Routledge
Taylor & Francis Group

Cognition and Instruction

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/hcgi20

Leveraging Prediction and Reflection in a Computational Setting to Enrich Undergraduate Students' Combinatorial Thinking

Elise Lockwood

To cite this article: Elise Lockwood (2022): Leveraging Prediction and Reflection in a Computational Setting to Enrich Undergraduate Students' Combinatorial Thinking, Cognition and Instruction, DOI: 10.1080/07370008.2021.2020793

To link to this article: https://doi.org/10.1080/07370008.2021.2020793

COGNITION AND INSTRUCTION
https://doi.org/10.1080/07370008.2021.2020793

Routledge
Taylor & Francis Group

Leveraging Prediction and Reflection in a Computational Setting to Enrich Undergraduate Students' Combinatorial Thinking

Elise Lockwood

Department of Mathematics, Oregon State University, Corvallis, OR, USA

ABSTRACT

In this paper, I discuss undergraduate students' engagement in basic Python programming while solving combinatorial problems. Students solved tasks that were designed to involve programming, and they were encouraged to engage in activities of prediction and reflection. I provide data from two paired teaching experiments, and I outline how the task design and instructional interventions particularly supported students' combinatorial reasoning. I argue that emergent computational representations and the prompts for prediction and reflection were especially useful in supporting students' reasoning about fundamental combinatorial ideas. I argue that this particular mathematical example informs broader notions of disciplinary reflexivity and representational heterogeneity, providing insight into computational thinking practices in the domain of mathematics. Ultimately, I aim to explore the nature of computing and enumeration, shedding light on why the two disciplines are particularly well-suited to support each other. I conclude with implications and avenues for future research.

Introduction and motivation

In what ways can students' experiences with computing support and enrich their domain-specific reasoning? In recent decades, many mathematics and science education researchers have explored questions like this, which interrogate the ways in which computational thinking practices interact with students' understanding of mathematical and scientific concepts. Much of this work is aimed at fundamental epistemic questions about the nature of *computational thinking*, a construct popularized by Wing (2008, 2016), and how such thinking might interact with knowledge in particular domains. Recent work has argued for phenomenological (e.g., Farris et al., 2020; Sengupta et al., 2018) ways of framing computational thinking, and of situating computational thinking as practice-oriented rather than from a strictly cognitive perspective (e.g., Sengupta et al., 2013; Weintrop et al., 2016). Much of the recent work that examines computational thinking practices has focused on scientific modeling (e.g., Danish, 2014; Dickes et al., 2020; Farris et al., 2020; Sengupta et al., 2013; Wilensky & Reisman, 2006), and such studies reveal merits of computing in science classrooms. Here, I present results from a study involving undergraduate students' work within the discipline of mathematics (more specifically, within combinatorics, or the solving of counting problems) while they engaged in text-based programming in Python. I aim to provide insights into the ways in which computing (via phenomenological and dialogical perspectives on computational thinking) can support students in their combinatorial reasoning. Further, I seek to offer additional, unique insights into computational thinking given my particular disciplinary focus of combinatorics.

Overall, I attempt to highlight how enumeration, which focuses on determining the number of outcomes of counting processes, is especially well-suited for the use of computing. My goals are

CONTACT Elise Lockwood Elise.Lockwood@oregonstate.edu Department of Mathematics, Oregon State University, 338 Kidder Hall, Corvallis, OR 97331, USA.

to provide theoretical insight into how computational approaches can support students' enumeration thinking and activity, to support these theoretical ideas with empirical data, and to tie them to ongoing conversations about the phenomenological nature of computational thinking.

I draw upon results from two paired teaching experiments in which undergraduate students solved combinatorial tasks within the context of programming in Python. I explore the ways in which the practices of prediction and reflection and remixing code, along with the use of multiple representations, supported the students' combinatorial reasoning within this computational setting. I build on existing combinatorics education research (specifically Lockwood's [2013] model of students' combinatorial thinking) to frame the tasks developed in this study. While I focus on combinatorics, the results in this paper provide one example of how computational tasks and activities may be developed in a way that aligns with what we know about how students reason about and learn mathematical topics. In a field that is increasingly paying attention to the complexity of students' experiences of learning to code in disciplinary contexts, this work seeks to identify and answer epistemological questions related to students' combinatorial reasoning. That is, I seek to investigate how students' computational experiences interact with and support their reasoning about the particular mathematical domain of combinatorics. I attempt to address the following research questions:

1. In what ways can computational thinking practices facilitate certain desirable aspects of students' combinatorial thinking and activity?
2. What does this tell us about the nature of students' combinatorial reasoning and activity as it relates to computational thinking practices?

Literature review and theoretical perspectives

In this section, I present relevant literature and theoretical perspectives that frame this paper. In the next section, I offer a brief review of relevant prior literature on computing education in STEM. In doing so, I particularly highlight a phenomenological perspective on computational thinking, and I frame computational thinking practices within notions of representational heterogeneity and disciplinary reflexivity. I also describe the practices of prediction and reflection and of remixing, which were an integral aspect of the design of tasks and prompts in the teaching experiments. Finally, I focus on literature in the discipline of combinatorics education, and I elaborate Lockwood's (2013) model, which frames my theoretical perspective on combinatorial thinking and activity.

A brief review of prior work in STEM computing education

In this section, I briefly summarize current work in computing education, highlighting two aspects of existing literature that frame and motivate my study. First, I point to literature that proposes a phenomenological view (as opposed to a strictly cognitive view) of computational thinking, and I connect such work to the notion of representational heterogeneity. I then describe in some detail how I characterize two computational practices that emerged in my study—a cycle of prediction and reflection, and the practice of remixing code (e.g., Brennan & Resnick, 2012). Then, I highlight the theme of disciplinary reflexivity in computing education, noting prior literature that aims to explore and leverage practices that disciplines share in conjunction with computing.

A phenomenological view of computational thinking

Wing (2008, 2016) re-popularized the term *computational thinking*, offering broad descriptions and suggesting that it should be a construct to which more students gain exposure and

experience. Since Wing offered these ideas, authors have taken a variety of perspectives on computational thinking, and it has become a source of much interest and debate.[1] I highlight a trend that proposes emergent, alternative framings of computational thinking. In particular, researchers have viewed computational thinking not just as a cognitive construct, but they take a phenomenological approach to computational thinking (e.g., Dickes et al., 2020; Farris et al., 2020; Sengupta et al., 2018; Vieira et al., 2019). Sengupta et al. (2018) argued for "an epistemological shift from viewing coding and computational thinking as mastery over computational logic and symbolic forms, to viewing them as a more complex form of experience" (p. 3). This is in line with Papert's (1980) warnings against a technocentric perspective toward the use of computers in mathematics education—specifically, Papert warned that the work on the computer itself was not the point of the integration of computing. Rather, as Sengupta et al. (2018) emphasized, computing and computational thinking can and should be considered more broadly. Ultimately, Sengupta et al. (2018) argued for a move

> toward more phenomenological perspectives, both in terms of trying to understand and support the development of computational thinking as *experience* in the context of K-12 STEM education. Epistemologically, we have argued that computational thinking must be re-conceptualized more appropriately as an intersubjective experience, as opposed to a more cognitivist image of reasoning that can be assessed through the production of symbolic code. (Sengupta et al., 2018, p. 29)

This phenomenological view of CT, then, focuses on the experience of computational thinking, and it offers an approach that seeks to understand the experience of coding in context; this perspective values heterogeneity and challenges technocentrism.

There have been a number of studies in science and computing education that take a similar perspective toward computing and computational thinking. For example, Sengupta et al. (2013) worked in the context of having students engage with modeling in a middle school science classroom, and they proposed a framework for integrating computational thinking into K-12 science. They framed computational thinking as involving not just internal cognitive processes, but practices as well. In another study, Kafai (2016) argued that computational thinking "should be reframed as computational participation" (p. 26) and said that, "computational thinking is a social practice" (p. 26). This suggests that computational thinking can be viewed not just in terms of students' mental and cognitive actions, but in terms of broader practices in which they engage. In addition, Vieira et al. (2019) examined undergraduate engineering student's explanations by looking at their in-code comments. In this context, these authors were attending to students' discourse related to code—what they consider to be an explanation and how they think about and understand code. Focusing on such explanation practice belies a perspective of computing as encompassing a range of practices. In a study involving kindergarten and first-grade students studying honeybees, Danish (2014) attempted to move beyond attending to just one or two aspects of students' learning, instead using activity theory to attend to a complex system as a whole. He considered different kinds of representations, including computer simulations, drawings, and participatory representations to consider many factors at play in students' learning within this complex system. This suggests a perspective that does not emphasize a dichotomy of computational versus non-computational representations, but that embraces a complex system that may involve many practices and representations.

These researchers emphasized a view both that is different from a purely cognitive view of computational thinking and is distinct from a computer-centered technocentric framing. Such perspectives thus open up new possibilities for considering how practices, representations, and experiences may together inform students' computing, particularly in the context of another discipline. My work in this paper is situated within such a perspective, and I aim to contribute to the field's understanding

[1]Defining computational thinking is a complex endeavor—see, for example, Grover and Pea (2013) and Tedre and Denning (2016) for surveys and histories of the development of computational thinking in education.

of computational thinking practices within a mathematical (specifically, combinatorial) context. As much of this work has been done in with K-12 science students, often in the context of scientific modeling (e.g., Dickes et al., 2020; Farris et al., 2020; Sengupta et al., 2013; Wilensky & Reisman, 2006), I contribute to this body of literature by exploring phenomena related to students' computational thinking practices in a different setting and population of students.

In this paper, I focus on three particular aspects of computational experiences, and I demonstrate ways in which those aspects interact with and support students' combinatorial reasoning. These include representational heterogeneity, the fact that the computer facilitates feedback that fosters prediction and reflection, and reusing and remixing code. I elaborate these in the following sections, first addressing representational heterogeneity, and then discussing prediction and reflection and reusing and remixing code as computational thinking practices.

Representational heterogeneity. Previous work (such as that discussed above) has demonstrated that computing can be distributed across a range of computational and non-computational tasks. In light of this view, we can consider the use of representations across the many different ways in which computational thinking practices arise. This is related to a notion of *representational heterogeneity* (e.g., Dickes et al., 2020; Farris et al., 2020), in which students use and engage with multiple representations across computational and physical settings. Goldin (2014) defined mathematical representations as "… visible or tangible productions – such as diagrams, number lines, graphs, arrangements of concrete objects or manipulatives, physical models, mathematical expressions, formulas and equations, or depictions on the screen of a computer or calculator – that encode, stand for, or embody mathematical ideas or relationships." I adapt this slightly, to account also for mental representations and not necessarily external and concrete. Thus, in this paper, I take a representation to be *a mental or physical production that encodes, stands for, or embodies mathematical ideas or relationships*.

To demonstrate representational heterogeneity, in their work in middle school science classrooms, Farris et al. (2020) illustrated how fourth and fifth grade students' scientific modeling "was distributed across computational and physical representations" (p. 1334), and they contended that "attending to these forms of heterogeneity" (p. 1334) was and is important for understanding how relatively novice programmers can meaningfully engage with computational modeling as a means of doing scientific work. The authors discussed two cases involving students' work with scientific modeling (specifically in thinking about motion of various objects), which demonstrated students' rich engagement with multiple representations that included physical embodied activities, written drawings, paper-folding, drawn images and graphs, and block-based computer programming. They suggested that programming can be complemented by attention to alternative representations. As another example, Danish (2014) saw young students working with computational representations, representations that were drawn or written by hand, and what he called participatory representations, in which students enacted certain behaviors (in his case the behavior of honeybees); in Danish's study, these representations together were part of a complex system in which students engaged.

These multiple representations within their environment can not only support individual students' reasoning, but they can also allow for experiences that are not only cognitive. For example, such experiences may be social in nature (such as facilitating communication both among students and between students and instructors), embodied (physically manipulating or writing representations), or affective (feeling proud or confident about a certain representation). In light of the research that demonstrates representational heterogeneity, in the results of this paper I discuss various representations that students leveraged, noting that a variety of types of representations (computational, inscribed, verbal, mental) all contributed to the students' computational thinking and combinatorial reasoning within the setting of the study.

Computational thinking practices. Given that some researchers investigate computational thinking practices, as described above, I now operationalize some of the ways in which computational thinking practices are defined and used in the literature. The following quote by Sengupta et al. (2018) highlights this relationship between computational thinking as a cognitive perspective and as involving practice: "In this perspective, the term "thinking" in computational thinking is perhaps a semantic reduction of its intended meaning, because phenomenologically it involves both representational and epistemic work" (p. 6). As noted, Sengupta and colleagues have focused particularly on computational thinking practices related to abstraction, especially exploring and teaching such practices in K-12 science classrooms in modeling settings. Other researchers have presented practices that reveal the complex nature and possibilities of computational abstractions as experience. In particular, Weintrop et al. (2016) developed a "taxonomy of practices focusing on the application of computational thinking to mathematics and science" (p. 128). Weintrop and colleagues suggested ways in which teachers, administrators, policy makers, curriculum designers, and researchers might use the taxonomy. For education researchers, they noted that, "we view this work as both a theoretical and practical contribution to our understanding of the nature of science and mathematics education in our increasingly technological age" (p. 129). The very existence of this taxonomy of practices represents a perspective that such computational thinking practices are a way in researchers and other stakeholders may view, understand, and assess the construct of computational thinking.

In the study described in this paper, I draw on two of Weintrop et al.'s (2016) computational problem-solving practices. I focus on students' engagement with elementary Python programming, and I follow Weintrop et al. in their characterization of *programming*, which is a computational thinking practice that "consists of understanding and modifying programs written by others, as well as composing new programs or scripts from scratch" (p. 139). In my analysis and discussion, I also address Weintrop et al.'s practice of *troubleshooting and debugging.* They noted that "troubleshooting broadly refers to the process of figuring out why something is not working or behaving as expected" (p. 140) and noted that in computer science the activities involved in troubleshooting are called "debugging."

In considering assessment of computational thinking, Brennan and Resnick (2012) offered a framework that includes computational thinking concepts, computational thinking practices, and computational thinking perspectives. Among computational thinking practices, they listed practices they observed among students who were working with Scratch as being incremental and iterative, testing and debugging, reusing and remixing, and abstracting and modularizing (p. 7). I describe reusing and remixing in detail in a subsequent section as a feature of my students' work in this study. Brennan and Resnick's framework is another example of a study that adopts a broader view of computational thinking beyond just a cognitive perspective.

In addition to the use of multiple representations, there were two other practices in which the students in my study regularly engaged. These include the practice of engaging in a cycle of prediction and reflection, of remixing and reusing code. I briefly describe these in the following subsections.

Characterizing prediction and reflection. Throughout the study, as they programmed, the students regularly discussed their work with other each other and the researcher, and they were regularly asked for explanations, predictions, and reflections of their work. This kind of practice is in line with research discussed in Sengupta et al. (2018) and Veiera et al. (2020), in which students engage in discursive practice related to their computing. During the interviews, I regularly asked students to engage in prediction and reflection, and this was a fundamental aspect of the design of the study and the interview tasks. These constructs are central to the theoretical contributions of this paper, and so in this section I define how I characterize prediction and reflection.

In considering prediction among grade school students, Lim et al. (2010) noted that while prediction is clearly important, relatively little has been studied about prediction in mathematics education.

I follow Lim's (2006) characterization of prediction as "the act of conceiving an expectation for the result of an event without actually performing the operations associated with the event" (p. 103). Prediction thus entails anticipating and articulating the result of a particular event without first carrying out the event, and it was manifest in the interviews through conversation in which students elaborated their predictions by saying or inscribing them. While prediction could occur in a variety of contexts or situations, here I particularly mean predicting what the output of a counting process would be. In the setting in which my study took place, this meant predicting the outcomes generated by a counting process, both in terms of how the sets of outcomes would be organized (what output would be displayed) and in terms of how many outcomes appeared on the screen.

Reflection has a prominent role and rich history in education research, including work by Dewey (1910), who elaborated aspects of thinking, including reflection. According to Rogers (2002), Dewey outlined criteria for reflection, which involved viewing reflection as a holistic process that "is a means to essentially moral ends" (Rogers, 2002, p. 845) and "requires attitudes that value the personal and intellectual growth of oneself and of others" (Rogers, 2002, p. 845). While I acknowledge these aspects of reflection, I am particularly interested in the following criterion: "Reflection is a meaning-making process that moves a learner from one experience into the next with deeper understanding of its relationships with and connections to other experiences and ideas" (Rogers, 2002, p. 845). This view of reflection frames it as a process that allows for making connections and enriching and supporting conceptual understanding. Rogers notes that Dewey emphasized the importance of taking the time to reflect, where thinkers may explicitly slow down and pause activity in order to engage in reflection.

