Journal of Mathematical Behavior 62 (2021) 100857

Contents lists available at ScienceDirect

Journal of Mathematical Behavior

journal homepage: www.elsevier.com/locate/jmathb

Reinforcing key combinatorial ideas in a computational setting: A case of encoding outcomes in computer programming

Elise Lockwood [a,b,*], Adaline De Chenne [c]

[a] Oregon State University, 064 Kidder Hall, Department of Mathematics Corvallis, OR, 97331, United States
[b] University of Oslo, Centre for Computing in Science Education (CCSE), Sem Saelands vei 24, Fysikkbygningen, Oslo, 0371, Norway
[c] Oregon State University, 320 Kidder Hall, Department of Mathematics Corvallis, OR, 97331, United States

ARTICLE INFO

Keywords:
Combinatorics
Encoding outcomes
Computation
Programming
Discrete mathematics

ABSTRACT

Counting problems are difficult for students to solve, and there is a perennial need to investigate ways to help students solve counting problems successfully. One promising avenue for students' successful counting is for them to think judiciously about how they encode outcomes – that is, how they symbolize and represent the outcomes they are trying to count. We provide a detailed case study of two students as they encoded outcomes in their work on several related counting problems within a computational setting. We highlight the role that a computational environment may have played in this encoding activity. We illustrate ways in which by-hand work and computer programming worked together to facilitate the students' successful encoding activity. This case demonstrates ways in which the activity of computation seemed to interact with by-hand work to facilitate sophisticated encoding of outcomes.

1. Introduction and motivation

Combinatorial enumeration problems, or "counting problems," ask for the cardinality of a set of outcomes that satisfy particular conditions. Such problems are known to be difficult for students (e.g., Annin & Lai, 2010; Batanero et al., 1997; Hadar & Hadass, 1981; Lockwood & Gibson, 2016), and yet they have useful applications (Kapur, 1970; Tucker, 2002) and can facilitate rich mathematical thinking and practices (e.g., Kapur, 1970; Lockwood, 2011; Lockwood & Reed, 2018; Maher, Powell, & Uptegrove, 2011). In this paper, we explore one phenomenon involving students' combinatorial reasoning within a computational setting.

When solving counting problems, we must decide how to represent the outcomes we are trying to count. Our chosen representations of outcomes must contain all the information we need in order to count those outcomes, and we must create and use representations that encode all relevant features of the outcomes for the purpose of determining the number of outcomes. Depending on the problem, some information is irrelevant, such as physical features of objects (whether they are cats, people, numbers, etc.), but other information is essential, such whether objects are distinct. Typically, in combinatorics, we prefer representations that emphasize those features that are relevant for counting. We refer to this activity of representing or symbolizing outcomes as *encoding* outcomes. Importantly, often we can encode outcomes in such a way that illuminates how we can apply already known counting techniques to solve new problems.

Consider a problem in which we want to count arrangements of five people: *How many ways are there to arrange the five people Diego, Michelle, Brian, Knut, and Latoya in a line?* To solve this, we must first choose a way to encode each arrangement of the five people. Does

* Corresponding author at: Oregon State University, 064 Kidder Hall, Department of Mathematics Corvallis, OR, 97331, United States.
E-mail addresses: Elise.Lockwood@oregonstate.edu (E. Lockwood), dechenna@oregonstate.edu (A. De Chenne).

https://doi.org/10.1016/j.jmathb.2021.100857
Received 28 January 2020; Received in revised form 20 January 2021; Accepted 11 February 2021
Available online 1 March 2021

it matter if we encode the people as entire strings of letters or as single letters? As numbers? As something else? What features of the outcomes do we need to consider when we represent an outcome? These kinds of questions relate to the decisions we make when encoding outcomes. Sometimes, different ways of encoding outcomes do not have a meaningful effect on the solution to a problem – when counting the number of arrangements of 5 people, it does not much matter if we think of them as strings, numbers, letters, or objects. However, sometime decisions about encoding can affect one's ability to solve a problem more or less efficiently, or at all.

For example, suppose that we already know how to count 5-element subsets of the numbers 1 through 10. We can simply choose 5 of the 10 elements, so there are $\binom{10}{5}$, or $\frac{10!}{5!5!} = 252$ total outcomes. Suppose we are then posed with the Heads/Tails problem (adapted from Lockwood et al., 2018): *You are going to flip a coin 10 consecutive times. How many different outcomes of this process will have exactly five heads and five tails?* Our outcomes of this process are sequences of five heads and five tails, where different orderings of the sequences yield distinct outcomes. One way to encode outcomes is simply to write the strings as Hs and Ts, so HHHHHTTTTT and HHTHTHTHTT are examples of different outcomes. Counting the number of such strings may seem difficult if we have not seen such a problem before. But, by thinking of the outcomes in a certain way, we can recognize that any such outcome can be encoded as a set of five numbers that describe the positions in which the Hs are located. For instance, HHTHTTHTTH corresponds to the set {1, 2, 4, 7, 10}. There is thus a mapping of strings of five Hs and five Ts to a set of five distinct numbers. We can now solve this problem by considering 10 distinct positions and selecting which five positions will contain Hs. This reduces the problem to one we already know: counting 5-element subsets from the numbers 1 through 10.

This example is meant to highlight that there are decisions one can make about encoding outcomes that can make problems more or less easy to solve. As we will elaborate, there is research that shows that students can struggle to solve problems like the Heads/Tails problem because they do not think to encode outcomes as sets of positions (Lockwood et al., 2018). Being able to encode outcomes productively involves, first, understanding that a counter has the freedom to decide how to encode outcomes – there is no one correct way to encode an outcome for a problem. It also involves developing techniques for creating mappings or identifying properties of certain counting situations and outcomes. Further, there are classical, and often non-obvious, ways of encoding that can be learned and applied. Judicious encoding of outcomes, then, is a very useful tool to help students successfully solve counting problems. In this paper, we explore the phenomenon of encoding outcomes, and we investigate the role that a computational setting may play in facilitating sophisticated encoding of outcomes for students.

We offer a case of one pair of students and their engagement with encoding on three types of counting problems in a computational setting. This case demonstrates ways in which students effectively engaged in encoding activity, and it also explores how the activity of computation interacted with by-hand work to facilitate sophisticated encoding of outcomes. Through this case, we inform research on this important combinatorial activity of encoding outcomes, and we offer an example of potential ways in which computing may be used to help students encode outcomes and solve counting problems successfully. We attempt to answer the following research questions: *In what ways did the encoding activity of a pair of undergraduate students who were working within a computational setting affect their combinatorial thinking and activity? What role did the computer program and output play in their encoding activity?*

2. Literature review and theoretical perspectives

In this section, we present literature and mathematical discussion related to encoding outcomes in some fundamental combinatorial situations. We then elaborate three kinds of problems that involve sophisticated encoding that emerged in this study, and we discuss the literature on these problem types as well as key mathematical ideas. Finally, we discuss the computational setting of this study, and we argue for the appropriateness of this computational setting as a way to enrich students' understanding of encoding outcomes.

