduced his prediction by 1). Allen then decided to use the computer and his computational experience to explore these relationships more systematically. First, he adjusted the code from Figure 2 to count the number of outcomes for 7 books (rather than 8), yielding 35, and he computed the ratio of P(7,3)/C(7,3), yielding 6 (which contradicted his prediction of 5). Allen then adjusted his code to allow for him to explore more examples efficiently. In particular, he created a function that would let him explore multiple numbers of books within a range, from which he always selected 3 books. For each number of books, the program computed the ratio of his estimated guess (P(n,3) with the actual correct value that he found computationally (which is C(n,3)). Allen's code is displayed in Figure 3, and the output shows verified that the ratio P(*n*,3)/C(*n*,3) was 6 in each case (again, we write P(*n*,3)/C(*n*,3) for clarity, as Allen thought about this ratio as his original solution divided by the actual solution). We view Allen's initial exploration and creation of this function activity as an instance of forming, namely *searching for similarity or regularity*, as he was making predictions and looking for and expecting to observe patterns. Then, when he ultimately determined that the ratio was 6, we take this as an instance of *identifying a regularity*.



```
In [23]: def fun(book):
             count = 0
             for i in range(1,book+1):
                 for j in range(i+1,book+1):
                     for k in range(j+1,book+1):
                         count = count + 1
                         #print(str(i)+" "+str(j)+" "+str(k))
             return (book*(book-1)*(book-2))/count

In [24]: for i in range(4,21):
             print fun(i)

         6
         6
         6
         6
         6
         6
         6
         6
         6
         6
         6
```

Figure 3: Allen's identification of regularity via programming a function

Allen then decided to extend his work by changing the number of books being selected in his code; that is, rather than selecting three books, he wrote code to select four, five, and then six books. It is noteworthy that Allen had not yet provided justification for the constant 6, and so the decision to extend his work seems to stem from his desire to observe a pattern (thus we view this as an instance of forming). Further, the computer facilitated this forming activity by allowing him to adjust his previous code so other numbers of books were selected. For each of these new instances, he divided the answer from his original solution method by the actual answer, and he observed a constancy in each case. Essentially, Allen fixed an *m* and used his code to calculate P(*n,m*)/C(*n,m*) = *m*! as *n* ranged for values of from *m* + 1 to 21, (again, he expressed this ratio as his original solution divided by the actual solution). The constant he found in each case represented the number of ways to arrange the books after the books have been selected. After finding values in cases three through six, he remarked "Okay, I think I found a really high-level relationship that is several layers." We asked him to elaborate, and Allen stated the following.

Allen: So, this is the number of books. This is three books, four books, five books, six books. So, if that's the case, then with two books it should be 3. Three would be 6, four would be 24, five would be 120, and six would be 720… what I noticed is each time you go up, you multiply by the next number. So, 6 times 4 equals 24, which multiplied by 5 equals 120, which multiplied by 6 equals 720.

Allen went on to observe that "these are all factorials." Using this information, he constructed and justified a closed form for C(*n,m*) (we do not include analysis of this data due to space).

To summarize this episode, Allen used his code to find a constant ratio between his (incorrect) original solution and the correct solution he computed. He then identified a pattern between these constants as the number of books being selected was increased. We observed *searching for similarity or regularity* when Allen identified the constant 6 in his work on selecting three books. Then, we observed *identifying a regularity* when Allen found a pattern among constants as the number of books selected increased. We argue that the computer was fundamental in this process, as Allen generated these constants by writing and implementing code. The computer, and the outcomes generated, seemed to afford Allen the opportunity to search for patterns and identify relationships. In Allen's case, he used the computer in two important but different ways. First, he used the computer to generate answers to problems he could not yet solve by hand (computing the correct number of combinations before he knew the closed form of C(m,n)). Second, he wrote a function to generate multiple cases, which allowed him to search for regularity in multiple cases efficiently. This episode thus sheds light on how the computational setting supported Allen in the specific generalizing activity of forming.

Discussion and Implications

In this paper, we have offered instances of students engaging in generalizing activities of relating and forming (in terms of Ellis et al.'s (2017) R-F-E framework) within the context of programming in Python. We have tried to make the case that in these cases the computer offered specific affordances for generalizing activity. These included copying and pasting to support relating, and generating correct solutions to multiple problems in order to support forming. While some of these activities would technically be possible by hand, the manipulation that the computer allows seemed to expediate this practice of generalization for the students. As an additional note, we mostly observed the computer being used to support relating and forming, and extending that we observed among students tended to be done by hand (students often extended formulas or expressions they had written by hand). We thus hypothesize that the computer may be most effective for supporting relating and forming, which then contribute to extending. More work is needed to explore whether and how extending arises explicitly via computational activity.