Reflection has also received significant attention in mathematics education research. In discussing mathematical problem solving, Wheatley (1992) said, "When solvers reflect on solution activity, they "distance" themselves from the activity and "hold the activity in thought" (Sigel, 1981). In this way, they make their activity an object which can be examined" (p. 536). I resonate with this characterization of reflection as a mental act in which students examine activity in which they have engaged. I also note that in my case, students could reflect not just on their activity, but also on the particular representations that emerged during that activity (so, students could reflect on an expression, a bit of code, the code's output, as well as on the activity that generated such representations and outputs). Ultimately, I consider reflection to be a mental act in which a person (here, a student) examines a mathematical concept, process, object, or activity. In my study, this typically meant that prediction and reflection were closely connected, as students often reflected on their prediction after they saw the actual output of a process they had predicted. Again, the reflection emerged via utterances and conversations between the students and the interviewers. This is in line with another criteria of Dewey, namely that reflection needs to occur "in community, in interaction with others" (Rogers, 2002, p. 845).

Reusing and remixing code. Another common practice that occurred in my study was that students reused and remixed code. Brennan and Resnick (2012) identified reusing and remixing as a practice among their students, noting that "[b]uilding on other people's work has been a longstanding practice in programming, and has only been amplified by network technologies that provide access to a wide range of other people's work to reuse and remix" (p. 8). Related to remixing and reusing is the idea of anchor code, which, according to Wagh et al. (2017), is "a body of code that creates a stable base from which further explorations take place" (p. 656). These authors examined seventh-grade students' programming in the context of biology, and they examined the extent to which students leveraged a bit of anchor code as they progressed through tasks. They found that "anchor code is evidence for conceptual learning and computational practices" (p. 656).

Reusing and remixing some fundamental bit of code is a documented computational thinking practice that can be productive for students, and the students in my study regularly drew upon this practice in their work. My students often copied and pasted existing code to make tweaks or

slight changes to it—sometimes it was code that was provided to them, and sometimes it was code that they had written themselves. As I will discuss in the results, this practice was useful and important both because it helped students who were perhaps less familiar with Python and its syntax, and because it highlighted what features of a segment of code are essential and what are more superficial (depending on the task at hand). In the results and discussion, I argue that even while they were remixing, the students were engaged conceptually both in their mathematical and computational thinking.

Disciplinary reflexivity. By focusing on computational thinking practices that exist in both computing and science, researchers highlight a motivation for disciplinary reflexivity, where common practices may be leveraged so as to benefit students' understandings in multiple disciplines. Increasingly, education researchers conduct studies that leverage connections (both in terms of concepts and practices) between computing and a given discipline.

Researchers have investigated whether computing can complement and support work in a given discipline, and whether, reflexively, certain disciplinary ways of thinking or practices can complement support computing work (e.g., Guzdial, 1994). We have seen this in a variety of STEM disciplines, including science broadly (e.g., Dickes et al, 2020; Hambrush et al., 2009; Sengupta et al., 2013; Sengupta et al., 2018; Weintrop et al., 2016), biology (e.g., Danish, 2014; Wagh et al., 2017; Wilensky & Reisman, 2006), physics (Caballero, 2015, 2019; Odden et al., 2019; Young et al., 2019), engineering (e.g., Magana et al., 2013, 2016; Vieira et al., 2019), and mathematics (e.g., Broley et al., 2018; Buteau et al., 2015, 2016, 2020; DeJarnette, 2019; Francis & Davis, 2018; Gadanidis et al., 2017; Lockwood et al., 2019; Lockwood & De Chenne, 2020, 2021). There are also researchers investigating connections to non-STEM fields such as social studies (e.g., Hammond et al., 2019; Naimipour et al., 2019) and art (e.g., Knochel & Patton, 2015). Such initiatives involve a myriad of different approaches, settings, and participants. In much of this work, researchers have shown that such integrated, disciplinary approaches to having students engage with computing can be beneficial and productive, and that students do indeed benefit in terms of their understanding of and engagement with computing and also with regard to specific disciplinary concepts and practices. Weintrop et al. (2016) particularly emphasize the disciplinary focus of their taxonomy of computational thinking practices, noting that their approach to defining computational thinking

> takes the form of a taxonomy of practices focusing on the application of computational thinking to mathematics and science. This approach employs mathematics and science as meaningful contexts in which to situate the concepts and practices of computational thinking and draws on the ways mathematicians and scientists are using computational thinking to advance their discipline. (p. 128)

Computing in mathematics education research. Given my focus on mathematics, I briefly elaborate some of the existing work on computing in mathematics education so as to situate my work on computing within the discipline of mathematics. There is a rich tradition of using a computational setting to study mathematical understanding, which traces back to Papert's use of Logo to teach young children (e.g., Feurzeig et al., 2011; Papert, 1980). There has been an increasing amount of research aimed at studying the role of computational thinking and activity within mathematics education. These studies extend beyond examinations of technology and focus particularly on computing and programming (e.g., DiSessa, 2018; Kotsopoulos et al., 2017; Sinclair & Patterson, 2018). They cover a variety of mathematical domains and range from investigating young students (e.g., Benton et al., 2017, 2018; Hoyles & Noss, 2015; DeJarnette, 2019; Francis & Davis, 2018; Gadanidis et al., 2017) to undergraduates (e.g., Lockwood & De Chenne, 2020, 2021; Buteau & Muller, 2017) and mathematicians (e.g., Broley et al., 2018; Lockwood et al., 2019).

In light of this perspective of disciplinary reflexivity, I now spend some time focusing on the prior work and constructs within the discipline of combinatorics, which is the focus of this study. One of the goals of this paper is to offer a theoretical account for why certain aspects of

computing are particularly well-suited to bring out key features of combinatorial thinking and activity. To do this, I need to go into some detail about findings in combinatorics education.

Combinatorial problems are rich and accessible but are difficult to solve correctly

The importance of teaching and learning combinatorics in K-12 and undergraduate mathematics curricula is well established in the mathematics education literature (e.g., Batanero et al., 1997; Eizenberg & Zaslavsky, 2004; English, 1991, 1993, 2005; Hadar & Hadass, 1981; Kapur, 1970; Lockwood, 2011, 2013, 2014; Lockwood et al., 2015; Maher et al., 2011; Tillema, 2013). In addition to their practical applications, such problems are remarkable because they are accessible to students at a variety of levels, allowing students to engage in meaningful exploration even with relatively little mathematical background (e.g., Kapur, 1970; Maher et al., 2011).

Unfortunately, although combinatorial problems are accessible, applicable, and have potential for developing rich mathematical reasoning, students at all ages struggle to solve them correctly (e.g., Annin & Lai, 2010; Batanero et al., 1997; Lockwood, 2014). Several studies support this claim by reporting low overall success rates (below 50%) for undergraduates solving basic combinatorial problems (e.g., Eizenberg & Zaslavsky, 2004; Lockwood & Gibson, 2016). Researchers have suggested reasons for such struggles; for example, Eizenberg and Zaslavsky (2004) pointed out that because of the nature of combinatorial problems and their very large numerical answers, such problems are difficult to verify.

Researchers have investigated students' combinatorial thinking in a variety of ways over previous decades, with attempts to help students improve their success at solving combinatorial problems. This has included examining phenomena such as: potential errors and pitfalls (e.g., Batanero et al., 1997; Hadar & Hadass, 1981), students' listing strategies (e.g., English, 1991; Lockwood & Gibson, 2016), and students' reasoning about combinatorial concepts, including permutations, combinations, the multiplication principle, and combinatorial proof (Lockwood et al., 2017, 2021; Maher et al., 2011; Reed & Lockwood, 2021; Tillema, 2013; Tillema & Gatza, 2016). From this corpus of work, I focus on one particular result that has been established in the literature: students benefit from focusing concretely on sets of outcomes as they solve combinatorial problems. This notion of sets of outcomes was initially elaborated in Lockwood (2013) and further explicated in Lockwood (2014) and Lockwood et al. (2015). I discuss this work in the following section.

Lockwood's (*2013*) model of students' combinatorial thinking

In this section I elaborate Lockwood's (2013) model of students' combinatorial thinking as a key part of the literature and as my theoretical perspective toward counting in this paper. It is important to emphasize that this model was not initially conceived within the context of programming or computing. Lockwood's (2013) model frames students' combinatorial thinking in terms of three key components: *Formulas/Expressions*, *Counting Processes*, and *Sets of Outcomes* (Figure 1). I briefly characterize each of the components.

Counting processes

Lockwood (2013) defined counting processes as referring to "the enumeration process (or series of processes) in which a counter engages (either mentally or physically) as they solve a counting problem. These processes consist of the steps or procedures the counter does, or imagines doing, in order to complete a combinatorial task" (p. 253). As examples, Lockwood offered the implementation of a case breakdown or successive applications of the multiplication principle. For instance, consider the Outfits problem: *A shirt-pant outfit consists of a shirt and a pair of pants. Given a set of 3 different shirts and a set of 2 different pairs of pants, how many total shirt-pant*

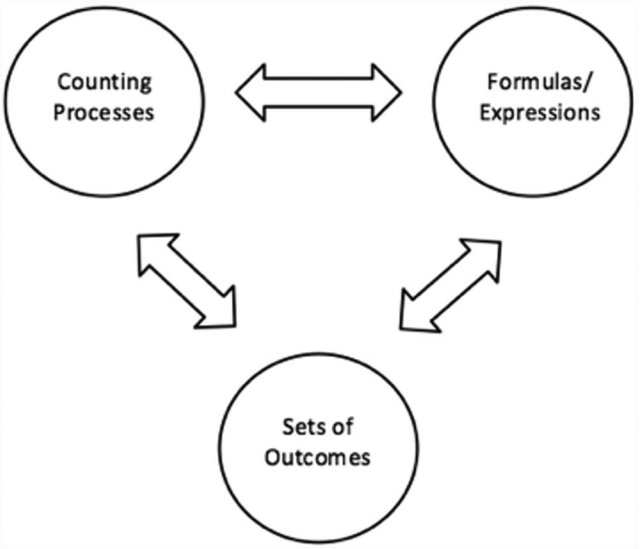

Figure 1. Lockwood's model of students' combinatorial thinking (as presented in Lockwood et al., 2015).

outfits are possible? We can imagine a process to solve this problem, in particular beginning with a shirt, and pairing each shirt with a pair of pants. Since each shirt can be paired with two different pairs of pants, we get $3 \cdot 2 = 6$ total outfits. This is a version of an *odometer strategy* (English, 1991), which involves, "holding one item constant while systematically varying each of the other items" (English, 1991, p. 451).

When someone engages in a counting process, they commonly enact a sequence of steps or stages, each of which contributes to some part of a composite outcome. This is typically (though not always) closely related to the multiplication principle, where we multiply the number of available independent options at each stage of the process together. Lockwood et al. (2017) raised this idea when they discussed operational statements of the multiplication principle, which characterize the multiplication principle as "counting ways to complete a process (without specifying the structural outcomes of that process)" (p. 392). Counting processes tend to involve the carrying out of a series of steps, and, as Lockwood et al. (2017) indicate, if there is also one-to-one correspondence between ways of completing the process and distinct composite outcomes, then we can conceive of counting ways of completing a process as giving all possible outcomes that satisfy given constraints (thus providing "the answer" to a counting problem).

Sets of outcomes

Sets of outcomes "refer to the collection of objects being counted – those sets of elements that one can imagine being generated or enumerated by a counting process" (Lockwood, 2013, p. 253). Sometimes outcomes can be relatively simple and straightforward to describe or represent, but sometimes they require some additional bit of encoding. For instance, in the PIN problem, *How many 5-digit Personal Identification Numbers are there, using the digits 0 through 9 where repetition is allowed?,* the outcomes are clearly simply 5-digit numbers. However, in the Groups of Students problem, *How many ways are there to split a class of 20 students into four unlabeled groups of five people?*, the outcomes are now partitions of 20 people into four groups of five, which may be a bit more difficult to describe and represent. In each case, though, the cardinality of the set of outcomes provides the answer to the counting problem. Lockwood uses the term "outcomes" intentionally to reflect that they are the outcome of a process. Lockwood and colleagues (e.g., Lockwood, 2013;

Lockwood & Gibson, 2016) have previously made the case for the importance of outcomes and having students focus on sets of outcomes.

Formulas/expressions

Formulas/expressions refer to mathematical expressions that yield a numerical value, and Lockwood (2013) considers "two expressions to be different if they differ in form" (pp. 252–253). These expressions are combinations of numbers, variables, and operations, and Lockwood has previously emphasized that these often reflect counting processes or ways in which sets of outcomes are structured. For instance, consider the Heads and Tails problem: *How many outcomes are there for flipping a fair coin seven consecutive times?* We can conceive of a solution to this problem that has seven independent stages for each consecutive coin flip, and because there are two options for each stage, we could employ the multiplication principle to generate an expression of 2·2·2·2·2·2·2. Such an expression is, on the one hand, just a number (128), but it can also represent a 7-stage process, where the number of options at each stage in the process is 2. While an actual numerical value (such as 128) can be viewed as a mathematical expression within the formulas/expressions component, Lockwood has not typically emphasized the numerical values, focusing instead on what expression generates the numerical value, such as 2·2·2·2·2·2·2.

Relationship among counting processes and sets of outcomes

Lockwood has previously argued that the relationship between sets of outcomes and counting processes is an essential aspect of successful counting (Lockwood, 2013; 2014; Lockwood et al., 2015). Specifically, she has described a set-oriented perspective toward counting, which is "a way of thinking about counting that involves attending to sets of outcomes as an intrinsic component of solving counting problems" (Lockwood, 2014, p. 31). Lockwood found that this relationship between counting processes and sets of outcomes is important for two reasons. First, when focusing on this relationship, students have to consider the effect that their counting processes have on the set of outcomes. By linking a counting process to an outcome, one has to be explicit about how outcomes are being generated and structured. This reduces instances of answers that result from a reliance on formulas that are not well-understood and can help attune students to instances of overcounting. Second, a focus on this relationship affords students greater flexibility when they approach combinatorial problems. Such flexibility can engender meaningful and effective combinatorial problem solving.

In terms of the model, then, one way to frame students' difficulties with counting is that students do not clearly connect their counting processes with the outcomes they are trying to enumerate (Lockwood et al., 2015). Thus, a possible avenue for improving students' combinatorial problem solving is "to reinforce the relationship between counting processes and sets of outcomes, and to help students integrate the set of outcomes as a fundamental aspect of their combinatorial thinking and activity" (Lockwood, 2014, p. 36). In the remainder of this section, I describe how the activity of listing can help to build this relationship, and I also discuss limitations of by-hand listing as a motivation to leverage computation.

Researchers have previously hypothesized that having students create lists of outcomes could be useful for students' successful counting. In particular, in Lockwood and Gibson's (2016) study, they coded undergraduate students' solutions to counting problems as correct or incorrect and as involving *no listing*, *articulation*, *partial listing*, or *complete listing*.[2] The quantitative results showed that listing behavior (taken as partial and complete listing) was positively correlated with correctly answering counting problems. This prior work suggests that the activity of listing has the potential to strengthen

[2]A *partial listing* code was given "if there was some evidence that the student created a list or partial list of the outcomes, but they may have not written the entire list correctly or may have truncated their listing when they identified a pattern" (p. 254). A *complete listing* code was given if "a student provided a complete, correct list of outcomes" (pp. 254–255).

the important relationship between counting processes and sets of outcomes, and thus to serve as an avenue by which students can ultimately solve combinatorial problems more successfully. However, even though listing is a potentially valuable combinatorial, solutions to combinatorial problems can be intractable to list by hand but be tractable for a computer (such as listing all 128 outcomes of flipping a coin 7 times, for example). It is often not feasible for students to generate complete lists of outcomes by hand. Partial listing can be beneficial but also has limitations, as patterns do not always extend to all cases, and students may not detect subtle errors when they attempt to generalize. Furthermore, when students partially list, they may think that they understand how a process continues, but they may not be precise in articulating the full process to completion. So, even if it seems like one knows how a process is completed, if one doesn't have to actually list all outcomes, those details can get lost and glossed over (e.g., Lockwood & Gibson, 2016; Lockwood & Purdy, 2019).

Thus, there is a dilemma—we know that listing can be valuable, but by-hand listing has drawbacks. This leads to a question of how we can move past limitations of by-hand listing in order to facilitate generating lists in more complex problems and contexts. Fortunately, there is a natural answer to this question: we can leverage computational activities, allowing students to reap benefits of by-hand listing by designing algorithms and computer programs to enumerate lists. One primary motivation for the study described in this paper, then, is to demonstrate that computational settings can provide natural opportunities for students to strengthen the relationship described in Lockwood's (2013) model between counting processes and sets of outcomes, which in turn can enrich students' combinatorial reasoning and activity. In this way, the model can offer a discipline-specific combinatorial framing for how and why computing might serve to reinforce important combinatorial concepts and practices.

Methods

In this paper, I offer a theoretical argument that is supported by empirical data. For the sake of space, I provide a brief description of the data that I use to exemplify and support my main ideas. For more details on the methods, see Lockwood and De Chenne (2020).