2.1. Encoding outcomes: relevant literature and mathematical ideas

2.1.1. Literature on encoding outcomes

By *encoding outcomes*, we mean determining a particular way of representing or symbolizing an outcome of a counting process (in the sense of Lockwood, 2013) that preserves information about the nature of the outcomes so as not to alter the cardinality of the set of outcomes. We use the term encoding (rather than representing or symbolizing more broadly) both because it signifies the specific combinatorial process of articulating outcomes, and because such activity has been referred to as encoding elsewhere in the combinatorial literature (e.g., Lockwood et al., 2015; Lockwood et al., 2018). As noted in the introductory examples, encoding outcomes is important because different ways of writing outcomes can be more or less productive when solving problems. The phenomenon of encoding outcomes has come up in the literature in a couple of ways. First, thinking carefully about how outcomes might be expressed and organized is related to Lockwood's (2013, 2014a, 2014b) view that reasoning about outcomes is an essential aspect of students' combinatorial thinking and activity, and our findings are situated within work that advocates for students' focus on sets of outcomes. Lockwood (2014a) discussed a "set-oriented perspective," which is "a way of thinking about counting that involves viewing an explicit focus on sets of outcomes as a fundamental aspect of solving counting problems" (p. 32). Because it can be difficult to detect errors in counting processes that seem logically sound, sets of outcomes can offer "a powerful resource for making sense of such discrepancies" (Lockwood, 2014b, p. 298).

In addition, Lockwood and Gibson (2016) discussed the role of listing outcomes in solving counting problems, demonstrating a positive correlation among novice counters between systematic listing activity and correctly solving counting problems. Lockwood and

Gibson also qualitatively examined what made for productive listing, and one of their findings was that *useful notation and appropriate modeling of outcomes* was a feature of productive lists (by which they meant lists that were used on a problem that was solved correctly). These researchers suggest that encoding outcomes is an important aspect of combinatorial problem solving; we build on such work by addressing encoding explicitly, extending such work into a computational setting.

2.1.2. The role of mapping, bijections, and isomorphisms in encoding outcomes

When encoding outcomes, often it is useful to create mappings between isomorphic situations, particularly leveraging bijections between sets of outcomes. In the Heads/Tails problem, the creation of a mapping between outcomes like HHTHTTHHTT and a set of numbers {1, 2, 4, 7, 8} for locations of Hs allows us to see that counting arrangements of five Hs and five Ts is equivalent to counting 5-element subsets of a 10-element set. In this way, creating such a mapping is a very useful tool for encoding outcomes productively. When students engage in encoding, they are essentially creating a bijection between two sets, and often it is between a set they do not know how to count and a set they do know how to count. In light of the important role that mapping and isomorphism can play in encoding outcomes, we briefly discuss research on these topics within combinatorics.

Some researchers have specifically focused on bijections and isomorphisms within the teaching and learning of combinatorics (e.g., Maher, Powell, & Uptegrove, 2011; Mamona-Downs & Downs, 2004; Speiser, 1997). These researchers offer examples in which students made connections between particular problems over time, suggesting that they viewed the problems as being isomorphic in some way. Maher et al. (2011) and colleagues report on students' creating isomorphisms among problem types. They note that in their longitudinal study, "In high school, students began the process of building isomorphisms, using their own notation as well as standard notation to describe how some problems were related to each other and ultimately to Pascal's triangle" (Maher et al., 2011, p. 14). For example, Muter and Uptegrove describe students who had previously worked on a towers problem that involved "determining how many towers can be built of various heights when selecting from cubes of various numbers of colors" (Maher et al., 2011 p. xv). When they worked on a problem involving counting the number of pizzas with certain combinations of toppings[1], one student said, "Everything we ever do always is like the tower problem" (Muter & Uptegrove, 2011, p. 107). Muter and Uptegrove go on to report that students could identify relationships between towers made from blue and red cubes, pizzas with certain toppings, and binary sequences. Tarlow (2011) similarly reports on students connecting problems involving pizzas and towers, as well as settings involving binomial coefficients. The students "constructed a three-way isomorphism between the tower problem, the pizza problem, and the numbers on Pascal's Triangle" (p. 130). In sum, recognizing and leveraging similarity and creating isomorphisms among counting situations is a powerful (often necessary) aspect of solving counting problems correctly.

2.2. Three types of counting problems that involve sophisticated encoding of outcomes

Prior research has demonstrated that some categories of counting problems can be especially difficult for students, particularly related to encoding outcomes. In designing tasks for this study, we wanted to examine how a computational setting might enrich the students' encoding activity on such problems. We discuss literature and mathematical ideas related to three problem types that can be especially difficult to encode: a) problems that are modeled as distribution problems, b) "Category II" combination problems, and c) arrangement with restricted repetition problems.

2.2.1. Problems modeled as distribution problems

Batanero et al. (1997) (drawing on Dubois, 1984) articulate differences between three basic models of combinatorial problems – a selection model, a distribution model, and a partition model. In the selection model, "a set of m (usually distinct) objects is considered, from which a sample of n elements must be drawn" (p. 183). Here, problems are stated in terms of selecting or sampling some number of objects from a set of objects. This selection model does not mean the problem must only count unordered sets of objects – a sampling model could be used to count arrangements with unrestricted repetition. They say that, "Another type of problems refers to the **distribution** of a set of n objects into m cells" (p. 184, emphasis in original). In the distribution model, problems are phrased in terms of distributing objects out in some way. Here again, objects and cells might be identical or distinct, and this reflects different combinatorial operations. Finally, they say, "we might also be interested in splitting a set of n objects into m subsets, that is, in performing a **partition** of the set" (p. 185, emphasis in original). In these problems, sets of objects are divided into subsets.

These models do not necessarily describe different problems, and many problems can be framed in term of each model. For example, a selection problem may state, *You have 5 friends, and you want to give them 3 identical hats. In how many ways can you pick which 3 of your 5 friends will get hats?* This problem can also be stated in terms of a distribution model, *You have 5 friends, and you want to give them 3 identical hats. In how many ways can distribute hats to 3 of your 5 friends?*, and in terms of the partition model, *How many ways can you separate your 5 friends into a group of 3 (who will receive a hat) and a group of 2 (who will not receive a hat)*? These different statements describe the same physical scenario (giving hats to friends), and the answer to each problem is the same.