There is more to study specifically about the role of the computer (and specifically machine-based computing) in facilitating students' generalizing activity. We have focused on combinatorics, but such computing may elicit generalizing in other ways in other domains. Researchers could explore ways that the computer might facilitate other kinds of generalizing activity in other domains or in other kinds of problems. Further, we have focused on one perspective of generalization, drawing explicitly on Ellis et al.'s (2017) framework, but researchers could consider other possible framings of generalization to consider the computer's role in supporting students' generalization.

In addition, our results demonstrate instances in which students engage in the practice of generalization. However, there are many other practices, and Lockwood et al. (2019) have suggested that other practices like proving or problem solving might closely be related to the kind of machine-based computing described in this paper. Thus, future research could be conducted on ways in which computing might support other mathematical practices.

Acknowledgments

This material is supported by the National Science Foundation under Grant No. 1650943.

References

Batanero, C., Navarro-Pelayo, V., & Godino, J. (1997). Effect of the implicit combinatorial model on combinatorial reasoning in secondary school pupils. *Educational Studies in Mathematics, 32*, 181-199.

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

Blikstein, P. (2018). Pre-College Computer Science Education: A Survey of the Field. Mountain View, CA: Google LLC. Retrieved from https://goo.gl/gmS1Vm

Buteau, C., Gueudet, G., Muller, E., Mgombelo, J., & Sacristan, A. I. (2019). University students turning computer programming into an instrument for 'authentic' mathematical work. *International Journal of Mathematical Education in Science and Technology*. doi:10.1080/0020739X.2019.1648892

Buteau, C. & Muller, E. (2017). Assessment in undergraduate programming-based mathematics courses. *Digital Experiences in Mathematics Education, 3*, 97-114. doi:10.1007/s40751-016-0026-4

Cetin I., & Dubinsky, E. (2017). Reflective abstraction in computational thinking. *Journal of Mathematical Behavior, 47*, 70-80.

De Chenne, A. & Lockwood, E. (In press). Student verification practices for combinatorics problems in a computational environment. In the proceedings of the 23rd Annual Conference on Research in Undergraduate Mathematics Education.

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

diSessa A. A. (2018). Computational Literacy and "The Big Picture" Concerning Computers in Mathematics Education. *Mathematical Thinking and Learning*, 20(1), 3–31. https://doi.org/10.1080/10986065.2018.1403544

Eizenberg, M. M., & Zaslavsky, O. (2004). Students' verification strategies for combinatorial problems. *Mathematical Thinking and Learning, 6*(1), 15-36.

Ellis, A. B. (2007). A taxonomy for categorizing generalizations: Generalizing actions and reflection generalizations. *Journal of the Learning Sciences*, *16*(2), 221–262.

Fenton, W. & Dubinsky, E. (1996). *Introduction to Discrete Mathematics with ISETL*. New York: Springer-Verlag.

Feurzeig, W., Papert, S., & Lawler, B. (2011). Programming-languages as a conceptual framework for teaching mathematics. *Interactive Learning Environments, 19*(5):487–501.

Hoyles, C., & Noss, R. (2015). A computational lens on design research. In S, Prediger, K. Gravemeijer, & J. Confrey, (Eds.) Design research with a focus on learning processes: an overview on achievements and challenges. *ZDM, (47)*6, 1039 -1045.

Lockwood, E., DeJarnette, A. F., & Thomas, M. (2019) Computing as a mathematical disciplinary practice. Online first in *Journal of Mathematical Behavior*. doi: 10.1016/j.jmathb.2019.01.004

Lockwood, E. & De Chenne, A. (2019). Using conditional statements in Python to reason about sets of outcomes in combinatorial problems. *International Journal of Research in Undergraduate Mathematics Education*.

NGSS Lead States. (2013). *Next generation science standards: for states, by states*. Washington, DC: The National Academies Press.

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

Tedre, M. & Denning, P. J. (2016). The long quest for computational thinking. *Proceedings of the 16th Koli Calling International Conference on Computing Education Research,* 120-129.

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 and Education Technology, 25*, 127-147. Doi: 10.1007/s10956-015-0581-5.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*. *49*(3), 33-35.