Data collection and analysis

Participants and data collection

I present data from two teaching experiments [(TE), in the sense of Steffe & Thompson, 2000] with undergraduate students at a large public university. The TE methodology allows for researchers to investigate students' reasoning over time, giving an in-depth look at students' thinking about particular mathematical concepts. To identify participants for the teaching experiments, the research team conducted individual selection interviews with 13 students who were recruited from a vector calculus class. I did not want to interview experienced counters who might recall formulas or combinatorial ideas, so I recruited students with little combinatorial experience.[3] I also asked students about their previous programming experience. From the selection interviews, I selected three pairs of students to participate in multi-session interviews; I report on two of them here both due to space, and because the third pair was limited in their scheduling availability and did not engage with the same tasks as the other pairs. I paired students with similar levels of programming experience to avoid one student with more experience dominating the interviews. My main goal in reporting on two pairs of students is not to compare and contrast them. Rather, I bring up

[3]In the selection interviews, I asked what classes they had taken in high school and college. I also showed the students a list of several symbols, some of which could be associated with combinatorics. I asked students if they had seen each symbol before and, if so, what they thought the symbol meant. This gave insight into students' familiarities with combinatorial formulas. I also had students solve several basic counting problems, which further indicated whether they were trying to recall formulas and how they were reasoning about these problems.

examples from each pair to exemplify relevant phenomena, and in some cases, their respective work on the problems together illustrates certain points. Notably, in prior work involving combinatorial tasks, I have often interviewed vector calculus students, finding that they are at an appropriate level mathematically but are not overly familiar with combinatorial concepts (see, e.g., Lockwood & Purdy, 2019; Reed & Lockwood, 2021). My intent is not to make general claims about all students, or even all vector calculus students, but rather I aim to demonstrate phenomena that inform the nature of combinatorics and computing. I hypothesize that other students with different mathematical backgrounds may also benefit from computing as these students did.

Pair 1 consisted of two female students (pseudonyms Diana and Charlotte) who were first- and second-year chemistry majors, respectively. Diana and Charlotte had not taken a combinatorics or discrete mathematics course in college, and selection interviews revealed that they did not have familiarity with basic counting formulas or symbols. They stated that they had no programming experience whatsoever prior to the interviews.

Pair 2 consisted of two male students (pseudonyms CJ and Corey) who were first-year engineering majors. Neither CJ nor Corey had taken a combinatorics or discrete mathematics course in college. During selection interviews, CJ did not recall formulas; Corey mentioned occasionally that he had seen a combination and permutation formula before, but he engaged in problem solving that suggested he was not recalling those formulas. CJ had taken an html class in high school, and Corey had taken an Advanced Placement CS course in high school and was enrolled in a MATLAB for engineers course at the time of the interviews. However, both CJ and Corey stated that they did not consider themselves to be strong programmers.

During both of the TEs, the students worked at a large desktop computer in my (the interviewer's) office. Understanding that agent-based and block-based programs are used in much of computing education literature (e.g., Farris & Sengupta, 2014; Sengupta et al., 2013; Wagh et al., 2017; Wilensky & Reisman, 2006), I chose Python because it is a relatively straightforward language that is used in school and work contexts. I wanted to use an existing language rather than developing my own language with the hope that these tasks may eventually be more widely used, and I also chose to prioritize the authentic experience of an existing text-based language over other affordances of block-based or task-specific programming languages (e.g., Guzdial & Naimipour, 2019). In addition, there are many free resources for Python-based programming.

I generally kept to simple, fundamental elements of coding, such as nested for loops and conditional if statements, which are not specific to Python. The students used the programming environment PyCharm, which allowed them to type and edit code into a window. When they ran the code, the output would appear in an adjacent window. The environment thus enabled the students to look at and reflect upon the output of their code—they could scroll through the outputs they produced. I presented the tasks and prompts both on paper and in the PyCharm environment. I use the term *computational setting* to describe this particular setting in which the students in my study were working. Specifically, they had access to a computer in the form of PyCharm and programming in Python, and they also could use resources like paper and writing implements, and they could discuss problems with another student. I distinguish between this setting and a setting in which students do not have any access to a computer as they solve tasks. I do not use this phrase to suggest a dichotomy between computational and non-computational environments, but rather to specify the interview setting for these students that involved a number of tools and resources, including a computer.

The interviews were videotaped and audiotaped, and the research team made video captures of the screen. We created spliced videos that showed both the students and their computer screen. In concert with a perspective of representational heterogeneity, within each pair, both students took turns typing and coding, and they also regularly wrote down inscriptions and diagrams on paper as they worked. Further, they were engaged and worked together, regularly talking with each other as they reasoned about the problems. The TEs were staggered and both occurred

during the same academic term.[4] Generally, during the interviews there was a main interviewer and another interviewer running the camera; both interviewers regularly asked students to explain their reasoning, to elaborate ideas, and to engage in prediction and reflection. I consider the students' work during the interviews to be reflective of computational thinking, which encompassed work on the computer, by-hand inscriptions, verbal utterances, and interactions between the researchers and among the students.

Data analysis

All of the interviews were transcribed, and the research team created enhanced transcripts that included images from student work and the computer screen. To analyze data for this paper, I re-watched all of the spliced videos in both TEs, making notes on the enhanced transcripts of episodes that shed light on the interaction between the computational setting and the students' combinatorial reasoning in terms of Lockwood's (2013) model. I particularly highlighted episodes that (a) indicated a connection between components of the model, or (b) involved prediction and reflection. I selected several episodes from each TE that I thought would be good candidates to explore in this paper, which gave evidence of students' engaging in prediction and reflection, especially where the students related components of the model. I flagged those episodes and put them in a master document, and then I revisited each of them and further analyzed interplay between elements of the model and the activities of prediction and reflection. This analysis involved reviewing enhanced transcripts and taking notes about what ideas and relationships were occurring.

I then drafted descriptions of counting processes, sets of outcomes, and formulas/expressions, and relationships among them, using episodes from the master document to inform how these components related to the practice of computing. Through this process, I reexamined the Outfits problem, which raised a number of interesting issues that related to the research questions. I reviewed both pairs of student work on the Outfits problem and wrote up analytical descriptions of the problem in terms of their combinatorial reasoning (in terms of Lockwood's model) and the computational representations. Having fleshed out student work on the Outfits problem, I identified data excerpts in the License Plate and ROCKET problems that highlighted two other noteworthy phenomena (specifically, an instance when students tried out two different processes, and an instance when students reviewed the entire set of outcomes). Further reflection and reviewing the data also revealed the Coin Flips problem as a noteworthy episode. Through these episodes, I articulated ways in which the practices of prediction and reflection served to reinforce connections for students among components of Lockwood's model as the students engaged in computing.

Tasks and instructional interventions

Due to space, I focus on four specific tasks that the students solved. I discuss how specific instructional prompts aimed at fostering prediction and reflection served as an important element of the tasks and instruction in the TE, contributing to the representational heterogeneity that emerged in the students' work. Indeed, the tasks, being situated in a setting in which students had programs and the computer available to them, afforded students with coding experiences.

[4]Pair 1 started 2 weeks before Pair 2, which allowed for some refinement of tasks at the beginning of the teaching experiment with Pair 2. Pair 1 met for a total of 16 h, spanning 7 weeks, and Pair 2 met for a total of 12.5 h, spanning 4 weeks. Specifically, Pair 1 met over 11 sessions that were each 60–90 min long over the course of 7 weeks. Pair 2 meet over 9 sessions that were each 60–90 min long over the course of 4 weeks.

Tasks

The students solved a sequence of counting problems, and these tasks broadly followed a progression of combinatorial concepts. They solved basic Cartesian product problems, and then problems involving arrangement with unrestricted repetition. They then worked on problems involving basic operations of multiplication and addition, arrangement without repetition (or permutation) problems, and then selection without repetition (or combination) problems. Finally, they solved problems that targeted particular elements of encoding, such as distribution problems.

I offer data from student work on the Outfits problem, the License Plate problem, and the ROCKET problem, and the Coin Flips problem. Tables 1–4 show the statements of problems I will discuss in the results. Note that within these problems, there are a variety of different specific activities in which students may engage, including reading, interpreting, or modifying code that is provided to them, and creating their own code either from scratch or by reusing and remixing other code. This variety highlights the overall range of experiences the students were able to have as they coded. Some tasks included additional activities like evaluating a given output and writing code that would generate that output.

Prompts for prediction and reflection

In addition to these task statements, an important aspect of the instruction involved me regularly prompting the students to engage in activities of prediction and reflection. As noted, this reflects a more dialogical perspective of computational thinking, where utterances and discussion may reflect students' computational thinking. During the course of the TE, I regularly and explicitly asked the students to predict what they thought the output of the code (the set of outcomes) would be. Often this included predicting a total number they expected to see, so they often predicted a mathematical expression as well. Eventually this became a norm in the interviews, and both pairs came to anticipate this question and would suggest their predictions before they were asked. Similarly, I tried to establish a norm of having students reflect on their work, particularly asking questions in which they looked across representations (such as the code, mathematical expressions, the sets of outcomes, etc.) both to see if their results made sense and to make connections among these representations.

Table 1. All parts of the Outfits problem (we focus particularly on Parts 1a, 1d, and 2b)

1a) Given a set of shirts and a set of pants we would like to know the total type and number of outfit combinations possible. Look at the code below. What do you think this code does? What will the output of this code be?

```
Shirts = ['tee','polo','sweater']
Pants = ['jeans','khaki']

outfits = 0
for i in Shirts:
    for j in Pants:
        print(i,j)
        outfits = outfits+1
print(outfits)
```

1b) Why is one of our print statements located on the inside of the loops and the other on the outside?

*1c) What would happen if we put the **print(outfits)** statement inside the for loops? First make a guess and then try changing the code. Discuss the results.*

1d) How would your program change if you wanted to print the outfits so the pants are before the shirts?

2a) Create some new code that would use 5 different pieces of clothing to create an outfit. How many outfits have you created? What is a mathematical expression that represents the number of outputs of your code?

2b) Create some new code that would use 3 loops and have 24 different outfits total. Explain what you did. What is a mathematical expression that represents the number of outputs of your code?

*2c) Can you write some code (still using the outfits context) that would reflect an expression of 2*3*5*6? Why did you write the code as you did?*

Table 2. Parts a and b of the License Plate problem (we focus on 4a in this paper).

4a) A license plate consists of six characters. How many license plates consist of three numbers (from the digits 0 through 9), followed by 3 lower case letters (from the first 5 letters in the alphabet), where repetition of characters is allowed? What is a mathematical expression that represents the number of outputs of your code?

4b) How many license plates consist of 3 upper case letters (from the first 5 letters in the alphabet) followed by three numbers (from the digits 0 through 9), where repetition of characters is allowed? What is a mathematical expression that represents the number of outputs of your code? How does this relate to the mathematical expression in 4a?

Table 3. Arrangement problems, including the ROCKET problem (we focus on 1d in this paper).

1a) Consider the problem, "How many ways are there to rearrange 5 people: John, Craig, Brian, Angel, and Dan?" Below is some code that counts the number of arrangements. What does this code do? Note, the ! symbol means "not," so j != i means "j is not equal to i."

```
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)
```

1b) 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?

1c) Here is some code output using the numbers 1–5. Can you find a way to correlate each of these outputs with a unique arrangement of the letters in PHONE? What would the code look like?

```
(1, 2, 3, 4, 5)
(1, 2, 3, 5, 4)
(1, 2, 4, 3, 5)
(1, 2, 4, 5, 3)
(1, 2, 5, 3, 4)
(1, 2, 5, 4, 3)
(1, 3, 2, 4, 5)
(1, 3, 2, 5, 4)
…
(5, 3, 2, 1, 4)
(5, 3, 2, 4, 1)
(5, 3, 4, 1, 2)
(5, 3, 4, 2, 1)
(5, 4, 1, 2, 3)
(5, 4, 1, 3, 2)
(5, 4, 2, 1, 3)
(5, 4, 2, 3, 1)
(5, 4, 3, 1, 2)
(5, 4, 3, 2, 1)
120
```

1d) Now can you adjust the code from PHONE to list the number of arrangements of the letters in the word ROCKET? Can you list it both as numbers and as letters? How many arrangements to you expect to get, and why do you think that? How will the list of outcomes be structured?

Results

I present data from one or both pairs of students on four tasks, and I demonstrate how these tasks facilitated student engagement with key combinatorial concepts. I argue that the fact that the students were engaging in the experience of computing gave them access to particular computational representations of combinatorial concepts, and that these representations, along with prompts to have students predict and reflect, afforded opportunities for students to interact richly and deeply with the combinatorial content. Through the students' work on these tasks and

Table 4. The coin flips problem (we focus on 5a in this paper).

5a) Write a program to list (and determine the total number of) all possible outcomes of flipping a coin 7 times.
5b) Your friend Shelly can't decide whether a mathematical equation to solve this problem is 2^7 or 7^2. What do you think is right and why?
5c) Suppose you flip the coin an 8^{th} time. How does this change your code? How does this change your outcomes? What is the total number of possible outcomes now?

activities, we gain new insight into the nature of students' combinatorial thinking and activity as it relates to computing.

Introduction to syntax and combinatorial concepts through the outfits problem

I begin by presenting each pair's work on the Outfits problem, which was the first task the students were given that involved the computer. This problem had a number of parts (all of which are presented in Table 1), and I focus on Parts 1a, 1d, and 2b. I aim to demonstrate what it was like for students to interact with the computational interface of Python programming in PyCharm, showing that the syntax was not an overwhelming barrier even for Diana and Charlotte, who had no prior programming experience. I also highlight that prediction and reflection were computational thinking practices in which the students engaged even from the very beginning of the interviews.

Outfits problem part 1a—an introduction to syntax via prediction and reflection

I begin with Charlotte and Diana's initial reasoning about Part 1a of the Outfits problem, which says: *Given a set of shirts and a set of pants, we would like to know the total type and number of outfit combinations possible. Look at the code below [Figure 2a]. What do you think this code does? What will the output of this code be?* This is a case where the lists of outcomes are not particularly large or complex, but still there were opportunities for students to develop their combinatorial reasoning.

The output of the code is given in Figure 2b. After the students ran the code, I asked them to reflect on the output of the code (the first underlined portion of the excerpt below). In second underlined portion, I asked questions that directed the students toward not just the list of outcomes, but also toward how the computer (via the code) would actually enact a process to list the outcomes. This was my implicit attempt to emphasize a relationship between counting processes and sets of outcomes for the students, and I wanted to understand how and why they solved and reasoned about the problem.

(a)

```
Shirts = ['tee','polo','sweater']
Pants = ['jeans','khaki']

outfits = 0
for i in Shirts:
    for j in Pants:
        print(i,j)
        outfits = outfits+1
print(outfits)
```

(b)

```
tee jeans
tee khaki
polo jeans
polo khaki
sweater jeans
sweater khaki
6
```

Figure 2. (a,b) Code given in Part 1a of the Outfits problem, and the output of the code.

Interviewer:	… What are you interpreting as what came out of there?
Diana:	So, the number on the bottom says 6, so I think that just tells us the total amount of outfits there are, and then the 6 above outfits, are the actual possibilities.
Interviewer:	Okay. Great. And why? Does it make sense that it listed them in that particular order?
Charlotte:	Yes, because with shirts it went from tee to polo to sweater. So, it does go from tee to polo to sweater, and same with the jeans and khakis.

This initial problem reinforced for the students that the code did indeed give a precise list of outcomes. That is, when Charlotte ran the code she received confirmation not only that the outfits would be printed, but, even more, that they would be printed in a particular way. In the next excerpt, I drew their attention back to the code and its relationship to the organization of the outcomes. By directing them to these representations of code and outcomes, I attempted to help students formulate connections among their respective representations, and, thus, combinatorially, among components of the model. This set the stage for representational heterogeneity and connections among representations that would occur throughout the interviews.

Interviewer:	What about the code do you think reflects that way of organizing the outcomes? Like, why are the t-shirts grouped together for example.
Charlotte:	I think maybe because it says, "print i, j," that it will just start with the same shirt, and then do all the possibilities of j, and then it will move on to the next i, the next shirt, and do all the possibilities of j.

I emphasize that even though this was Charlotte and Diana's very first exposure to code, the syntax and representations were not barriers that prevented them from reasoning about the problem. Rather, the experience of engaging with representations of both code and output fostered initial connections between counting processes and outcomes.

In CJ and Corey's work on Part 1a of the Outfits problem, we gain insight into how the code could reflect a particular process. CJ and Corey were more experienced coders, and we reasonably see different kinds of responses from them. Indeed, they seemed to understand the code as essen-

Corey:	Uh, so, alright. It looks like it's taking – starting with the shirts it goes through each shirt. So, it goes to i in shirts so i would be the place in this set of shirts. So, we'll start with tee, and then it'll go through another for loop which is basically saying for tee it will match up each pair of pants with that tee and then print out that value, so like tee, jeans. And then it will add an outfit. And then it will keep going until all the shirts and pants have been matched up.

tially representing a counting process. In their first attempts at solving the problem, Corey described the code as completing steps or stages in a process.