Batanero et al. (1997) "noticed that some pupils who could apply the definition of the combinatorial operation for the selection model were not able to transfer this definition, when changing the problem to a different combinatorial model" (p. 196). The

[1] Maher et al. (2011) describe 5-topping pizza problem as, "The local pizza shop offers a plain cheese pizza. On this cheese pizza, you can place up to five different toppings. How many pizzas is it possible to make?" (Maher et al., 2011, p. 13). They describe a height 4 towers problems as, "Your group has two colors of Unifix cubes. Work together and make as many different towers four cubes tall as is possible when selecting from two colors. See if you and your partner can plan a good way to find all the towers four cubes tall" (Maher et al., 2011, p. 12).

relationship between the selection and distribution model is particularly relevant for this study, because encoding outcomes often involves translating between different models. We draw on this study to emphasize that other researchers have acknowledged this distinction among combinatorial models and have made note of distribution problems in particular. Mathematically, these distribution problems offer an opportunity for discussing encoding of outcomes. As noted above, we can frame a situation involving giving hats to people both in terms of selecting friends to receive hats, and in terms of distributing hats to friends; in each case we are counting 3-element subsets of 5 people who will receive the hats. This may not seem like a meaningful distinction (particularly for an experienced counter), but Batanero et al. (1997) suggest that students do differentiate among models. Among a list of instructional recommendations, they say that teaching "should also emphasize the translation of combinatorial problems into the different models […] instead of the mere centering on algorithmic aspects and on definitions of combinatorial operations" (p. 196).

Given that students may be more familiar with selection models (as Batanero et al.'s [1997] findings suggest), one strategy to help students solve more counting problems successfully is to develop skills to translate from selection to distribution models. We chose problems within a distribution rather than partition model because they aligned with the combinatorial operations that we targeted in this study. Further, to our knowledge, previous research has not explicitly studied how these models might be programmed or solved with a computer.

2.2.2. "Category II" combination problems

In designing tasks for this study, we also wanted to investigate students' encoding on what Lockwood et al. (2018) refer to as "Category II" combination problems. Lockwood et al. hypothesized that students might conceive of two different categories of combination problems. Specifically, Category I combination problems "involve an *unordered* selection of *distinguishable* objects (p. 308, emphasis in original). These are combination problems framed in terms of a selection model; an example is the Basketball problem: *There are 12 athletes who try out for the basketball team – which can take exactly 7 players. How many different basketball team rosters could there be?* (p. 309). They then note that, "In other situations … combination problems may still appropriately be solved using a binomial coefficient, but there is a different encoding of outcomes as sets of distinct objects" (p. 308). An example is the Coin Flips problem: *Fred flipped a coin 5 times, recording the result (Head or Tail) each time. In how many different ways could Fred get a sequence of 5 flips with exactly 3 Heads?* (p. 309). Lockwood et al. call such problems Category II problems, which they say, "can naturally be modeled as *(ordered) sequences* of *indistinguishable* objects, but which can be encoded so as to be solved via a binomial coefficient" (p. 309, emphasis in original). Note, this is a different distinction than the models that Batanero et al. (1997) describe, as Lockwood et al. were particularly focused on combination problems.

We highlight two findings from Lockwood et al. (2018) that inform our current work. First, they found that "the participants were about twice as likely to attempt to use a combination on Category I problems than Category II problems" (p. 316). Second, while few students even attempted to use combinations on Category II problems, those that did were frequently incorrect on such problems. Essentially, students overwhelmingly recognized Category I as necessitating the combinations formula, but they did not recognize Category II as necessitating the combinations formula; if they did, they did not typically apply it correctly. This suggests that students may not see this other type of problem as also involving combinations. Lockwood et al. suggest that this issue may be related to how students modeled or encode the outcomes, saying, "one possible explanation for our findings is that students are not recognizing that outcomes of Category II problems, which may be naturally modeled as ordered sequences of two (or more) indistinguishable objects, can be appropriately encoded as *sets* of objects" (p. 317, emphasis in original). If one can model the outcomes as counting sets of distinct positions (which can be thought of as distinct numbers), then Category II problems can be translated into isomorphic problems of selecting subsets from a set of numbers (which is a more familiar problem).

Lockwood et al. (2015) reported similar findings in a report of a pair of undergraduate students who solved counting problems in a 10 -h teaching experiment. In that study, the pair of students solved almost all of their counting problems correctly, but they struggled to solve one particular Category II problem. Lockwood et al. (2015) suggest that the students had difficulty because they struggled to encode the outcomes as something they knew how to count. Given students' difficulties with Category II problems, we wanted to explore how students might encode such problems in a computational setting.

2.2.3. Arrangement with restricted repetition problems

A third kind of problem that can require sophisticated encoding of outcomes is arrangement with restricted repetition problems. Such problems involve arranging different types of objects, where some of the objects are repeated a certain number of times. For instance, a problem we gave students says: *How many different arrangements are there of letters in the word BANANA?* Here we have different types of objects (B, A, and N) but some of those types of objects (three As and two Ns) are repeated and considered indistinguishable. There are a couple of ways to solve this problem. One is to leverage equivalence and division to count arrangements (Lockwood & Reed [2018] discussed this approach to the BANANA problem from another study). This labeling strategy is described by CadwalladerOlsker (2013): "If the objects are indistinguishable, one approach is to make the objects distinct by temporarily applying "labels," then later removing those labels" (p. 931). On the BANANA problem, this technique involves first treating all characters as distinct. There are 6! ways to arrange such characters, and we can divide by 3! to account for identical arrangements of the As and divide by 2! to account for identical arrangements of Ns. Thus, the total arrangements of the letters in the word BANANA is $\frac{6!}{3!\cdot 2!}$.

There is another way to solve the problem, which involves successively selecting positions in which to place identical letters. In the BANANA problem, we could consider 6 distinct positions, and then we could select three of those positions in which we might place the As. For any placement of the As, we could then select two of the remaining three positions to place the Ns, and then we could place the B in the last remaining position. This would give a total of $C\binom{6}{3} \cdot C\binom{3}{2} \cdot C\binom{1}{1}$.[2] Technically this kind of problem could also be seen as a Category II problem, in the sense of Lockwood et al. (2018), as we use a combinations formula to solve a problem about arranging sets of identical elements.

This second solution strategy for these problems, which yields products of binomial coefficients, involves sophisticated encoding of outcomes. In such a solution, there is a step of having to encode an outcome as involving selecting distinct positions. There is an implicit bijection between an arrangement of letters (like AANANB) and sets of positions that represent for respective letters (like {{1,2,4}, {3,5}, {6}}), and it is necessary for students to establish this relationship to be able to understand and apply such a solution. As we discuss further in Section 2.3, we hypothesize that a computational setting can be a useful means by which to draw attention to such a mapping that a student might make.

2.3. Encoding outcomes in a computational setting

In this section, we describe what we mean by a computational setting and computational activity, and we motivate our use of a computational setting for encoding outcomes in particular. We have previously characterized this overall project in Lockwood & De Chenne (2020).