Corey:	I think it will print each pair of outfits. So, it will print tee jeans, tee khaki, then polo jeans, then polo khaki, then sweater jeans, then sweater khaki, and then at the end it will print how many outfits there were.
Interviewer:	Okay, great. And how many do you think it will be?
Corey:	Six.

I then asked what output they would expect to see when they ran the code. Corey gave the following response, and CJ seemed to agree with this thinking.

CJ:	So, I guess the i for the shirts is listed before the j in pants. So, when it says for i, and the first one in i is tee, and then it will go through all the j, tee jeans, tee khaki. And then it'll just move on to the next list in i, or the next value in i.

They then ran the code and got the output they expected, and I asked why the output was structured as it was (prompting the students to engage in reflection). In CJ's response, he described the listing process that he felt the code represented.

CJ and Corey's comments suggest that they could interpret the code on the screen as involving some process that would go through and list (through printing) the outcomes in a particular way. This episode thus highlights what it could mean for code to represent a counting process. My question of, "When we run it, what do you think we'll see on this screen?" shows that I was asking for prediction, and, in terms of the model, I attempted to draw attention to the idea that a counting process would generate a set of outcomes. I acknowledge that the set of outcomes for this problem is very small, and one could easily generate the entire list without the computer. However, this episode represents an initial instance of students articulating a counting process in terms of a computational representation (the code itself) and then explicitly connecting it to the set of outcomes (the output of the code). This use of prediction and reflection, as well as generating and reflecting upon output is not unique to my study, and computing education researchers have argued for such features and affordances of the computer. But here we see that this computational thinking practice supported students' combinatorial reasoning—specifically connecting counting process and sets of outcomes.

Outfits problem part 1d—connecting counting processes and lists of outcomes

Part 1d of the Outfits problem referred to the code from Part 1a, and asked, *How would your program change if you wanted to print the outfits so the pants are before the shirts?* Each pair made slightly different adjustments to their code from Part 1a, and, in each case, the code generated a list of outcomes they did not expect. These unexpected lists, along with prompts to predict and reflect, fostered rich discussion with each pair of students about their code and the outcomes, and I argue that this experience facilitated discussions about, and implicitly reinforced, the fundamental combinatorial notion that counting processes generate sets of outcomes that are organized in particular ways.

To solve Part 1d of the Outfits problem, Diana and Charlotte engaged in remixing code (in the sense of Brennan & Resnick, 2012), as they began by swapping the i and j in the print statement in their code (Figure 3a)—this is a reasonable and correct way to print pants before shirts. In the excerpt below, I asked them to predict what the output would be, and Diana suggested

Diana:	So, I think you would just swap the, i, and j in the print, so that way the j comes first, instead of the i [Figure 3a].
Interviewer:	Okay. Okay. Great. And so, what do you think your output would look like? So, go ahead and switch it, we'll run it in a second, but what do you think the outcomes are gonna look like?
Diana:	I think that it would put like all of the jeans first, with like the shirt combinations, and then all the khakis first.
Interviewer:	Great. And what, um, what order do you think the shirts will be in?
Diana:	Like tee, polo, sweater.
Charlotte:	Yeah.
Interviewer:	Oh, so you can try it.
Diana:	Oh, it didn't do that.
Charlotte:	Oh. So, it still did jean, khaki, jean, khaki. Interesting.
Diana:	Do you think, Charlotte, that that might be because it has the for i in loop first? Like because the i is first there, and then the j is the next one in.
Charlotte:	That makes sense, yeah.

(a)

```
Shirts = ['tee','polo','sweater']
Pants = ['jeans','khaki']

outfits = 0
for i in Shirts:
    for j in Pants:
        print(j,i)
        outfits = outfits+1
print(outfits)
```

(b)

```
jeans tee
khaki tee
jeans polo
khaki polo
jeans sweater
khaki sweater
6
```

Figure 3. (a,b) Charlotte and Diana's code and output for switching the variables in the print statement.

that the code would print all of the jeans and then all of the khakis, where each shirt would be paired with each type of pants. The students then ran the code (yielding the output in Figure 3b), and in the underlined portion they appeared surprised by their output.

Diana and Charlotte's approach was reasonable—if the initial code that printed (i,j) listed groups of shirts first, it would stand to reason that switching the variables in the print statements would simply print groups of pants first. However, the nested for loops structure of the code meant that the shirts were still being cycled through before the pants, and so the code yields output in Figure 3b. The students were not wrong in that they did in fact produce outcomes with pants first, but they were clearly surprised by what was printed (based both on their prediction, and the fact that they said, "Oh, it didn't do that."). Diana's comment to Charlotte explaining why it printed like it did (the second to last line in the previous excerpt) suggests that she hypothesized that the order of the loops had something to do with the way the output was printed. I contend that Charlotte and Diana's remixing and reusing of code did not prevent them from engaging meaningfully and conceptually with important ideas—it allowed for them to work efficiently, and, I would argue, it is not clear that anything was lost in terms of their combinator-

Interviewer:	I think you were right. Like, you did list the pants first, like switching the i and the j did list the pants first and the shirts second. What do you think you would do to your code to get it to be listed in the way that Diana was describing, where you have jeans and then khakis?
Diana:	I think that you could put pants after, like for the i, pants represents i, and then shirts represents j, and we wanna make sure that i comes first and j comes first.

ial reasoning by remixing code rather than writing their own new code.

I then asked them to try another approach to generate code that would organize outcomes how Diana had described.

They then again remixed (copied, pasted, and adjusted) to produce the new code below their initial code (Figure 4). They ran it and generated outputs that were indeed organized how Diana had initially described, with a group of jeans and a group of khakis in the first column. I note that Figure 4 shows that the computer can display both two different bits of code (that represent two different processes), along with the two different outputs of the respective code, which makes

Interviewer:	… What's the differences in those programs, and why is that affecting the way in which they're listed.
Diana:	It looks to me like the loops that you set up, and which category goes with which variable, like either i or j, that kind of determines like the sequential order, but then like the order of i and j in the parentheses after print, determines like which column they go in.
Charlotte:	Yeah.

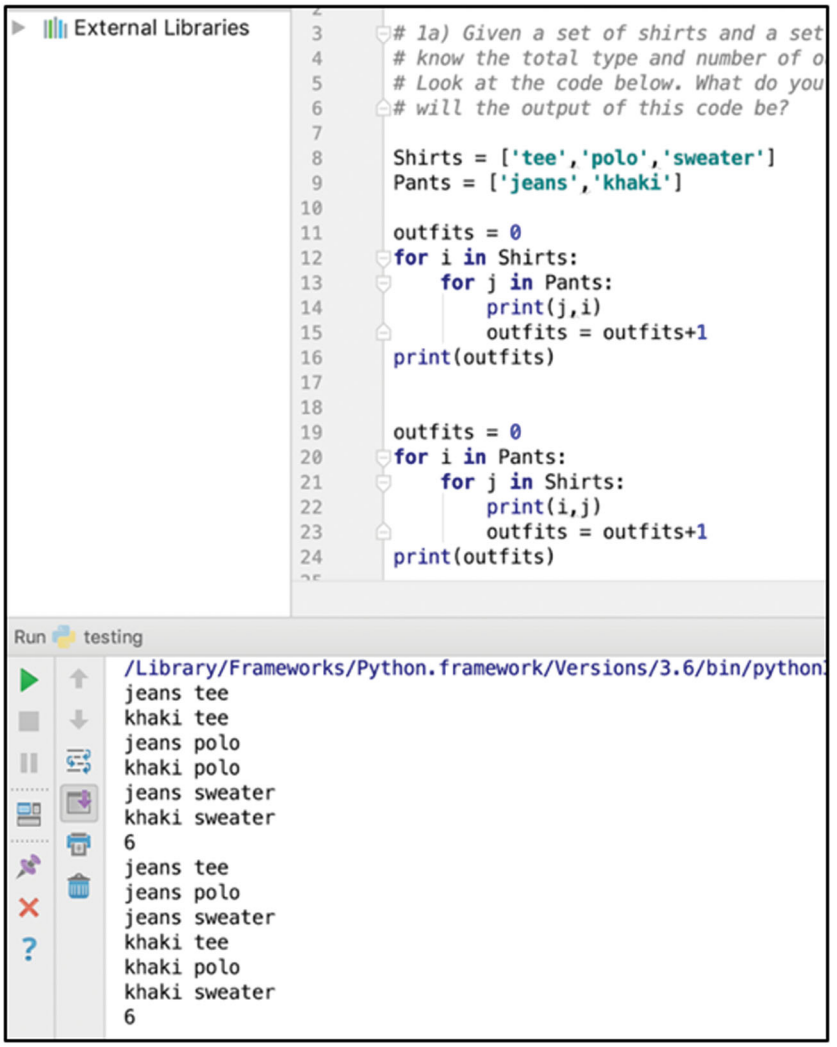

Figure 4. The students try another approach for listing outcomes.

apparent what the differences are both in the code and in the output. I asked them to address differences in the following exchange.

I infer that for Diana and Charlotte at this point, they were making sense of the output in relation to the code, but their discussion and reasoning was primarily about making sense of the code and the syntax. Again, because this was their first exposure to Python and to any programming language, this makes sense. The point is that for Charlotte and Diana, the prediction and reflection illuminated features of the code and the syntax. As they progressed through the teaching experiment, they would gain more experience with the code and would come better to understand the syntax and also make further connections with mathematical and combinatorial ideas. Furthermore, here the computer and the representations afforded by it gave the students the chance to see two different processes and two different outputs at once. Indeed, here, multiple aspects of their coding experience (representational heterogeneity, remixing and reusing code, and immediate feedback that fostered prediction and reflection) all served to reinforce the

combinatorial idea that different counting processes generate and organize sets of outcomes in different ways.

CJ and Corey had a very similar experience on this task. They also engaged in remixing and modified the code from Part 1a, and they predicted a different set of outcomes than was produced. This episode, too, generated rich discussion about counting processes and sets of outcomes, and they engaged in prediction and reflection. As the code in Figure 5a shows, the students switched the variables in the loops without changing the order of the loops (j being associated with Shirts, and i being associated with Pants), and they still printed (i,j). They predicted the output would be grouped according to jeans (e.g., jeans tee, jeans polo, jeans sweater) and

Interviewer:	Okay, so is that what you were expecting?
CJ:	Um, not necessarily because still the tee is still –
Corey:	Yeah, I was expecting it to be like jeans tee, jeans polo, jeans sweater.
CJ:	Well, it's still calling j shirts and then the i is inside the – the pants is listed inside the shirts. I don't know if that –
Corey:	Oh, okay. Maybe –um
CJ:	It's listing i before j.
Interviewer:	And you still did do – it asked you to print jeans first and then shirts and so you did do that. But, sort of, what are you trying to make it do?
Corey:	Put all the groups on the left side so it would be like jeans, jeans, jeans, khaki, khaki, khaki.

then khakis, but their code produced different output than they expected (Figure 5b); they discussed this in the excerpt below.

I point out that, again, the students did what was asked of them (printing outfits with pants first). But, it seemed that they wanted the outcomes to be structured according to groups of pants, and they initiated changing their code to reflect their desired organization. The exchange

(a)

```
# Given a set of shirts and a set of pants we would like to
# know the total type and number of outfit combinations possible.
# Look at the code below. What do you think this code does? What
# will the output of this code be?

Shirts = ['tee','polo','sweater']
Pants = ['jeans','khaki']

outfits = 0
for j in Shirts:
    for i in Pants:
        print(i,j)
        outfits = outfits+1
print(outfits)
```

(b)

```
jeans tee
khaki tee
jeans polo
khaki polo
jeans sweater
khaki sweater
6
```

Figure 5. (a,b) The students edited code and the unexpected output.

(a)

```
# Given a set of shirts and a set of pants we would like to
# know the total type and number of outfit combinations possible.
# Look at the code below. What do you think this code does? What
# will the output of this code be?

Shirts = ['tee','polo','sweater']
Pants = ['jeans','khaki']

outfits = 0
for i in Pants:
    for j in Shirts:
        print(i,j)
        outfits = outfits+1
print(outfits)
```

(b)

```
jeans tee
jeans polo
jeans sweater
khaki tee
khaki polo
khaki sweater
6
```

Figure 6. (a,b) Code that prints outcomes with pants first, grouped by jeans and then khakis.

Corey:	There we go.
Interviewer:	Okay, so what did you do and why did you think that would fix it?
Corey:	I changed the name. So, whereas before it was just shirts and pants. and I changed the variables. But then I changed the variables back, and then I just changed the names because the names are what really matters, now that I look at this. Because it looks like changing the variables didn't really do much. It just kind of switched the words around rather than printing it differently.
Interviewer:	Okay. Yeah. And switching the variables did something but maybe didn't structure it how you wanted it to be.
Corey:	Yeah. Didn't do it the right way.
Interviewer:	Yeah. What are you thinking, CJ?
CJ:	Yeah, so we just made it so it was looking at the pants first and then – so it was filing through the pants and so it looks at jeans and then for inside the pants and then it goes for different types of shirts and then it'll move on to the next thing.

below shows their process of remixing and adjusting the code from Figure 5a to the code in Figure 6a to reflect their desired organization, which yielded the outcomes in Figure 6b.

I offer a couple of comments about this problem, as both pairs of students had similar experiences. First, in terms of Lockwood's model, this task illuminated for the students the idea that two different counting processes may create two different lists of outcomes. It seemed from the students' initial predictions that they wanted to organize pant-shirt pairs according to groups of pants and then groups of shirts, which is natural and appeals to the odometer strategy of holding an item constant and systematically varying others (English, 1991). In fact, this kind of organization seemed to be how they initially thought about (and perhaps even wanted) those outfits to be listed—Corey even called such a listing "the right way." Given their predisposition to list in a particular way, I contend that it was enlightening for students to see that there was a different way in which the outcomes could be printed (and, further, the code they had written actually produced this alternative organization). As Lockwood (2013; 2014) has previously claimed, even understanding that lists could be generated in different ways is important for students' combinatorial thinking and activity. This simple Outfits problem seemed to bring this phenomenon to light for these students. I also argue that computing played a role in highlighting this phenomenon. Because the students could see what code was produced when they pressed the "run" button, they received feedback on what output the code actually generated. I suggest that having something other than themselves (in this case, the computer) create the lists lent credence to the idea that a counting process generates certain outcomes in a particular way. I also contend that the practice of prediction played an important role in this episode, and we see how prediction highlighted a disparity between their thinking and what the code actually generated. This, in turn, enabled the students to reflect on the outcomes and attempt to make sense of the situation, and the students established important ideas both in terms of the coding and in terms of combinatorics. As with Charlotte and Diana, aspects of their coding experience (representational heterogeneity, remixing and reusing code, and immediate feedback that fostered prediction and reflection) reinforced that counting processes generate and organize sets of outcomes.

Outfits problem part 2b—predicting and reflecting on mathematical expressions

The previous two examples of Parts 1a and 1d of the Outfits problem particularly highlighted certain elements of Lockwood's (2013) model, namely connecting counting processes and sets of outcomes. In this section, I provide an instance in which the students reasoned about the third component of the model, formulas/expressions, in the context of the Outfits problem. Due to space, I focus only on CJ and Corey's work on Part 2b, although Charlotte and Diana's work on this problem was similar (and we see evidence of their reasoning about mathematical expressions on the License Plate problem). Part 2b of the Outfits problem states: *Create some new code that would use 3 loops and have 24 different outfits total. Explain what you did. What is a mathematical expression that represents the number of outputs of your code?* I discuss this problem because

it demonstrates an instance in which students' attention was explicitly on a mathematical expression, and it further exemplifies how these tasks and prompts within a computational setting could leverage aspects of Lockwood's (2013) model of students' combinatorial thinking. CJ began by articulating their overall goal for the problem.

> CJ: Yeah, we need to change how many shirts, pants, and hats. So when we find the total combinations we can multiply the amount of shirts times the amount of pants times the amount of hats to hit 24.

They decided that multiplying 2, 2, and 6 would give 24. They produced the code in Figure 7,

Interviewer:	What do you think the total will be and how do you think it will be structured?
Corey:	24 should be the total.
CJ:	I think it will go through i is listed first, i is shirts so it'll go through the first shirt tee, jeans, baseball and then we have tee, jean, winter. And then we have tee, khaki, baseball. Tee, khaki, winter. And then it'll move on to polo.

and before they ran it, I again asked them to predict both the total number of outcomes and the structure of the output.

I also asked them what they expected a mathematical expression that represented the total would be, and Corey guessed 2·2·6, and CJ said, "Or 6·2·2." They predicted a total of 24, and they ran the code, which produced the output in Figure 8a. Again, I asked them to reflect on their work, saying, "Did it give you what you expected?" CJ explained, "It's listing the shirts first because that's what i is and we look at i first. That's our first for loop."

The students proceeded to discuss the mathematical expression and whether they were thinking of it as 2·2·6 or 6·2·2. We highlight one brief exchange that demonstrates CJ making a connection between the expression of 6·2·2 as related to the set of outcomes, and he particularly

CJ:	The way I – I can't wrap my brain around the 6 times 2 times 2 exactly, but for every shirt there's four different combinations and then so for how many shirts there are there's going to be that many combinations. So I can see the 6 times 4 because there are four things for every shirt.
Interviewer:	Okay. And are you seeing those four – can you actually use the mouse maybe to point to where are the four?
CJ:	So there's four combinations of jeans, baseball; jean, winter; khaki, baseball; khaki, winter [highlights the outcomes in Figure 8b]. And that goes through for every single shirt as well.
Interviewer:	Okay, cool. Is there any way to think of that 4 as 2 times 2? You're maybe not thinking of it currently as 2 times 2.
CJ:	Yeah, I wasn't exactly. But I guess there are two different hats for every pair of pants. And then there's two different types of pants. For the other pair of pants you need – it's the same–there's two combinations for it so you times it by 2.

```
# Create some new code that would use 3 loops and have 24 different outfits total.
# Explain what you did. What is a mathematical expression that represents the
# number of outputs of your code?

Shirts = ['tee','polo','sweater','cutoff','button up','flannel']
Pants = ['jeans','khaki']
Hats = ['baseball','winter']

outfits = 0
for i in Shirts:
    for j in Pants:
        for k in Hats:
            print(i, j, k)
            outfits = outfits + 1
print(outfits)
```

Figure 7. Corey and CJ's code for Part 2b of the Outfits problem.

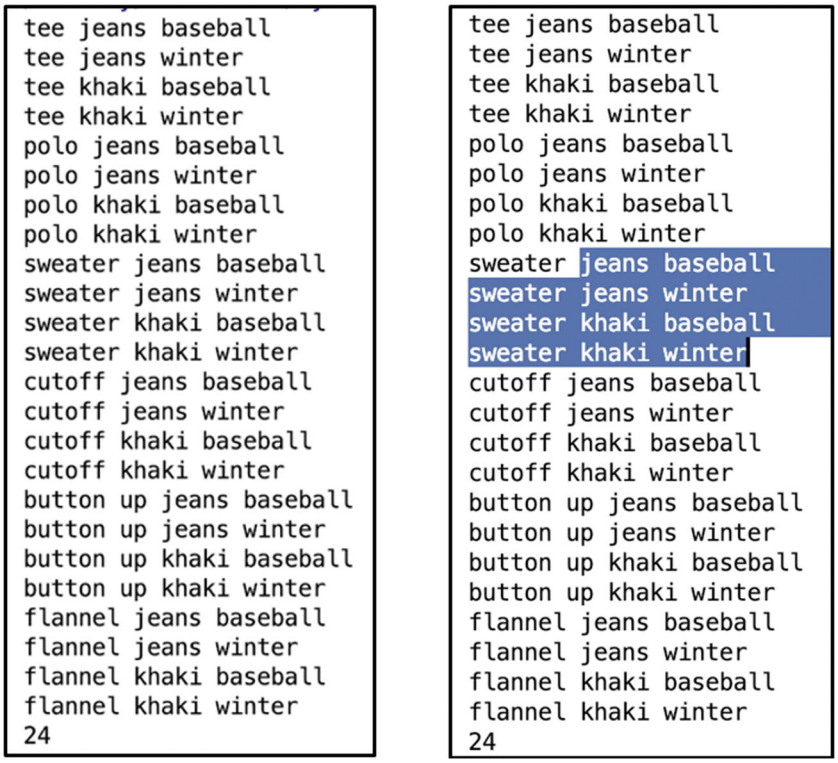

Figure 8. (a,b) The students' output for Part 2b of the Outfits problem, and CJ's four highlighted options within a sweater.

highlighted some of the outcomes within the list of outputs from the computer to demonstrate the four combinations within each six options for shirt (Figure 8b).

This brief episode highlights a couple of different points about the formulas/expressions components of Lockwood's (2013) model and how students might reason about such expressions via prediction and reflection in a computational setting. One observation is that students can be asked to predict either a numerical value (such as: *What do you expect the total to be?*) or to predict an actual mathematical expression (such as: *What is a mathematical expression that would express the total?*). CJ and Corey could use a counter in their code to keep track of individual outcomes, which had been modeled for them in the very first part of the Outfits problem. Because of this, the students could determine the total number of outcomes they would get, which offered some immediate numerical verification for students. This ability to verify solutions is something that is often difficult to accomplish in combinatorics (e.g., Eizenberg & Zaslavsky, 2004), and so this was a particularly valuable feature of the students' engagement with computing.

However, in designing these tasks and conducting these interviews, I ultimately wanted students to be able to articulate a mathematical expression and not just a numerical total, because a mathematical expression can often provide some justification and conceptual foundation for why the numerical total is what it is. Thus, I was not satisfied simply to have the students see that, say, their guess of 24 aligned with the 24 in the output (even if such a realization was sometimes useful). Said another way, I hypothesized that this aspect of their computational experience (predicting and reflecting) would support certain desirable combinatorial reasoning (namely, justifying a mathematical expression). I thus regularly attempted to draw students' attention to mathematical expressions as they engaged in both prediction and reflection.

Overall, the students' work on this Outfits problem demonstrates how prediction and reflection afforded opportunities for them to connect counting processes and sets of outcomes, ultimately

giving them insight into both counting processes and the programs they wrote. The students were introduced to code and to the idea that code would generate a precise list of outputs that was organized in some way. These tasks helped students begin to formulate (even implicitly initially, and then explicitly) connections between counting processes, sets of outcomes, and mathematical expressions, often through elicited activities of prediction and reflection. In some cases, these activities also helped students to gain insight into what the code was actually doing in the loops and print statements. So, even in these initial tasks, some of the features of computing, including certain computational representations and immediate feedback, served to facilitate and reinforce activities of prediction and reflection. The Outfits problem was not particularly difficult and did not have large sets of outcomes. But, perhaps because of the small sets of outcomes and the relatively simple code structure, students were able to engage in prediction and reflection and to use that activity to connect components of Lockwood's model, being introduced to computational representations in the process.

Relating components of the model in subsequent episodes in the teaching experiment

I now offer three additional examples of ways in which representations within the computational setting and prompts for prediction and reflection facilitated students' productive combinatorial thinking. I aim to demonstrate that the phenomena observed in the Outfits problem persisted in subsequent tasks, and I particularly want to show the students' work on problems with larger sets of outcomes, which allow for some additional insights and points of discussion.

Experimentation and alternative solution methods in the license plate problem

The following example illustrates Diana and Charlotte's understanding of the relationship between counting processes and sets of outcomes particularly within the computational setting. This is especially important in highlighting how the computing experience, and the prediction and reflection the computer afforded, together served to strengthen combinatorial relationships within Lockwood's model. Here, during their second interview session, Diana and Charlotte were considering Part 4a of the License Plate problem: *A license plate consists of six characters. How many license plates consist of three numbers (from the digits 0 through 9), followed by 3 lower case letters (from the first 5 letters in the alphabet), where repetition of characters is allowed? Write some code to solve this problem. What is a mathematical expression that represents the number of outputs of your code?* They had two different ideas for code that could solve the problem, and by using the computer, they could easily test both approaches. Diana initially suggested that they code six nested loops, where the first three loops iterate through a set of Numbers, and the last three loops iterate through a set of Letters. She suggested this correct approach in the excerpt below, and the students eventually coded this (Figure 11). However, Charlotte suggested an alternative approach (underlined below), and they considered a process that was reflected by creating two nested for loops and then simply printing (i,i,i,j,j,j). This is not an unreasonable answer, as it

Diana:	Maybe we should figure out a way to have it recognized that there's three letters – or three numbers in a row, followed by two letters in a row. So, we could do like a sorting system and like repeat the sets. So, like for i and numbers for j and numbers for k and numbers, and then keep going with the letters for l and letters for m and letters for n and letters.
Charlotte:	Do you think it would be easier if we kept like these – I say just change them. But then when it comes down to print, do you like, i comma, i comma, i comma, j,j,j?
Diana:	Yeah, we could try that.
Charlotte:	I don't really know if that would work or not [they proceeded to write the code in Figure 9].

```
# A license plate consists of six characters. How many license plates consist
# of three numbers (from the digits 0 through 9), followed by 3 lower case
# letters (from the first 5 letters in the alphabet), where repetition of characters
# is allowed? What is a mathematical expression that represents the number
# of outputs of your code?
Numbers = ['0','1','2','3','4','5','6','7','8','9']
Letters = ['a','b','c','d','e']

Licenseplate = 0
for i in Numbers:
    for j in Letters:
        print(i,i,i,j,j,j)
        Licenseplate = Licenseplate+1
print(Licenseplate)
```

```
5 5 5 e e e
6 6 6 a a a
6 6 6 b b b
6 6 6 c c c
6 6 6 d d d
6 6 6 e e e
7 7 7 a a a
7 7 7 b b b
7 7 7 c c c
7 7 7 d d d
7 7 7 e e e
8 8 8 a a a
8 8 8 b b b
8 8 8 c c c
8 8 8 d d d
8 8 8 e e e
9 9 9 a a a
9 9 9 b b b
9 9 9 c c c
9 9 9 d d d
9 9 9 e e e
50
```

Figure 9. (a,b) Diana and Charlotte's initial code and output for the License Plate problem.

will still print outcomes with three numbers followed by three letters, but it does not actually produce all of the desirable outcomes. They coded this approach, which is seen in Figure 9.

Interviewer:	And then, yeah. What do you think will come out of that? Or what are you anticipating the outcomes will look like?
Diana:	I think it will definitely give us strings of three numbers, followed by three letters, but I'm a little worried it might do only the identical numbers, followed by identical letters.
Charlotte:	Right.
Diana:	But I don't know. I don't know how it will read it for sure.
	[. . .]
Charlotte:	Right. There we go. Okay. And – oops.
Diana:	Okay. So, it did give us only the repeats.

I asked them to predict the outcomes. After they made a prediction, they ran the code and yielded output displayed in Figure 9b.

In the underlined portion of the excerpt above, Diana stated what she expected the outcomes to be. She reiterated that they wanted outcomes with three numbers followed by three letters. However, she correctly predicted that perhaps the code would only print identical numbers and letters. Then, when they actually ran the code, they saw that they had indeed gotten only a subset (50) of the set of outcomes that they actually wanted (Figure 9b). In terms of Lockwood's (2013) model, this underscores the relationship between a counting process and a set of outcomes by showing them what their particular process yielded. I infer that Diana's prediction, and her caution about why there would only be identical elements, suggests that she was thinking carefully about what outcomes the counting process represented by her code might generate.

I reminded Diana of her initial idea of coding 6 nested loops, and she then edited the existing code to produce the code in Figure 10. I again asked for them to predict the structure of the outcomes, and, in response, Diana predicted an expression for the total number of outcomes. This exemplifies that, through prediction about these computational representations, the

Interviewer:	So, again, kind of what do you predict will come out of that you think? In terms of … the structure of the outcomes?
Charlotte:	I mean, hopefully it's gonna give us the – what we actually want. But I'm trying to think about structure.
Diana:	Yeah. It'll probably do – it'll keep these – because these are the starting points. So, like the triple 0 and then it'll probably go like 001, 002, 003, and include all those different combinations. And in terms of like a mathematical expression, I was thinking, maybe it might multiply the options by each other. So, like with our paper here – Since this is 0–9, then there are 10 options. And there's 10 options here, and there's 10 options here. It might do 10 times 10 times 10 times 5 times 5 times 5. Because there's five options here.

```
# A license plate consists of six characters. How many license plates consist
# of three numbers (from the digits 0 through 9), followed by 3 lower case
# letters (from the first 5 letters in the alphabet), where repetition of characters
# is allowed? What is a mathematical expression that represents the number
# of outputs of your code?
Numbers = ['0','1','2','3','4','5','6','7','8','9']
Letters = ['a','b','c','d','e']

Licenseplate = 0
for i in Numbers:
    for j in Numbers:
        for k in Numbers:
            for l in Letters:
                for m in Letters:
                    for n in Letters:
                        print(i,j,k,l,m,n)
                        Licenseplate = Licenseplate+1
print(Licenseplate)
```

Figure 10. Diana's code after their initial exploration of the License Plate problem.

students related sets of outcomes and mathematical expressions with the process reflected by their code.

They wrote down the product 10·10·10·5·5·5, which I told them was 125,000. They ran the code,[5] yielding a numerical output of 125,000, and they confirmed that it had done what they expected.

In summary, this episode suggests that the students were thinking about the nature of the outcomes and what they would look like on the screen. They saw that they wanted to print a 6-tuple, and they came to realize that using the command print(i,i,i,j,j,j) only gave repeated elements in their output. The computational setting allowed them to explore this idea and gave feedback to confirm that their output would only be repeats. Then, the computer facilitated their testing another idea of 6 nested for loops, and they surveyed the results of those outcomes to see that they were in fact not just printing repeating elements as they had in their initial code. In this way, the computational setting was closely tied to their prediction and reflection. This immediate feedback, facilitated by the computer, allowed them to get a better sense of what the outcomes were in that situation, and, in particular, what the outcomes would be of the specific process they had coded. Here they could reflect on the computational representations of the code and the outcomes.

Identifying structure in lists of outcomes in the ROCKET problem

I now present an episode from CJ and Corey's fourth session in which they were working on arrangement without repetition problems (also called permutation problems). They had initially been given code that answered the question: *How many ways are there to rearrange 5 people: John, Craig, Brian, Angel, and Dan?*, and they also then adjusted that code to solve arrangements of the word PHONE (see Table 3 for exact wording of these problems). I do not focus on this initial work (some of which is described in Lockwood & De Chenne, 2020), but I focus on a problem involving arrangements of the word ROCKET. This occurred later in the teaching experiment (Session 4) and involved a different kind of problem than the Outfits problem (arrangements without repetition). Here, they used the set of outcomes to justify an expression, which is something I have not yet demonstrated in the data.

[5]I briefly note what occurred when they ran the code. It is not relevant to their reasoning about the problem, but it this highlights a potential computational limitation. When the students went to scroll up on this problem, the computer did not print outcomes prior to 300eed. Essentially, the computer was not able to print all 125,000 outcomes in PyCharm. We thus reduced the problem to only involving the numbers 0 through 5, and the students proceeded to work on that problem for the rest of the session. I bring this up here to note that there was some limitation to what the computer could produce, and this was something we dealt with during the interview.

I asked the students the following questions: *Now can you adjust the code from PHONE to list the numbers of arrangements of the letters in the word ROCKET? … How many arrangements do you expect to get, and why do you think that? How will the list of outcomes be structured?* The students began with their code from the PHONE problem (Figure 11a). They remixed their code, as they adjusted it by changing the initial list (called People, a remnant from the initial arrangement problem) to include the letters R, O, C, K, E, and T and adding an additional for loop and conditional statement (Figure 11b). I note here that I interpret the remixing as an indicator that they realized what features of the code were more or less important to preserve. That is, the fact that they remixed and reused code does not suggest a weaker conceptual understanding, but rather it highlights that they could attend to the salient features of the program (such as the contents of the initial list, the structure of the for loops, and the details of the conditional statements) and not overly focus on superficial features (such as the name of the initial list).

Before they ran the code in Figure 14b, I again asked for prediction, both in terms of a total number and in terms of some structure of the set of outcomes. Corey said of the total, "I believe there will be 6·5·4·3·2·1" which he confirmed was 720. Then the students started to write out the first few outcomes (Figure 12).

I then asked the students to predict what they might expect to see in the whole set of outcomes, particularly asking about the behavior of the first column. Corey identified a correct overall structure, noting that the first column would be split into sixths, each starting with a different first letter of ROCKET, and he articulated structure within each of those six chunks of outcomes. I also asked them to think about the behavior at the rightmost column, which the students also correctly noted would cycle through the remaining letters individually. After this conversation, the students ran the code, yielding 720 as they expected, and Corey said, "A sixth of all 720 will be R and then another sixth will be O, another sixth will be C, another sixth will be K, and then E and T. And then just like the last problem—for each of those letters—a fifth of those will be one of these letters."

Later in the work on these tasks, the students were asked to justify to someone why there would be 8·7·6·5·4·3·2·1 arrangements of the letters in the word PORTLAND. They struggled a bit with explaining this, and I asked if creating code to see the whole set of outcomes might explain why the options are multiplied together. CJ began to scroll through the output of the

CJ:	And just visually seeing how when R is the first one, you can see the second ones shifting from O to C to K to E to T and then in the third, that's just when R is the first one. And then you can see the third one going C to K to E to T to O to C. And then you can just see it moving back and forth and so you can see a pattern there.