2.3.1. Mathematics education research involving computing

There is a history of (and continued motivation for) examining student's mathematical learning in computational settings, both in K-12 settings (e.g., Benton et al., 2018; DeJarnette, 2019; Feurzeig et al., 2011; Papert, 1980; Sinclair & Patterson, 2018) and in post-secondary contexts (Buteau & Muller, 2017; Cetin & Dubinsky, 2017; Fenton & Dubinsky, 1996; Lockwood & De Chenne, 2020; Lockwood et al., 2019). The field has also offered broad surveys, frameworks, and recommendations (DiSessa, 2018; Hickmott et al., 2018; Hoyles & Noss, 2015; Kotsopoulos et al., 2017). In addition, researchers (e.g., Weintrop et al., 2016) and policymakers (e.g., NGSS Lead States, 2013) emphasize the importance of computing in STEM, and Blikstein (2018) offers a number of rationales for integrating computer science concepts into educational settings, including increasing job opportunities for students and developing computationally literate citizens who can be active participants of society. Wing (2016) argues that computation could be considered a third pillar of science (together with theory and experimentation). Simply put, computing is as relevant and accessible as it ever has been, both in STEM fields and in society. Even with such existing studies and projects that emphasize the importance of computing, there is a need for more examples of the relationship between computational activity and students' mathematical thinking and activity. We view our specific focus on programming as situated within but distinct from research that investigates on the role of technology in the teaching and learning of mathematics (e.g., Drijvers et al., 2010; Hollenbrands, 2007; Kaput & Thompson, 1994). In the next section, we describe how we are taking a computational setting and computational activity.

2.3.2. Computational setting and activity

In this paper, we draw on the taxonomy of computational thinking in mathematics and science from Weintrop et al. (2016). This taxonomy characterizes practices within mathematics and science that reflect and foster computational thinking. In our study, the computational practice with which students engaged was primarily programming in Python. This involved several elements from the taxonomy of Weintrop et al., including preparing problems for computational solutions, troubleshooting and debugging, and assessing different approaches/solutions to a problem, but we focus on programming. We adopt the characterization of programming from Weintrop et al. (2016), which "consists of understanding and modifying programs written by others, as well as composing new programs or scripts from scratch" (p. 139). In the programming activity in which our students engaged, they wrote their own programs and interpreted and modified existing programs. The students also engaged in other practices described in the taxonomy, such as using computational models to understand a concept and constructing computational models, but we focus on their programming activity. We present this taxonomy to connect our students' computational activity in a computational setting to existing ways of framing such activity.

In light of the definition of programming given by Weintrop et al., we characterize our use of the terms *computational setting* and *computational activity* in this paper (we have offered similar characterizations in Lockwood & De Chenne (2020)). The computational setting involved an environment in which students solved counting problems while programming in the Python language. The students worked together on a desktop on which Python was installed, and the tasks they were given were written both on paper and in an interpreter called PyCharm. We follow Lockwood & De Chenne (2020) and take computational activity to be the students' activity while they were working within this computational setting. Such activity included a variety of actions, including predicting outcomes, programming and running code, reflecting upon sets of outcomes, writing expressions, interpreting or modifying code, and connecting representations. We consider all such activities to have occurred within this computational setting because the computer was an integral component of the design and implementation of the interviews (not just an optional tangential tool).

[2] Note that we can place the respective letters in any order we choose. Another viable solution would be first to place the Ns, then the B, then the As, yielding an expression of $\binom{6}{2} \cdot \binom{4}{1} \cdot \binom{3}{3}$.

2.3.3. Encoding outcomes in a computational setting

We conclude this section by discussing encoding activity within a computational setting. Much of this discussion will be demonstrated in the results, but we briefly describe theoretical motivation for how and why the computational setting might be a particularly appropriate context for encoding activity. We draw on the example of the Heads/Tails problem from the introduction.

When we are working with a computer (as with any tool), we are limited by our own knowledge of that tool. We may be particularly motivated to leverage successful prior work with that tool, and thus to make efforts to reformulate a new situation in terms of a previous familiar situation. For novice programmers, we hypothesized that they would be especially motivated to translate difficult-to-code outcomes as something more familiar, even if this was due to limitations of their computational knowledge and experience. For instance, on the Heads/Tails problem, students might know how to write code to count the number of 5-element subsets from the numbers 1 through 10. They might not have the coding knowledge to code 10-character strings of Hs and Ts that have exactly five Hs and five Ts. However, if they recognize that they could instead simply count positions in which the Hs go, they could use their code that lists 5-element subsets instead. If they already know how to count subsets of numbers, then the translation would be beneficial for them. In this way, we hypothesized that students might find ways to create mappings and leverage code they have previously written. Because isomorphisms are so powerful in combinatorics, there are many possibilities for students to write code that draws on an isomorphic problem that they have previously completed. In addition, some aspects of combinatorial objects lend themselves particularly well to encoding outcomes using a computer. We drew on this feature of combinatorics as we designed tasks.

3. Methods

We present a case of one pair of students who participated in a 16.5 h-long teaching experiment over the course of seven weeks during one academic term. We focus on their work on three different types of problems (seven problems in total), which occurred in Sessions 5, 7, and 9 of the experiment. We have described these participants and data collection elsewhere (Lockwood & De Chenne, 2020), but here we emphasize problems in which they engaged in encoding outcomes.

3.1. Participants and data collection

We employed a teaching experiment (TE) methodology for this study. Steffe and Thompson (2000) note that a primary purpose of teaching experiments is "for researchers to experience, firsthand, students' mathematical learning and reasoning" (p. 267). A TE extends a clinical interview by attempting to understand not just a student's current knowledge, but the progress students make over time, and it is designed "for the exploration and explanation of students' mathematical activity" (p. 273). In addition to designing and giving students tasks that we hypothesized would elicit certain ways of understanding for the students, we at times intervened with pointed questions that we hoped would draw their attention to certain aspects of their work. On some occasions, if the students appeared to be stuck, we intervened and asked direct questions related to their encoding of outcomes. Thus, our claims in the results of this paper are not that the students *spontaneously* encoded outcomes in sophisticated ways – sometimes they did, but sometimes their encoding was the result of a prompt by the interviewer. Even with such prompts, we gain first-hand, detailed insight into their thinking and activity.

The students (Diana and Charlotte, pseudonyms) were first- and second-year chemistry majors, respectively, who were recruited from a vector calculus class.[3] They were two of 13 students who participated in individual selection interviews, which revealed that they did not have prior experience counting,[4] they had no programming experience, and they were able to solve problems and articulate their thinking. The TE consisted of 11 teaching episodes that were each 60−90 min long, and they occurred over the seven weeks of a single academic term.[5]

During the teaching episodes, Charlotte and Diana sat at the interviewer's desk and worked at a large desktop, using the programming environment PyCharm (JetBrains, 2017). The students could type code into a window, and the output of the code would appear in an adjacent window after they ran it. The students had writing materials, and problem statements were printed out and also written in PyCharm. We video and audio recorded the students as they worked, and we captured their computer work by recording the screen. The students worked well together discussing the problems, and they informally took turns typing and writing answers by hand.

3.2. Tasks

Generally, the TE covered a number of combinatorial topics and was geared toward engaging students in combinatorial thinking and activity within a computational setting (more details are provided in Lockwood & De Chenne [2020]). Table 1 shows a broad progression of the targeted concepts, along with representative kinds of problems that examined each targeted concept. The students

[3] We have previously had success with interviewing vector calculus students for studies involving combinatorics, as they tend not to try to recall formulas but still have some level of mathematical maturity.