(a)

```
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)
```

(b)

```
arrangements = 0
People = ['R','O','C','K','E','T']

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:
                                    for p6 in People:
                                        if p6!=p5 and p6!=p4 and p6!=p3 and p6!=p2 and p6!=p1:
                                            arrangements = arrangements + 1
                                            print(p1, p2, p3, p4, p5, p6)
print(arrangements)
```

Figure 11. (a,b) The students' code for PHONE, which they adjusted to get code for ROCKET.

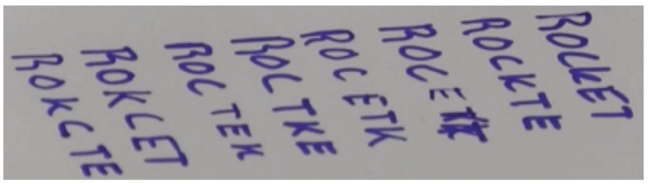

Figure 12. Corey and CJ's list of the first eight outcomes.

arrangements of ROCKET. He scrolled through the set of outcomes that had been generated by the computer, making the following comment:

As he scrolled, he explicitly drew attention to the structure of the list of outcomes and articulated a pattern he saw. I infer that CJ was attending to a structure within the whole set of outcomes, and the fact that he could scroll through this set of outcomes (which is an activity afforded by the computer) helped him to explain and justify why multiplication made sense. In fact, he went on to use a similar explanation to explain his solution to the PORTLAND problem. The point here is that the students were able to use the computational representation of the set of outcomes as part of their justification. In this way, the computational setting offered an additional representation that the students could use as a source of justification for why the mathematical expression worked. I consider this to be an instance of the students reflecting on the outcomes and connecting them to the process of multiplication (and the expression of 8·7·6·5·4·3·2·1). This episode shows that in some more advanced problems, the students were able to leverage computational thinking practices in order to make connections among components of Lockwood's (2013) model. Similar instances occurred throughout the teaching experiment.

Leveraging multiple representations in the coin flips problem

I now offer one final example that highlights Charlotte and Diana's uses of multiple representations and that demonstrates how the computational representations offered some combinatorial insights for the students into ways in which counting processes generate outcomes. Charlotte and Diana were working on the Coin Flips problem, which stated the following: *Write a program to list (and determine the total number of) all possible outcomes of flipping a coin 7 times.* The students remixed code from the License Plate problem discussed previously to solve the Coin Flips problem, as seen in Figure 13.

I asked the students why they made seven nested loops, and Diana said, "Because there is seven different digits, or characters in the outcome. So, we're going to sort it into seven columns and each one of those is independent and not – like they're variable, so they don't depend on each other." She explained when she said "sort" she meant, "The first flip of the coin is in the first column, the second flip of the coin is in the second column, and so on." This exchange and discussion of columns suggests that the students were thinking about what the output of the program would be, and that they desired to have output that consisted of 7-tuples.

```
# Write a program to list (and determine the total number of)
# outcomes of flipping a coin 7 times.
Numbers = ['heads','tails']

Licenseplate = 0
for l in Numbers:
    for m in Numbers:
        for n in Numbers:
            for o in Numbers:
                for p in Numbers:
                    for q in Numbers:
                        for r in Numbers:
                            print(l,m,n,o,p,q,r)
                            Licenseplate = Licenseplate+1
print(Licenseplate)
```

Figure 13. The students' code for the Coin Flip problem.

Interviewer 1:	Great. And sorry, again, what do you think the outcomes are gonna look like then? How will they be structured? And how many do you think you're gonna get?
Charlotte:	So, since there are seven and you can get two possible outcomes for each, you just do two times two, times two, times two, times two, times two, times two.
Interviewer 1:	Cool.
Charlotte:	For a total of seven.
Diana:	Yeah. So, like two to the seventh.

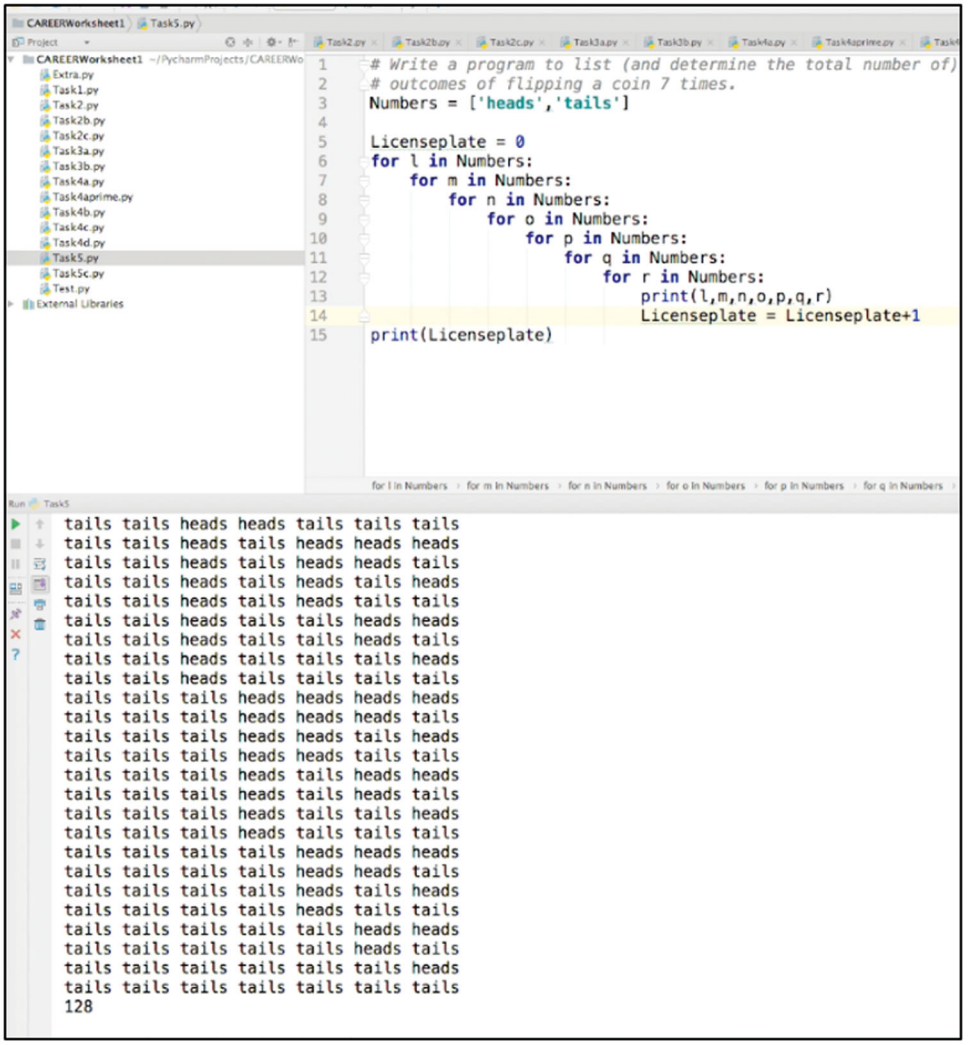

Figure 14. Output from the students' code.

Before they ran the program, I asked them to predict how the outcomes would be structured and how many there would be. In the excerpt below, I note that they correctly predicted that there would be 2^7 total outcomes.

Importantly, the students were correct and seemed about to reason about the answer of 2^7. However, the students getting the correct answer was not the only (or even the primary) aim of this task. I also wanted the students to reason about the outcomes and to understand a relationship between what the counting process was doing and what code would emerge. Had I only cared about the answer, we might have been satisfied with their answer of 2^7; however, since I was

motivated to see how they would reason about the structure of the outcomes, I asked the students to predict how the output would be structured. They leveraged the aspect of their computational experience that afforded prediction and reflection, knowing that they would be able to run the code and check their prediction. In the exchange, Charlotte's intonation suggested uncertainty ini-

Interviewer 1:	Yeah. And then how do you think they're gonna be structured?
Charlotte:	It'll be like – They all seem – (pauses) I don't know.
Diana:	I think it'll probably do like seven heads first, and then six heads and one tails, and five heads and two tails, maybe?
Charlotte:	And go all the way down?
Diana:	And go all the way down.
Charlotte:	And then go to seven tails, and all the way down?

tially, but then Diana suggested a potential structure of seven heads, then six heads and one tail, and five heads and two tails. This was an incorrect prediction, as the first three outcomes are not HHHHHHH, HHHHHHT, HHHHHTT, but rather HHHHHHH, HHHHHHT, HHHHHTH.

In their exchange below, the students were surprised by the actual output that was generated by the code. They did get an answer of 128 total, but they indicated that this was not what they had expected. Indeed, Charlotte said, "that's weird," and Diana said, "I'm not exactly sure how

Charlotte:	It switches back and forth between –
Interviewer 1:	Yeah. What do you mean?
Charlotte:	So, the last column goes heads, tails, heads, tails, heads, tails. So, it does –
Diana:	Oh, yeah. So, it's – yeah. It did have the first one as like seven heads, but then it didn't go like six heads, one tails, five heads, two tails. It went still six heads, but the tails was not on the end. It was in the –
Charlotte:	Yeah. It goes through each option of – So, it goes from seven, and then it goes like six heads and it shows you multiple different options. That's weird. Tails, tails, heads.
Diana:	Yeah. I'm not exactly sure how it's sorting it.

it's sorting it." Note that this offers a contrast with Corey and CJ's work on the ROCKET problem described previously, where they almost immediately realized the underlying structure of the outcomes.

I interpret that the students experienced a perturbation related to their combinatorial reasoning, in the sense that they encountered an unexpected result that differed from what they had predicted. Here, the whole list of outcomes that the computer generated caused the students to notice an organization of the outcomes that did not match what they had expected to see.

In particular, Charlotte then paused their activity to think about what was going on. She began to scroll through the output, as she noticed some regularity in the list of outcomes and wanted to make sense of what was going on in terms of patterns and structure. The following exchange is important because it demonstrates combinatorial insight that the students were gaining in terms of the organization and structure of the set of outcomes, which I argue was supported by their experience computing. Charlotte first saw some regularity in heads; she observed that while the first column consisted of heads then tails, the second column split each part of the first column

Charlotte:	I'm needing to scroll real quick. So, I think it's varying in each column first instead of each row. So, for the first column it's going through all the heads, and then it goes to all the tails. And then the second column it's doing like – Then it goes heads, tails within heads category, and then for the tails.
Interviewer:	So, show me what you mean in that second column.
Charlotte:	Okay. So, in the first column it just goes all heads and then all tails. And then in the second column it goes heads, and then – Because it splits kind of like the first column into two –
Interviewer:	Great.
Charlotte:	– between heads and then it eventually it switches over to tails. And then I think it does it for the third column; it splits it in three, heads, tails, heads. And then the fourth column, heads, tails, head, tails. It keeps going.
Interviewer:	Okay. So, check – so, okay –
Diana:	I think for the third column it would probably split into four because you have – your lack of heads in the second column and then it needs to split into heads and tails for the third column. And a new block of tails in the second column.
Charlotte:	Yeah. So, it's kind of like our tree diagram.
Diana:	Yeah.

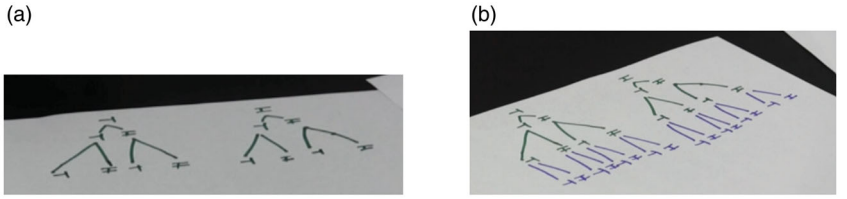

Figure 15. (a,b) The students' initial drawings of a tree diagram.

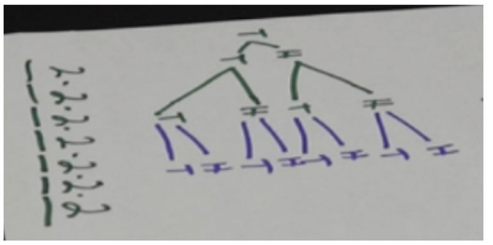

Figure 16. The students' representations of a tree diagram and an expression.

into two (so, heads were then split into heads and tails for the second column). She made a comment that perhaps the third column would be split into three, but then Diana intervened, suggesting that the heads in the third column would actually get split into four, not three.

Here Charlotte and Diana brought up a different representation, a tree diagram, and the interviewers prompted the students to pause and draw the tree diagram. After they had drawn two branchings (Figure 15a), Diana connected it to the list by saying, "So, this is what we were talking about with the third column as like it's splitting into four." They drew another set of branches (Figure 15b) and then Charlotte said, "It just keeps splitting. Like that? Just keep on splitting until we get to seven. So, we have one, two, three, four, so far. And then it'll do three more."

The students did not complete the tree, but we discussed their partial tree with them, and they connected even their partial tree to the mathematical expression of 2^7 they had initially found. Recall that Charlotte had been particularly curious about the fact that the last column switched back and forth between heads and tails and had not been sure why that was happening. After having seen the structure of the outcomes and connected them to the tree diagram, Charlotte said that "it makes way more sense" why the right most column in the set of outcomes was switching between heads and tails.

As we continued to discuss the code, the tree diagram, and the outcomes, Diana also wrote out the expression 2·2·2·2·2·2·2 (Figure 16), and she explained that, "There are two options here, and then that – like for each of those two options there's two more options [gestures/points to the tree diagram]. And then for each of those options there are two more options. So, it just multiplies it each time." Ultimately, I see this as an interesting instance of representational heterogeneity in mathematics, where students were drawing on many different representations as they engaged in computational thinking practices in combinatorial problem solving.

I conclude this example with a quote in which we see the specific value of the computational representation of the entire set of outcomes for Charlotte. The interviewer asked Charlotte, "Can you just talk a little bit more about that process and what you guys were looking for and how

Charlotte:	I mean, probably just starting from the beginning and looking at each different row and seeing that it didn't come out how we thought it would come out. Then I kind of just started looking at other options and I kind of noticed that the Ts start getting closer and closer to the top in each column, so then I wanted to see the first and second columns to see how it broke it up between heads and tails.

you made sense of how the outcomes were structured?" Charlotte's response suggests that something about the structure of the outcomes themselves, and the ability to survey of the entire outcomes list, was useful for her.

This feature of computing gave her some insights that she could raise to Diana. Ultimately, they seemed to gain a better understanding of why the expression 2^7 gave the correct answer, and, importantly, why their process would generate those outcomes in exactly that way. I argue that this experience of examining and reflecting on outcomes deepened their understanding of key combinatorial concepts that I wanted them to understand—namely, that there is a relationship between counting processes and sets of outcomes.

This episode highlights an example where even though the students initially got the correct numerical answer, we have evidence that they did not actually understand that process (and how exactly that multiplicative process would organize and structure the set of outcomes). This is because the students did not initially correctly predict how the outcomes would be structured, and they were unsure and perplexed by the actual outcomes that they saw. Notably, Charlotte was engaging in activity that was different from listing outcomes herself—instead she was reflecting on an entire set of outcomes. And in reflecting on that output, she ultimately was able to tease out precisely what happened. Thus, I argue that the experience of actually seeing the representation of whole list of outcomes did prompt and encourage them to pause and to make sense of what was happening in the problem. In the end, they were able to interpret what was happening within the list of outcomes, and they connected this to another representation of a tree diagram and to a mathematical expression.

Discussion, implications, and avenues for future research

In this paper, I have attempted to demonstrate ways in which students' computational experiences (in the sense of computational thinking practices they experienced while engage in Python programming activities) interacted with and supported their combinatorial reasoning. In Table 5, I summarize these findings; together these address my first research question, which stated: *In what ways can computational thinking practices facilitate certain desirable aspects of students' combinatorial thinking and activity?* For each problem presented in this paper, I list the aspects of computational experience in which the students engaged, and I list the combinatorial insight that was supported and enriched by those particular experiences. This summary highlights the specific ways in which students' experience with computing during the teaching experiment served to enrich certain desirable aspects of their combinatorial reasoning. In the remainder of this section, I offer points of discussion and synthesize my findings, highlighting what we learned specifically about students' combinatorial reasoning within their computational experiences (which addresses my second research question: *What does this tell us about the nature of students' combinatorial reasoning and activity as it relates to computational thinking practices?*). I also describe limitations of the study and conclude with implications and avenues for future research.

The computational setting introduced computational representations, facilitating representational heterogeneity and connections among representations

By working within the computational setting, students gained access to new computational representations of combinatorial ideas. By using the term "computational representation," I do not intend to promote a dichotomy or a divide between computational and non-computational representations, and indeed I acknowledge that students can and do use a variety of representations in their computing (e.g., Farris et al., 2020; Sengupta et al., 2013). However, I maintain that there are some representations that only exist via the computer, namely the code itself (as written in the computer) and the output of that code (in the form of entire lists produced in the PyCharm

Table 5. Connecting computational experience with combinatorial insights the students gleaned.

Problem	Computational experience	Combinatorial insight
Outfits 1a and 1d	• Representational heterogeneity (code, list of outcomes) • Feedback fostering prediction and reflection • Reusing and remixing code	• There is a relationship between processes and outcomes (different processes structure outcomes differently)
Outfits 2b	• Representational heterogeneity (code, list of outcomes, mathematical expression) • Feedback fostering prediction and reflection	• A mathematical expression reflects a process and represents a structuring of the set of outcomes • Mathematical expressions and formulas can be justified by attending to structure in a set of outcomes
License Plates	• Representational heterogeneity (code, list of outcomes, mathematical expression) • Feedback fostering prediction and reflection	• There is a relationship between processes and outcomes (different processes structure outcomes differently)
Rocket	• Representational heterogeneity (code, list of outcomes, mathematical expression) • Reusing and remixing code	• There is a relationship between processes and outcomes (different processes structure outcomes differently)
Coin Flip	• Representational heterogeneity (code, list of outcomes, mathematical expression, tree diagram) • Feedback fostering prediction and reflection • Reusing and remixing code	• There is a relationship between processes and outcomes (different processes structure outcomes differently) • A mathematical expression reflects a process and represents a structuring of the set of outcomes • Mathematical expressions and formulas can be justified by attending to structure in a set of outcomes • It is important to understand the nature of the outcomes one is trying to count (that is, what constitutes an outcome)

interface). I consider now these computational representations within the broader perspective of computational thinking practices, in light of prior work that values representational heterogeneity. I acknowledge that students draw on a variety of representations—including drawings, dialogue, inscriptions, and more (e.g., Danish, 2014; Farris et al., 2020; Sengupta et al., 2018)—even as they engage in computational thinking practices. My goal in this section is to focus on computing particularly within the context of combinatorics—to acknowledge the existence of such computational representations broadly but then to focus specifically on what such representations afford in terms of students' combinatorial reasoning. In this way, I speak to particular ways in which combinatorics and computing reflexively interact, thus addressing the research questions I have posed. In light of the framing of disciplinary reflexivity, I now draw some conclusions about computing particularly in relation to combinatorial thinking and activity.

Counting processes and computer code as a representation

I begin by discussing counting processes and the role that computing played in introducing and reinforcing these processes for the students in my study. I interpret that the Python code served as a representation of counting processes that students could refer to and reason about. Such code has several elements, including, for instance, the initially defined lists, the loops themselves, conditional statements within the loops, variables within loops, and print statements. The code offered students a way to encapsulate a counting process and a way to think about certain

elements of such processes. More specifically, in writing or evaluating code (including reusing or remixing code), a student has to think precisely through an exact process that they want to have happen, they have to reason systematically about the step-by-step algorithm of a certain counting process, and they have to communicate that to the computer. Such communication with the computer implies a necessary level of precision in describing a process. This came out in Charlotte and Diana's work on the License Plate and Coin Flips problems and in Part 1d of the Outfits problem for both pairs of students. In these cases, the students were not initially precise in their process, but they refined their thinking through refinement of their code to become more exact in articulating the details of their processes. In particular, in Part 1d of the Outfits problem, they initially thought that switching variables in their code would produce outfits that were structured by groups of pants followed by shirts. This is not what was produced though, and they had to reexamine and refine their process by editing the code. Combinatorially, in articulating a counting process, one could describe it verbally, or one could write down a paragraph or a series of steps.

This feature of counting and computing emphasizes why these two domains are particularly well-suited. Thinking about code as a process is common in computing education, and in some sense the students' work in this paper is not unexpected in terms of existing research on computing education. However, taken in the context of combinatorics, this common practice takes on special meaning and has rich, specific mathematical implications. In particular, the combinatorics education research has highlighted that counting processes alone can be hard to articulate and describe, and that students do not always know what their processes are doing (Annin & Lai, 2010; Lockwood, 2014) and, specifically, what the output of a process actually is. In light of this research, then, we see how the practice of computing connects well with and offers implications for this existing problem and issue within combinatorics education. This exemplifies disciplinary reflexivity—we have a common computational thinking practice involving reasoning about processes and algorithms, and we have a disciplinary setting where careful reasoning about, reflection on, and understanding of processes is especially vital. Thus, in terms of counting processes and understanding them, the fact that enumeration as a discipline involves the articulation and implementation of precise counting processes, it is particularly well suited to benefit from programming and using code to articulate processes precisely.

Sets of outcomes and output as a representation

The computer also afforded the generation of entire sets of outcomes, and these entire sets served as a representation about which students could reflect and reason. The fact that the students could survey a whole set of outcomes allowed them to reflect on the overall structure of the output, and they could engage in activities like being able to scroll through and identify regularity and patterns. Such activity seemed to help with practices like justification, as seen in work on the ROCKET and Coin Flip problems. In addition, being able to survey whole sets of outcomes allowed them to critique their own reasoning and reflect on predictions that did not accurately reflect the output of the code, which we saw in students' work on the Outfits and License Plates problems. In this study, the computational setting afforded the generation of entire sets of outcomes, even large ones, and this showed students how their process created and organized outcomes.

I acknowledge that the computer is not *necessary* to give students access to entire sets of outcomes. Students perhaps could list outcomes for some small problems, and, in theory, they could generate large sets of outcomes by hand given enough time and resources. Additionally, teachers or researchers could give entire large sets of outcomes for students to examine. However, I contend that those activities have their limitations—they do not let students generate large sets of outcomes immediately or flexibly, and students may not be assured to create complete lists

precisely. Thus, I argue that computing does indeed offer an affordance for students by providing access to a unique representation via generating entire sets of outcomes quickly and efficiently.

The use of the computer to generate output upon which students can reflect is not new, and researchers have documented affordances of this in a variety of contexts, like having students see outcomes of modeling activities, which they can then adjust (such as the predator-prey models in Wilensky & Reisman, 2006). However, the point is that we see an application of this in the context of mathematics and counting, which highlights the disciplinary-specific implications of this computational representation. Notably, here the actual outcomes and outputs are extremely important—not as a visualization or picture of overall modeling—but literally as the output of particular lines of code (such as each iteration of a loop, each print statement, etc.). Francis and Davis (2018) highlight the fact that multiplication is related to looping structures in computing, and they point out ways in which computational experiences can draw out subtleties for students in the familiar but important operation of multiplication. This common practice of producing output takes on special, added meaning in the context of combinatorics, particularly as it relates to multiplication. We know in combinatorics education that understanding and reasoning about outcomes is important (e.g., Lockwood, 2013, 2014), and here this computational thinking practice specifically aligns with a fundamental conceptual feature of enumeration.

Further, we see the reflexive nature of how mathematics can inform computational thinking practices as well. As the students reflected on particular features of these entire sets of outcomes, they engage in desirable computational thinking practices, including reusing and remixing and predicting and reflecting. For instance, in considering the structure of the overall set of outcomes (such as Charlotte and Diana's reasoning about the Coin Flips problem, or Corey and CJ's insights about the ROCKET problem), the students have to make sense of the output. This is not just a holistic understanding where they get the total value, but they must understand how to coordinate different columns and rows of outcomes. This pushes them to think about multiple stages of a process and multiple dimensions of the output, extending their reasoning beyond simple linear processes. Here, they have opportunities to reason iteration and automation, to think structurally, and to engage in activities like abstracting and generalizing. As I note in the limitations, I did not explore the impact of combinatorics on computational experiences in depth, but there is some evidence of ways in which the mathematical setting enriched the students' computational experiences.

Execution and implementation of programs as reinforcing connections among representations

In addition to affording novel representations of combinatorial concepts, the computational setting facilitated students' formulating connections between these computational representations. Because students could run the code and generate outcomes, they could directly and explicitly see that a particular bit of code yielded a specific list of outcomes. This creation of and reflection upon output is not a newly documented phenomenon; as noted in the literature review, there has been much work that has examined computational thinking practices and the use of multiple representations in light of complex systems in a variety of domains. Previous studies in computing discuss students' production of and reflection upon computer output. Often this involves modeling, where researchers and teachers seek to use features of the computer to handle systems that might be too complex or difficult to manage by hand (e.g., Danish, 2014; Wilensky & Reisman, 2006). In particular, the computer can run many simulations, offer visualizations, and help students to articulate and specify processes. However, here I examine some discipline-specific implications of such activity.

One notable feature is that the computational setting allowed for the computer to act as an external, objective arbiter of a counting process that a student might want to employ. This allows students to get outside of their own thinking and see how an objective machine would implement

their process (as articulated through computer code). We saw this in the students' work on Part 1d of the Outfits problem and on the License Plate and Coin Flips problems. This is important, because when counting by hand, it is often difficult to understand and articulate what a process is actually doing, and it is even harder to evaluate whether one's own process is correct or incorrect. Indeed, Lockwood (2014) has shown that when solving counting problems by hand, students can be convinced by a seemingly logical process that is, in fact, incorrect (others have described similar phenomena, too, such as Annin & Lai, 2010; Batanero et al., 1997). By giving a computer precise instructions to generate outcomes, and by then actually generating output of that process, students can see how different processes might yield different structures on the same set of outcomes.

Even more, the computational setting affords immediate feedback, giving students opportunities to test and experiment with different processes and what the output of these processes will be. The computational setting offered a built-in, immediate feedback loop that related code with output. In terms of Lockwood's (2013) model, I interpret that in the combinatorial context, this feedback loop allows students to see explicitly how a counting process generates and organizes a list of outcomes. This immediate feedback is bolstered by the activity of prediction, which highlights these representations and connections between them. In this way, the computational thinking practices of prediction and reflection uniquely interact with the computational setting to support important combinatorial relationships.

As a specific example, we can consider a mathematical operation like multiplication, which the students regularly used. The multiplicative structure is evident in the code, as the for loops in their programs reflected repeated addition, which is one way in which students can think about multiplication (see also Francis & Davis, 2018). It is also evident in the structure of the outcomes, and the students conveyed the multiplication in their written mathematical formulas and expressions. As noted in the literature review, combinatorics education researchers have focused on multiplication, and it happens so frequently that there is a fundamental guiding principle called the Multiplication Principle (also referred to as the Fundamental Counting Principle or the Fundamental Principle of Counting [see Lockwood et al., 2017]) that describes when multiplication is appropriate to use in solving combinatorics problems. In counting, there is an idea that multiplication works because *for each* option at a given stage in the counting process, there are a certain number of options for any subsequent stage. This is highlighted and reinforced in the computational representations, particularly with nested for loops in the code itself. Because multiplicative structure is evident across a variety of representations, this is an example in which particular combinatorial ideas are reinforced by computing.

Understanding that the computer generates outcomes according to whatever set of instructions it is given can also offer some verification of a process, which we saw in the students' work on the Outfits and License Plates problems, where they predicted, tested, and verified a process and a numerical value. Researchers have indicated that one issue with counting problems is that they are difficult to verify (e.g., Eizenberg & Zaslavsky, 2004). This feature of the computer can also help students detect errors in a process, or to recognize when a process might not produce the outcomes that are desired. We saw this in Part 1d of the Outfits problem and in Charlotte and Diana's work on the License Plate problem. The code and the output of code can thus offer some reasonability check about what a process even entails at all, which is something that researchers have suggested that students find difficult (e.g., Lockwood, 2014; Lockwood, et al., 2015).

In considering disciplinary reflexivity, the students' combinatorial experiences of predicting, running code, and reflecting on outcomes, is related to computational thinking practice of debugging and troubleshooting (e.g., Weintrop et al., 2016). Generally, I have not framed the students' work in this study as debugging because in the examples I have provided they were not having to adjust to an error in their code. Rather, they were having to consider an aspect of their combinatorial understanding, and they rarely debugged code for the sake of improving their code.

However, in considering the students' work I have presented, we can see how their engagement in the prediction and reflection cycle, particularly about the specific combinatorial constructs of counting processes that generate sets of outcomes, can potentially reflexively inform the computational practice of debugging. This is also related to one way in which Dewey spoke of reflection, which is that reflection requires attitudes that value personal growth (Rogers, 2002). The students were willing to engage in prediction, to be reflective about unexpected results, and to then use these insights to gain conceptual understanding. This is exactly the kind of disposition and practice from which students who engage in computing may benefit.

Ultimately, we know from prior research that representational heterogeneity is not uncommon in computing education, and such use of multiple representations is valuable for students as they engage in computing (e.g., Danish, 2014; Dickes et al., 2020; Farris et al., 2020). By examining students' work in a combinatorial setting, we gain particular insight into what specific kinds of representations actually were used more effectively for students, and in particular, what the computer afforded in the form of two representations that are particularly salient and important for combinatorics specifically.

Limitations

I now mention a few limitations both in terms of the study design and with regard to having students working in a computational setting as I have described. One disadvantage of considering counting processes within a computational setting is that sometimes the computer might be able to convey some process that is useful for some smaller tasks but then becomes impractical. For instance, the nested loop structure is useful for representing multiplication, and, I would argue, is important in highlighting multiplicative structure within combinatorial problems. However, nested loops quickly become impractical—to produce arrangements of all of the letters in the alphabet would require 26 nested for loops, which would be tedious work for students to code. This limitation is particularly related to the computational tools and structures used in this study. I tended to keep the structure simple, highlighting for loops and conditional statements, without many other computational structures such as recursion, and it would be possible for students to use additional coding structures to be more efficient in their code. In addition, it might be that while reusing and remixing code can be productive, if students rely on it too heavily, they might not be aware of or might be hesitant to explore additional coding structures. I note, though, that the goal of leveraging the computational setting is to provide some insight into combinatorics. Students were able to gain such insights and engage in desirable computational activity even using tools and structures that would not, in practice, generalize (such as nested for loops). Thus, while I acknowledge this limitation, I maintain that the insights it can offer on tractable problems outweigh the disadvantages.

There are also limitations of the computational setting that arise when handling sets of outcomes, and there are problems that the computer cannot easily handle. The limitations can arise both in the computational efficiency (so, some basic code like nested loops may simply not be able to have a run time that would appropriately produce all of the outputs), or there are sometimes practical limitations that emerge from the particular coding environment. For instance, we recall Diana and Charlotte's work on the License Plate problem. They generated 125,000 and got the output they expected, but PyCharm itself did not print all 125,000 outcomes, and when they scrolled up they could not see outcomes before 300eed. This was simply a limitation of the PyCharm interface and the computing power of the computer—it surprised us as interviewers, and we had to adjust in the moment. So, while the computational setting opens up many additional kinds of problems with much larger sets of outcomes, it is not a perfect extension, and there are still many problems that are not computationally feasible. Again, I do not view this as a fatal flaw, because the additional space of problems for which students *can* produce outcomes is

still much, much larger than lists they can create by hand, and being able to survey entire sets of outcomes seemed to be productive for students. I believe that it is important to note this limitation, but it does not undermine the potential value of the computational setting for enriching students' combinatorial thinking and activity.

Another point is that although I draw on disciplinary reflexivity, I have mostly focused on the ways in which computational experience supports combinatorial reasoning (and not the other direction). This is largely due to the focus of my study and to space limitations, but I acknowledge that there is more to explore about the ways in which certain domain-specific thinking might support and enrich combinatorial experiences. This is also a potential avenue that could be pursued in future research studies.

Finally, I have restricted this set of interviews to a small population of students. However, based on the students' work on these tasks, I hypothesize that other students with different mathematical backgrounds may also benefit from computing as these students did, and there is nothing unique about these students being vector calculus students that afforded their engagement with these computational thinking practices in solving combinatorial problems. Because of the accessible nature of combinatorics and the fact that these students were relatively novice counters, there is reason to believe that younger students or students in less-advanced mathematics courses could also have similar experiences. Combinatorics education researchers would benefit from more extensive work with more students and broader populations.

Implications and directions for future research

The results in this paper offer an evidence-based, theoretically grounded instance of ways in which students' engagement with computational thinking practices can be leveraged to enrich their mathematical thinking. I highlight implications and directions for future research both in terms of combinatorics and for other domains more broadly. I have offered one specific example within combinatorics, but my intention is that there may be a broader implication for other domains, which is in line with a perspective of disciplinary reflexivity.

In previous sections I have outlined implications of these findings for combinatorics. In particular, computational thinking practices provide rich and unique opportunities for students to engage with representations of counting processes and sets of outcomes and to formulate and reinforce connections among these representations. An implication of this work, then, is that researchers and teachers who focus on combinatorics may want to give students opportunities to engage in combinatorial thinking and activity within computational settings. In terms of future directions for combinatorics education research, there are many potential paths to pursue. One option would be to explore additional kinds of counting problems and combinatorial situations, and to design tasks where students can explore combinatorial ideas computationally. Researchers could take any of the many particular topics that have been studied previously (such as the multiplication principle, issues of order, combinatorial proof, overcounting, set partitions, bijections, and more) and explore how those ideas might be taught and learned when students have access to coding and programming in a computational setting. In addition, while I chose Python as a particular programming environment, there are many other specific computational contexts in which students' combinatorial thinking and activity could be explored. For example, researchers could study whether writing pseudocode has positive effects for students, or whether other languages, block-based programming, or agent-based programming afford similar combinatorial insights for students.