[4] Selection interviews revealed that they were not familiar with common formulas or symbols associated with counting, and they did not demonstrate prior knowledge of combinatorial formulas or concepts by, for example, trying to recall formulas. For more details on the selection interviews see Lockwood & De Chenne (2020).

[5] Interviews occurred on TR of Weeks 1, 2, and 3, T of Week 4, TR of Week 5, and T of Weeks 6 and 7.

Table 1
A broad overview of combinatorial content covered in the TE.

Targeted Concept	Representative Problem
Cartesian Product problems	*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?*
Arrangement with unrestricted repetition problems	*A license plate consists of six characters. How many license plates consist of three numbers (from the digits 0–9), 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?*
Problems that involve coordinating multiplication and addition in counting	*There are 3 French books, 4 Russian books, and 5 Spanish books. How many ways are there to bring two of these books on a trip, both of which are not in the same language? Can you create a computer program to list all of the possible book combinations for your trip?*
Arrangements without repetition problems (permutations)	*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?*
Selection without repetition problems (combinations)	*Suppose you have 8 books and you want to take three of them with you on vacation. Can you write code to list all of the ways you can choose your three books? What do you expect the structure of the list of outcomes to look like?*
Distribution problems	*Suppose you have three unique hats and five friends. How many ways are there to distribute the three hats among your friends, if no one can have more than one hat?*
More advanced encoding problems; Category II and arrangement with repetition	*I flip a coin 10 times in a row. How many different outcomes have exactly the same numbers of heads and tails?*

first engaged with and coded problems involving arrangement with unrestricted repetition, arrangements without repetition (permutations) and selection without repetition (combinations) problems. Then, they were given problems that required more advanced encoding such as distribution, Category II combination, and arrangement with restricted repetition problems. Table 2 shows the tasks we will discuss in the paper; Python code that lists and enumerates all encoded outcomes for each of these problems is provided in Appendix A.

3.3. Data analysis

All of the teaching episodes were transcribed, and the research team made enhanced transcripts in which we embedded relevant images into the transcripts, which facilitated subsequent analysis. For this paper, the first author re-watched all 11 episodes of Charlotte and Diana, reading alongside the enhanced transcript as she did so. She took note on the documents and particularly highlighted and commented on any episodes that involved the students' encoding outcomes. This initial round of analysis yielded a set of commented, enhanced transcripts that highlighted episodes in which the students engaged in encoding. In another pass, the first author then went back through all of the commented, enhanced transcripts and identified any episodes that were particularly salient or noteworthy and that could inform our understanding of students' encoding. She then copied into another document these relevant episodes of encoding that had been identified. In doing this, she was attentive to the types of problems the research team had designed for the study, focusing on problem types that we hypothesized would involve sophisticated encoding (discussed in Section 2.2).

The first author then revisited each of the episodes in that document and identified a certain incident that seemed to be a turning point on the students' encoding, which occurred on their work on the Unique Hats problem (a distribution problem, presented in Section 4.1.1). Realizing how impactful this episode was on their subsequent work, she reviewed this episode and looked across other episodes of their work on distribution problems, articulating the way in which the students built upon their previous work in each of the distribution problems. She then looked for additional problems to see if and how the students continued to build upon their work, and she repeatedly viewed those episodes, looking for encoding activity. Then, the two authors discussed these findings, and the second author reviewed the results and clarified and refined interpretations the first author had made. Ultimately, these episodes together helped the authors to construct a narrative (Auerbach & Silverstein, 2003) of the students' progress on encoding outcomes specifically. To account for the role of the computer, once the narrative was written, we again reviewed the key episodes by writing them up and attempting to explain ways in which programming and their work in the computational setting interacted with their encoding activity.

4. Results

We now present Charlotte and Diana's work on problems involving encoding outcomes. We spend the majority of the results on the students' work on distribution problems in Section 4.1, as they established an important mapping between prior work and distribution problems. Then, in Sections 4.2 and 4.3, we explore their work on a Category II problem and an arrangement with restricted repetition problem, respectively (Table 3 gives an outline of the section). Throughout, we highlight ways in which the computational setting seemed to enrich their encoding activity.

4.1. Students' initial encoding of outcomes on distribution problems

In this section, we highlight Charlotte and Diana's work on four successive problems that involved distribution problems, in particular distributing different kinds of objects to people (as outlined in Table 3). Together these episodes show the students'

Table 2
Tasks involving encoding that are discussed in this paper.

Problem name and statement	Solution
Unique Hats: Suppose you have 3 unique hats and 5 friends. How many ways are there to distribute the 3 hats among your friends, if no one can have more than one hat?	$P(5,3) = 5 \cdot 4 \cdot 3 = 60$
Lollipop: How many ways are there to give 3 identical lollipops to 8 different kids (if no kid can have more than 1 lollipop)?	$\binom{8}{3} = \frac{8 \cdot 7 \cdot 6}{3!} = 56$
Identical Lollipop Identical Balloon: How many ways are there to distribute 3 identical lollipops and 3 identical red balloons to 8 different kids (if no kid can have more than 1 object).	$\binom{8}{3} \cdot \binom{5}{3} = 560$
Unique Lollipop Identical Balloon: How many ways are there to distribute 3 unique lollipops and 3 identical red balloons to 8 different kids (if no kid can have more than 1 object).	$P(8,3) \cdot \binom{5}{3} = 3,360$
Heads/Tails: I flip a coin 10 times in a row. How many different outcomes have exactly the same numbers of heads and tails?	$\binom{10}{5} = 252$
BANANA: How many arrangements are there of the word BANANA?	$\binom{6}{3} \cdot \binom{3}{2} \cdot \binom{1}{1} = 60$

Table 3
An outline of problem types discussed in the Results section.

Section		Description
4.1	4.1.1	Distribution problems with distinct objects; students have a breakthrough in encoding
	4.1.2	Distribution problems with identical objects
	4.1.3	Distribution problems with multiple sets of identical objects
	4.1.4	Distribution problems with multiple sets of identical and distinct objects
4.2		Category II combination problems
4.3		Arrangement with restricted repetition problems

impressive trajectory toward fluently and flexibly engaging in encoding activity. We first note that the students had successfully been able to solve permutation and (Category I) combination problems, and they had also written computer code to solve these problems. When doing so, the students coded nested loops with certain conditional statements to generate lists of permutations and of combinations. As an example, Table 4 shows the students' interpretation of code for a permutation problem, and Table 5 shows the students' code for a Category I combination problem (more detail about this work is provided in Lockwood & De Chenne [2020]).

We include these tables to emphasize that prior to the work we describe in these results, Diana and Charlotte worked with programming problems involving permutations and combinations, and they came to understand and justify how they could use programs with nested loops and conditional statements to solve such problems. This is important because, in our results, we describe their connecting back to work on these types of problems.