In this study, I designed tasks that would use code that is relatively simple, and I drew on only a few basic computational practices and structures. Even within this simple code there are components that can be examined, such as loops (whether or not they are nested), initial lists, print statements, conditional statements, variables, and more. In this paper, I do not focus on

how students understood various aspects of the code itself, and an avenue for future research would be to examine how students to reason about some of these elements, particularly in terms of mathematical ideas (within combinatorics or beyond). Another, related avenue for future research would be to incorporate tasks with more complex computational tools such as recursion, more complicated data structures, function definitions, etc. There are many more opportunities to examine additional kinds of coding structure as they relate to students' understanding of mathematical topics and ideas, in a variety of mathematical domains. Finally, there is much more to explore about the variety of representations that students use, as well as how they reason about, use, and move among them. Researchers could investigate students' experiences with multiple representations, including exploring what kinds of pedagogical interventions might effectively help students interpret, develop, and traverse representations.

Acknowledgments

The author would like to thank Zackery Reed, the editor, and the anonymous reviewers for invaluable feedback on initial versions of the manuscript, as well as Adaline De Chenne and Branwen Schaub for their work on this project.

Funding

This material is based upon work supported by the National Science Foundation under Grant [No. 1650943].

ORCID

Elise Lockwood http://orcid.org/0000-0002-4118-338X

References

Annin, S. A., & Lai, K. S. (2010). Common errors in counting problems. *The Mathematics Teacher*, *103*(6), 402–409. https://doi.org/10.5951/MT.103.6.0402

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*(2), 181–199. https://doi.org/10.1023/A:1002954428327

Benton, L., Hoyles, C., Kalas, I., & Noss, R. (2017). Bridging primary programming and mathematics: Some findings of designing research in England. *Digital Experiences in Mathematics Education*, *3*(2), 115–138. https://doi.org/10.1007/s40751-017-0028-x

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. https://doi.org/10.1016/j.ijcci.2017.12.004

Brennan, K., & Resnick, M. (2012). *New frameworks for studying and assessing the development of computational thinking*. Paper Presented at the Annual American Educational Research Association Meeting, Vancouver, Canada.

Broley, L., Caron, F., & Saint-Aubin, Y. (2018). Levels of programming in mathematical research and university mathematics education. *International Journal of Research in Undergraduate Mathematics Education*, *4*(1), 38–55. https://doi.org/10.1007/s40753-017-0066-1

Buteau, C., & Muller, E. (2017). Assessment in undergraduate programming-based mathematics courses. *Digital Experiences in Mathematics Education*, *3*(2), 97–114. https://doi.org/10.1007/s40751-016-0026-4

Buteau, C., Gueudet, G., Muller, E., Mgombelo, J., & Sacristán, A. I. (2020). University students turning computer programming into an instrument for 'authentic' mathematical work. *International Journal of Mathematical Education in Science and Technology*, *51*(7), 1020–1041. https://doi.org/10.1080/0020739X.2019.1648892

Buteau, C., Muller, E., & Marshall, N. (2015). When a university mathematics department adopted core mathematics courses of an unintentionally constructionist nature: really? *Digital Experiences in Mathematics Education*, *1*(2-3), 133–155. https://doi.org/10.1007/s40751-015-0009-x

Buteau, C., Muller, E., Marshall, N., Sacristán, A. I., & Mgombelo, J. (2016). Undergraduate mathematics students appropriating programming as a tool for modelling, simulation, and visualization: a case study. *Digital Experiences in Mathematics Education*, *2*(2), 142–166. https://doi.org/10.1007/s40751-016-0017-5

Caballero, M. D. (2015). *Computation across the curriculum: What skills are needed?* The Proceedings of the Physics Education Research Conference (pp. 79–82).

Caballero, M. D., Chonacky, N., Engelhardt, L., Hilborn, R. C., Lopez del Puerto, M., & Roos, K. R. (2019). Picup: A community of teachers integrating computation into undergraduate physics courses. *The Physics Teacher*, *57*(6), 397–399. https://doi.org/10.1119/1.5124281

Danish, J. A. (2014). Applying an activity theory lens to designing instruction for learning about the structure, behavior, and function of a honeybee system. *Journal of the Learning Sciences*, *23*(2), 100–148. https://doi.org/10.1080/10508406.2013.856793

DeJarnette, A. F. (2019). Students' challenges with symbols and diagrams when using a programming environment in mathematics. *Digital Experiences in Mathematics Education*, *5*(1), 36–58. https://doi.org/10.1007/s40751-018-0044-5

Dewey, J. (1910). *How we think*. D. C. Heath & Co.

Dickes, A. C., Farris, A. V., & Sengupta, P. (2020). Sociomathematical norms for integrating coding and modeling with elementary science: A dialogical approach. *Journal of Science Education and Technology*, *29*(1), 35–52. https://doi.org/10.1007/s10956-019-09795-7

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. https://doi.org/10.1207/s15327833mtl0601_2

English, L. D. (1991). Young children's combinatorics strategies. *Educational Studies in Mathematics*, *22*(5), 451–447. https://doi.org/10.1007/BF00367908

English, L. D. (1993). Children's strategies for solving two- and three-dimensional combinatorial problems. *Journal of Mathematical Behavior*, *24*(3), 255–273.

English, L. D. (2005). Combinatorics and the development of children's combinatorial reasoning. In G. A. Jones (Ed.), *Exploring probability in school: Challenges for teaching and learning* (Vol. 40, pp. 121–141). Kluwer Academic Publishers.

Farris, A. V., & Sengupta, P. (2014). *Perspectival computational thinking for learning physics: a case study of collaborative agent-based modeling*. Proceedings of the 11th International Conference of the Learning Sciences (ICLS 2014) (pp. 1102–1106).

Farris, A. V., Dikes, A. C., & Sengupta, P. (2020). *Grounding computational abstractions in scientific experience*. Proceedings of the 13th International Conference of the Learning Sciences (ICLS 2020) (pp. 1333–1340).

Feurzeig, W., Papert, S., & Lawler, B. (2011). Programming-languages as a conceptual framework for teaching mathematics. *Interactive Learning Environments*, *19*(5), 487–501. https://doi.org/10.1080/10494820903520040

Francis, K., & Davis, B. (2018). Coding robots as a source of instantiations for arithmetic. *Digital Experiences in Mathematics Education*, *4*(2–3), 71–86. https://doi.org/10.1007/s40751-018-0042-7

Gadanidis, G., Hughes, J. M., Minniti, L., & White, B. J. G. (2017). Computational thinking, grade 1 students and the binomial theorem. *Digital Experiences in Mathematics Education*, *3*(2), 77–96. https://doi.org/10.1007/s40751-016-0019-3

Goldin, G. A. (2014). Mathematical Representations. In S. Lerman (Ed.), *Encyclopedia of mathematics education*. Springer. https://doi.org/10.1007/978-94-007-4978-8_103

Grover, S., & Pea, R. (2013). Computational thinking in K-12: A review of the state of the field. *Educational Researcher*, *42*(1), 38– 43. https://doi.org/10.3102/0013189X12463051

Guzdial, M. (1994). Software-realized scaffolding to facilitate programming for science learning. *Interactive Learning Environments*, *4*(1), 001–044. https://doi.org/10.1080/1049482940040101

Guzdial, M., & Naimipour, B. (2019). *Task-specific programming languages for promoting computing integration: A precalculus example*. In 19th Koli Calling International Conference on Computing Education Research (Koli Calling '19), November 21–24, 2019, Koli, Finland. ACM. https://doi.org/10.1145/3364510.3364532

Hadar, N., & Hadass, R. (1981). The road to solving a combinatorial problem is strewn with pitfalls. *Educational Studies in Mathematics*, *12*(4), 435–443. https://doi.org/10.1007/BF00308141

Hambrush, S., Hoffman, C., Korb, J. T., Maugan, M., & Hosking, A. L. (2009). *A multidisciplinary approach towards computational thinking for science majors*. ACM Sigcse Bulletin. https://doi.org/10.1145/1508865.1508931

Hammond, T. C., Oltman, J., & Salter, S. (2019). Using computational thinking to explore the past, present, and future. *Social Education*, *83*(2), 118–122.

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* (pp. 1039–1045). Zdm.

Kafai, Y. B. (2016). From computational thinking to computational participation in K-12 education: Seeking to reframe computational thinking as computational participation. *Communications of the ACM*, *59*(8), 26–27. https://doi.org/10.1145/2955114

Kapur, J. N. (1970). Combinatorial analysis and school mathematics. *Educational Studies in Mathematics*, *3*(1), 111–127.

Knochel, A. D., & Patton, R. M. (2015). *If* art education *then* critical digital making: computational thinking and creative code. *Studies in Art Education*, *57*(1), 21–38. https://doi.org/10.1080/00393541.2015.11666280

Kotsopoulos, D., Floyd, L., Khan, S., Kizito Namukasa, I., Somanath, S., Weber, J., & Yiu, C. (2017). A pedagogical framework for computational thinking. *Digital Experiences in Mathematics Education*, *3*(2), 154–171. https://doi.org/10.1007/s40751-017-0031-2

Lim, K. H. (2006). Characterizing students' thinking: Algebraic inequalities and equations. In S. Alatorre, J. L. Cortina, M. Sáiz, & A. Méndez (Eds.), *Proceedings of the 28th Annual Meeting of the North American Chapter of the Psychology of Mathematics Education* (Vol. 2, pp. 102–109). Universidad Pedagógica Nacional.

Lim, K. H., Buendía, G., Kim, O. K., Cordero, F., & Kasmer, L. (2010). The role of prediction in the teaching and learning of mathematics. *International Journal of Mathematical Education in Science and Technology*, *41*(5), 595–608. https://doi.org/10.1080/00207391003605239

Lockwood, E. (2011). Student connections among counting problems: An exploration using actor-oriented transfer. *Educational Studies in Mathematics*, *78*(3), 307–322. https://doi.org/10.1007/s10649-011-9320-7

Lockwood, E. (2013). A model of students' combinatorial thinking. *The Journal of Mathematical Behavior*, *32*(2), 251–265. https://doi.org/10.1016/j.jmathb.2013.02.008

Lockwood, E. (2014). A set-oriented perspective on solving counting problems. *For the Learning of Mathematics*, *34*(2), 31–37.

Lockwood, E., & De Chenne, A. (2020). Using conditional statements in Python to reason about sets of outcomes in combinatorial problems. *International Journal of Research in Undergraduate Mathematics Education*, *6*(3), 303–346. https://doi.org/10.1007/s40753-019-00108-2

Lockwood, E., & De Chenne, A. (2021). Reinforcing key combinatorial ideas in a computational setting: A case of encoding outcomes in Python programming. *The Journal of Mathematical Behavior*, *62*, 100857. https://doi.org/10.1016/j.jmathb.2021.100857

Lockwood, E., & Gibson, B. (2016). Combinatorial tasks and outcome listing: Examining productive listing among undergraduate students. *Educational Studies in Mathematics*, *91*(2), 247–270. https://doi.org/10.1007/s10649-015-9664-5

Lockwood, E., & Purdy, B. (2019). Two undergraduate students' reinvention of the multiplication principle. *Journal for Research in Mathematics Education*, *3*(50), 225–267.

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

Lockwood, E., Reed, Z., & Erickson, S. (2021). Undergraduate students' combinatorial proof of binomial identities. *Journal for Research in Mathematics Education*, *52*(5), 539–580. https://doi.org/10.5951/jresematheduc-2021-01

Lockwood, E., Reed, Z., & Caughman, J. S. (2017). An analysis of statements of the multiplication principle in combinatorics, discrete, and finite mathematics textbooks. *International Journal of Research in Undergraduate Mathematics Education*, *3*(3), 381–416. https://doi.org/10.1007/s40753-016-0045-y

Lockwood, E., Swinyard, C. A., & Caughman, J. S. (2015). Patterns, sets of outcomes, and combinatorial justification: Two students' reinvention of counting formulas. *International Journal of Research in Undergraduate Mathematics Education*, *1*(1), 27–62. https://doi.org/10.1007/s40753-015-0001-2

Magana, A. J., Falk, M. L., & Reese, M. J. Jr., (2013). Introducing discipline-based computing in undergraduate engineering education. *ACM Transactions on Computing Education*, *13*(4), 116–122. https://doi.org/10.1145/2534971

Magana, A. J., Falk, M. L., Vieira, C., & Reese, M. J. Jr., (2016). A case study of undergraduate engineering students' computational literacy and self-beliefs about computing in the context of authentic practices. *Computers in Human Behavior*, *61*, 427–442. https://doi.org/10.1016/j.chb.2016.03.025

Maher, C. A., Powell, A. B., & Uptegrove, E. B. (Eds.). (2011). *Combinatorics and Reasoning: Representing, Justifying, and Building Isomorphisms*. Springer.

Naimipour, B., Guzdial, M., & Shreiner, T. (2019). *Helping social studies teachers to design learning experiences around data: Participatory design for new teacher-centric programming languages* [Paper presentation]. Proceedings of ACM ICER conference (ICER'19), August 2019, Toronto, Canada. https://doi.org/10.1145/3291279.3341211

Odden, T. O. B., Lockwood, E., & Caballero, M. D. (2019). Physics computational literacy: An exploratory case study using computational essays. *Physical Review Physics Education Research*, *15*(2), 020152. https://doi.org/10.1103/PhysRevPhysEducRes.15.020152

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books.

Reed, Z., & Lockwood, E. (2021). Leveraging a categorization activity to facilitate productive generalizing activity and combinatorial reasoning. Online first. *Cognition and Instruction*, *39*(4), 409–450. https://doi.org/10.1080/07370008.2021.1887192

Rogers, C. (2002). Defining reflection: Another look at John Dewey and reflective thinking. *Teachers College Record*, *104*(4), 842–866.

Sengupta, P., Dickes, A., & Farris, V. (2018). Toward a phenomenology of computational thinking in K-12 STEM. In M. Khine (Ed.), *Computational thinking in the STEM disciplines*. Springer. https://doi.org/10.1007/978-3-319-93566-9_4

Sengupta, P., Kinnebrew, J. S., Basu, S., Biswas, G., & Clark, D. (2013). Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies*, *18*(2), 351–380. https://doi.org/10.1007/s10639-012-9240-x

Sigel, I. A. (1981). Social experience in the development of representational thought: Distancing theory. In I. E. Sigle, D. M. Brodzinsky, & R. M. Golinkoff (Eds.), *New directions in Piagetian theory and practice*. Lawrence Erlbaum.

Sinclair, N., & Patterson, M. (2018). The Dynamic Geometrisation of Computer Programming. *Mathematical Thinking and Learning*, *20*(1), 54–74. https://doi.org/10.1080/1096065.2018.1403541

Steffe, L. P., & Thompson, P. W. (2000). Teaching experiment methodology: Underlying principles and essential elements. In R. Lesh & A. E. Kelly (Eds.), *Research design in mathematics and science education*. Lawrence Erlbaum Associates.

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

Tillema, E. S. (2013). A power meaning of multiplication: Three eighth graders' solutions of Cartesian product problems. *The Journal of Mathematical Behavior*, *32*(3), 331–352. https://doi.org/10.1016/j.jmathb.2013.03.006

Tillema, E. S., & Gatza, A. (2016). A quantitative and combinatorial approach to non-linear meanings of multiplication. *For the Learning of Mathematics*, *36*(2), 26–33.

Vieira, C., Magana, A. J., Roy, A., & Falk, M. L. (2019). Student explanations in the context of computational science and engineering education. *Cognition and Instruction*, *37*(2), 201–231. https://doi.org/10.1080/07370008.2018.1539739

Wagh, A., Levy, S., Horn, M., Guo, Y., Brady, C., & Wilensky, U. (2017). Anchor code: Modularity as evidence for conceptual learning and computational practices of students using a code-first environment. In CSCL *proceedings* (pp. 656–659). International Society of the Learning Sciences.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology*, *25*(1), 127–147. https://doi.org/10.1007/s10956-015-0581-5

Wheatley, G. (1992). The role of reflection in mathematics learning. *Educational Studies in Mathematics*, *23*(5), 529–541. https://doi.org/10.1007/BF00571471

Wilensky, U., & Reisman, K. (2006). Thinking like a wolf, a sheep, or a firefly: Learning biology through constructing and testing computational theories—An embodied modeling approach. *Cognition and Instruction*, *24*(2), 171–209. https://doi.org/10.1207/s1532690xci2402_1

Wing, J. M. (2016). *Computational thinking: 10 years later*. https://phys.org/news/2016-03-years.html#jCp.

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions. Series A, Mathematical, Physical, and Engineering Sciences*, *366*(1881), 3717–3725. https://doi.org/10.1098/rsta.2008.0118

Young, N. T., Allen, G., Aiken, J. M., Henderson, M., & Caballero, M. D. (2019). Using Random Forests to determine important features for integrating computation into physics courses. *Physical Review Physics Education Research*, *1*, 010114.