4.1.1. An introduction to encoding – distributing distinct objects

In the fifth session of the TE, Charlotte and Diana worked on the Unique Hats problem: *Suppose you have three unique hats and five friends. How many ways are there to distribute the three hats among your friends, if no one can have more than one hat?* In solving this problem, Diana and Charlotte first confirmed that they were assigning three different hats to five different friends. The following exchange shows their initial discussion about this problem. Even though much of their work on this problem (and their breakthrough on the problem) exists in their by-hand work, in their initial consideration of the problem they were thinking about how to program it on the computer.

> *Diana*: I think if we were to like code this, would you like wanna do like outcomes that have like pairs? Like hats in this column [refers to a first column], and then friends in this column [refers to a second column]?
> *Charlotte*: Like, are you talking about like how it lists down here [the output window]?
> *Diana*: Yeah, like if we coded it and these would be our outcomes.

Charlotte noted that perhaps they would only need to consider the three people who have hats and assume that the two other people do not have hats. Diana agreed and suggested that the outcomes would be three hat-friend pairs, drawing three pairs of dashes and circling each pair.

> *Diana*: Okay, so each outcome, if we coded this, would look like – It would kind of be like three pairs, like one of those things represents a friend, and then the other space represents a hat.
> *Charlotte*: Mm-hmm.
> *Diana*: And because there's three hats, you have like three pairs.
> *Charlotte*: Right.

The interviewer (also the first author) asked Charlotte to reiterate Diana's idea, and Charlotte suggested that they could focus on the three people that get hats, saying, "And then you would just print three options, since there's like only three hats." Charlotte seemed concerned that she would need to figure out how to see which friends did not get hats, and she said, "So I'm trying to think of a way that you could actually see the other two friends without hats." We intervened with a pointed question asking whether they would

Table 4
Charlotte and Diana's understanding of code that solves a permutation problem, as shown in Lockwood & De Chenne (2020).

Provided task and program	*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, p1, p3, p4, p5)`
	`print(arrangements)`

Students' explanation	*Diana*: I think for sure that the statements have the exclamation point each time, that's making it so that these values will not repeat, which makes sense when you have five people because you can't just repeat a person. *Charlotte*: Yeah, that makes sense. Yeah, kind of what she was saying, I think the code, yeah, just trying to figure out how many different arrangements each person can be in and then yeah, each of these exclamation points, like Diana said, is to make sure John isn't sitting in two different seats at the same time.

Table 5
Charlotte and Diana's understanding of code that solves a Category I combination problem, as shown in Lockwood & De Chenne (2020).

Provided task and program	*Suppose you have 8 books and you want to take a pair of them with you on vacation. How many ways are there to do this?*
	`arrangements = 0`
	`Books = [1, 2, 3, 4, 5, 6, 7, 8]`
	`for i in Books:`
	`  for j in Books:`
	`  if j > i:`
	`  arrangements = arrangements+1`
	`  print(i,j)`
	`print(arrangements)`

Students' explanation	*Int. 1*: Okay. Cool. Great. Now let's try. What do you think that's doing? *Diana*: Yeah, it gets rid of the duplicates because it prevents it – j has to be bigger than i, if i is 1, then j can be anything above it, but then you can't have it be flip-flopped. *Int. 1*: Good. What do you mean flip-flopped? *Charlotte*: You can't have 2, 1; then j being 1 isn't bigger than 2.

need to see who did not get hats, and Diana said, "I don't think we do" and Charlotte said, "Not really." This was an important moment for the students to realize that it would suffice to produce outcomes that show the three people who are receiving hats. Diana noted in the excerpt below that they could focus on three rather than all five people. At this point, Diana was still focused on wanting to print pairs hats and people.

> *Diana*: Yeah. I think we just need to include the three friends that are wearing hats, and I'm wondering instead of printing three variables we might need to print six variables to account for which hat they're wearing, and so like they would be kind of in a pair. […] Do you think we need to do something like that to account for which hat they're wearing? If we have like a set of hats and a set of people?

We interpret that Diana's question about wanting separate sets of hats and people suggests that she was wondering about whether or not to define and use two sets within the code initially. They tried to write code that would produce pairs, as Diana described, and they ran into errors. We infer that they encountered limitations of their computational knowledge and expertise – as novice coders, they were (understandably) not well-versed enough in Python to display the pairs of outcomes that they wanted. We suggest that ultimately this limitation of the computer served to motivate and necessitate a certain kind of encoding outcomes that *could* be more easily inputted into the computer using their existing level of knowledge about Python. Even if the students were not aware of this, we interpret that this was a product of the computational setting.[6]

We then intervened and asked them to predict what the answer to the counting problem would be. This was an attempt to have them think about a possible mathematical expression, as they were stuck with the actual programming. The students discussed this and struggled to come up with a correct expression. They were still focused on pairs, and we infer that coordinating the pairs was

[6] We acknowledge that more experienced programmers might not face such limitations, but we focus on this case of Diana and Charlotte and explore how their encoding developed over the course of these tasks.

presenting a difficulty for them in figuring out a mathematical expression, due in part to their not settling on a way to encode their outcomes effectively. As the students reached an impasse, we intervened again, asking them to write down examples of what they were counting. In this prompt and the exchanges that follow, we were motivated to have the students realize that they did not need to write pairs, and information about pairs could be encoded into a single sequence of numbers by identifying what the positions (or columns of their output) represented.

First, we asked them to write down some examples of what outcomes would be. Diana gave the following response as she wrote outcomes in Fig. 1. She described different possibilities for which people would get Hats *a, b,* and *c*. We interpret that in Fig. 1, an outcome of *1a3b2c* represented that Friend 1 wears Hat *a*, Friend 3 wears Hat *b*, and Friend 2 wears Hat *c*, which aligns with what Diana said in the following exchange.

> *Diana*: You might have like, so you could do like Friend 1 and Hat *a* or whatever […] Friend 2 is wearing Hat *b*. Friend 3 is wearing Hat *c*. So like this is your first outcome. And then you might have Friend 1 wearing Hat *a*, Friend 2 wearing Hat *b*, Friend 4 wearing Hat *c*. And then like Friend 1 wears Hat *a*, Friend 2 wears Hat *b*, Friend 5 wears Hat *c*. And then if we continue this on – Let's see 1 wears Hat *a*, and then I guess it would go Friend 3 wears Hat *b*, and then Friend 2 wears Hat *c*. …

Motivated to have the students encode the information about the hats into the columns themselves, we then asked whether or not the locations of *a, b*, and *c* would ever change in her written pairs. This was a significant moment, because it underscored a key idea in encoding outcomes – that the column itself can represent something and encode information.

> *Int. 1*: […] I'm noticing *a* is always in that first column, *b* is always in the second column, *c* is always in the third column.
> *Diana*: Yeah.
> *Int. 1*: Would that change?
> *Diana*: I think you can go through all the options without changing those, because as long as you're changing the numbers, you're still reaching all the possibilities. And for me keeping *a* in the first column, *b* in the second column, and *c* in third column helps like keep it structured and organized. Charlotte, do you think that it could go through all the possibilities with keeping the letters in the same columns?
> *Charlotte*: I think so, yeah, because then you're just changing the people and just kind of keeping the hats on the same area. Yeah, I feel like that would go through all the options. So maybe –Maybe this is different, then. […]
> *Diana*: Oh, that's a good point.
> *Charlotte*: You know what I mean?
> *Diana*: Yeah, because if we're not like rearranging the letters, or the hats –
> *Int. 1*: I guess if the *a* and the *b* and the *c* can always be in those columns, do you need to write the *a*s, the *b*s, and the *c*s?
> *Diana*: I guess not.
> *Charlotte*: No. Not really –So you might as well make it simpler and just ignore kinda like the hats and just print F1, F2, F3. You know what I mean?
> *Diana*: Yeah.
> *Charlotte*: Because the hats aren't changing. The hats are just like staying in each column.

We acknowledge that this question was quite direct, but this was in line with teaching moves that might occur in a TE, and we are not claiming that the students made this connection spontaneously. Diana then took this information, and in the underlined portion we see Diana explicitly associating the positions ("spaces") with a particular hat, and she drew *a, b,* and *c* under each space when she said, "This is like, *a, b, c*."

> *Diana*: Yeah, so like – That means like each of these spaces kind of represents a hat, like when a friend is in a space it represents them wearing the hat. This is like *a, b, c*. And then if we just like rearrange the numbers so like there's five friends. So 5 times 4 times 3, because it's like 5 options for this one, and then you used up one of them so there's 4 options here. And then you used up those two, so –
> *Charlotte*: Yeah, that makes way more sense.

Then, having made that association, Diana went on to articulate a (correct) solution to the problem. Now that she was considering numbers of options for spaces, we infer that she was in familiar territory and could apply prior experience and knowledge to solve the problem. In particular, she recognized that she had 5, then 4, then 3 options for what friends could go in each position (seen in Fig. 2), and Charlotte agreed with this solution. We highlight how important this shift in perspective was in helping them arrive at a succinct and correct answer. In contrast to their previous incorrect work, the students could relate this to a simple expression that they had encountered previously. The students made an important shift from needing to write pairs to encode information to writing three numbers that encoded that same information.

We then asked the students to reflect on their experience with this problem. Although they had not written a program for the problem yet, Diana anticipated that they would be able to reduce the complexity of their code because they did not have to consider a variable of hats in the code. Importantly, their code then resembled code they had previously used to solve permutation problems (such as the program shown in Table 4).

> *Charlotte*: I mean, like Diana said, kind of, just like each space represents a hat, so like if you don't change the hats and just change the people, then you can get all the different possibilities. And yeah, I mean that definitely helped with drawing it out.
> *Diana*: Yeah, because like once you see that the letters are not changing, then you kind of know that that doesn't have to be a variable in your code, because like it's constant, so it doesn't need to vary.

We asked the students why they would never need to write an outcome like 5*b*3*a*1*c*, and Diana noted that, "Because once you go

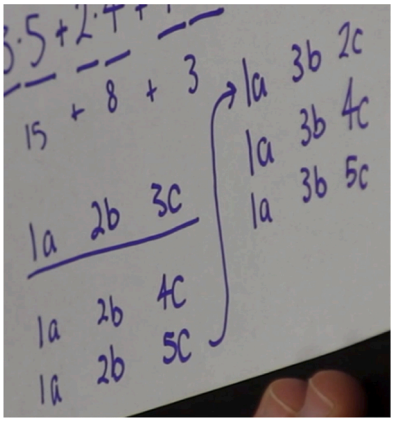


Fig. 1. Diana's listing of several initial friend-hat pairs.

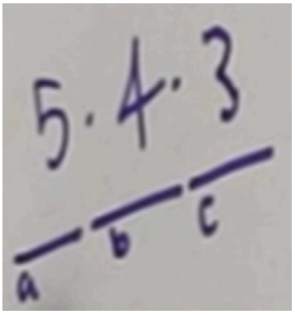


Fig. 2. Diana encoded spaces as different hats, and she proposed a solution of 5·4·3.

through like all of the options that have 1*a* first, like it moves to like 2*a* and then all of those options, and eventually you'll get to like… like 1 will be in the *c* column, so that counts for like 1 being paired with *c*, and like 5 will be in the *a* column, so like that accounts for 5 being paired with A." Charlotte also said, "Yeah…if you have … the first person wearing the *c* hat, second person wearing the *b* hat, and the third person wearing the *a* hat, that's the same as, you know, the third person wearing the *a* hat, like that way. It's gonna be the same." These responses suggest that both Charlotte and Diana could justify their answer.

This episode was very important for the students' understanding of how to encode information that, while possible to express as pairs of friends and hats, could be expressed more succinctly as sequences of numbers. Although the students were working within a computational environment, and, we would argue, were motivated by the computational setting, their key insight did not directly involve the computer. It came about by thinking about outcomes, and engaging in by-hand listing was effective in their recognition of this encoding technique. However, even if this important insight involved by-hand listing, their ultimate task was to code their solution. Given their new insight, we proceeded with having them trying to program the problem. As they began, Charlotte noted, "You just kind of treat each slot as a hat." The students then wrote the code in Fig. 3a, and Diana made the following prediction about the first outcomes she would expect. This code is similar to code Charlotte and Diana wrote and used for previous problems that asked for the number of ways to permute a set of objects.

```
1
2   arrangements = 0
3   Friends = ['1', '2', '3', '4', '5']
4   for f1 in Friends:
5       for f2 in Friends:
6           if f2 != f1:
7               for f3 in Friends:
8                   if f3 != f1 and f3 != f2:
9                       arrangements = arrangements + 1
10                      print(f1,f2,f3)
11  print(arrangements)
12
```

```
5 2 1
5 2 3
5 2 4
5 3 1
5 3 2
5 3 4
5 4 1
5 4 2
5 4 3
60
```

Fig. 3. a,b: The students' code and the last nine outcomes for the Unique Hats problem.

Diana: I think our total is gonna be 60 because of the 5 times 4 times 3, and then I think it'll be organized just like this, like these are the first six terms, except without the letters. It'll just have the numbers like 123, 124, 125.

The students then ran the code in Fig. 3a, and Fig. 3b shows part of the output. We asked them to interpret, in terms of hats and people, an outcome of 152. Their responses below underscore their understanding of the problem, confirming that they thought of the positions of the numbers as representing hats *a, b,* and *c*. Diana further explained why 125 would be distinct from 152 in this problem, as it would represent different people getting different hats.

Charlotte: So, I feel like 152, it's just showing you that the first person is wearing we'll just say Hat *a*. The fifth person is wearing Hat *b*, and the second person is wearing Hat *c*.
Diana: And that's like different than, for example, 125, where the first person wears Hat *a*, the second person wears Hat *b*, and the fifth person wears Hat *c*.

In a final bit of discussion and reflection, we asked the students why it was not necessary to encode outcomes as pairs. Diana's response below reiterates what we contend is the fundamental aspect of understanding the encoding that they gained from this problem – that the columns (and positions) themselves can encode information.

Diana: Yeah. It's kind of like each column, like, or each – Like each variable and each outcome, like it is a pair, like it's paired to a column. One is paired to column one, and this one, two, is paired to the second column, and three is paired is paired to the third column. […]
Int. 1: Okay, great, and in this case, what do those columns mean?
Diana: The different hats, the types.

To summarize, in this episode we saw a very important shift in the students' reasoning about outcomes, which was guided by the interviewer's prompts. Diana and Charlotte realized that columns in an output can contain information and can implicitly serve to encode information of pairs of objects and not just a single object. Because they understood that they could encode information as single pairs (and decide what the columns would represent – in this case different hats), they viewed a distribution of distinct objects as a familiar problem of arranging distinct objects. They then used code they had seen previously to count distributions of distinct objects.

We address this further in the Discussion section, but we contend that the multiple representations afforded by the by-hand work and the programming activity contributed to the students' successful work on their encoding activity in this problem. The key insight came about during by-hand work, and these ideas could likely be elicited completely without the computer. However, being asked to program a solution did seem to have an effect on their motivation to find another way to encode outcomes, particularly when they ran into programming limitations.

4.1.2. Distributing identical objects

We now briefly describe the students' work on a problem involving distribution of identical objects. We highlight how they drew on the previous distribution problem to approach and solve this new problem. In Session 7, they were working on the Identical Lollipops problem: *How many ways are there to give 3 identical lollipops to 8 different kids (if no kid can have more than one lollipop)?* Immediately Diana asked whether they should "treat the lollipops as like spots? And then like the kids like cycle through spots?" This suggests that their realization that they could encode positions (or "spots") as containing information persisted from the Unique Hats problem to this problem. We asked Diana why she thought of spots and why it might be helpful, and she reiterated a connection to prior activity.

Diana: I thought it would because there was a problem we did before, that was like assigning something to a group of people. I can't remember if it was like books, or something. But, like, we like thought at first, like we would have pair them up, but then like when we were doing the code, it was like each spot like represented a pair of things. So, in order to get through like all the combinations of kids, like paired with lollipops, you don't need to, like, write in a pair if you're doing your code, you just need to have it as a spot.

The students guessed 8·7·6 as the answer [Fig. 4], which makes sense because that is what they would have gotten by extending their work on the previous Unique Hats problem. They started to discuss what outputs they would expect by adopting this approach. Even in their initial discussion below, they carried over ideas from the Unique Hats problem, and they identified columns and positions as representing distinct Lollipops *a, b,* and *c*.

Charlotte: Okay. Well, we could label this as like Lollipop *a*, Lollipop *b*, and Lollipop *c*. And then just like pair them. […]
Diana: So, like the first order would be like 123. And that means that like kid number 1 has *a,* kid number 2 has *b*, and kid number 3 has *c*. And then, like, it would go to that, so kid number 1 still has one – or A. Kid number 2 still has *b*, and kid number 4 has *c*.

The students then discussed whether they wanted the Lollipops to be different. Charlotte nicely described what it would mean to have the positions represent distinct lollipops.

Diana: Oh, yeah. Maybe we do want to repeat that say 3, 2, 1, don't we?
Charlotte: Yeah. So – Yeah, because each slot is something different. So, like what we did before was like with the books, you know? Like taking book *a, b,* and *c* is the same as taking books *c, b, a*. But this – each like slot is something completely different. So, like, having 123 is different than 321 because, you know … Person 1 has Lollipop *a*, and 2 has *b*, 3 has *c*. But then if you switched it to 321, then the third person has *a*. The second person still has *b*. And the first person has *c*.

This is a correct interpretation of what a solution of 8·7·6 would imply, but we clarified that in this problem the lollipops are identical. After they realized the lollipops should be identical, they discussed what they should do and talked about writing some code.

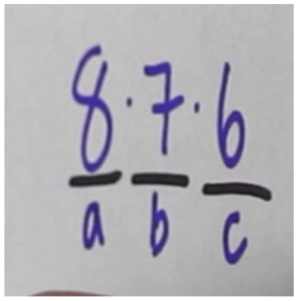


Fig. 4. a The students' initial solution to the Identical Lollipops problem.

When they realized they did not want to consider both 123 and 321 as distinct, Charlotte inserted a conditional statement of "if j > i" into the code. This was part of their scheme for solving selection without repetition problems[7], which we have discussed elsewhere (Lockwood & De Chenne, 2020). They asserted that using the conditional greater than statement in their code would give their desired total of $\frac{8\cdot7\cdot6}{6} = 56$.[8] Charlotte correctly predicted the output of the code (Fig. 5b) they had written (Fig. 5a).

In a final reflection on this problem, we again asked them to interpret what a particular increasing sequence would mean in terms of the problem. The students demonstrated that they understood that a sequence of numbers could be thought of as child-lollipop pairs.

> *Int. 1*: And so, in terms of that, like whether or not you have a lollipop, what does, say, a three letters sequ – like what does 2, 4, 5 mean?
> *Charlotte*: That just stands for like Child 2 has a lollipop, Child 4 has a lollipop, and Child 5 has a lollipop.
> *Int. 1*: Okay. Awesome. Yeah, is that how you're interpreting too, Diana?
> *Diana*: Yeah.

This episode shows another instance in which the students could think about counting the number of ways to distribute items as being equivalent to counting the number of ways to count sequences of numbers. Here, they refined their solution to the Unique Hats problem to count distributions of identical items, relating that to counting strictly increasing sequences of numbers.

4.1.3. Distributing multiple sets of identical objects

We now show how the students solved a problem that involved distributing two kinds of identical objects simultaneously, the Identical Lollipops Identical Balloons problem: *How many ways can you distribute 3 identical lollipops and 3 identical red balloons to 8 kids (no kid can have more than one object)?* The students faced some challenges initially reasoning about this problem, but they persisted, demonstrating very sophisticated reasoning about how to encode outcomes in a meaningful way. We include this problem because it highlights an additional instance of their encoding, and it demonstrates how the programming activity interacted with their combinatorial thinking and activity.

The students' first approach was to engage in similar encoding as they had on the previous Identical Lollipops problem. They encoded sequences of six numbers, where the first three positions represented which kids had a lollipop, and the second three positions represented which kids had a balloon. We then asked them to write a program to enumerate the outcomes, and, following what they had done on the Identical Lollipops problem, they coded six nested for-loops, all with the conditional statements involving greater than (Fig. 7). In the exchange below, they again clarified how they were encoding these outcomes, with the first three columns representing lollipops and the last three columns representing balloons. This code counts strictly increasing sequences of length six from the numbers 1 through 8, and this was similar to work they had done previously in the TE (such as in Table 5).

> *Charlotte:* So, a possible outcome could just be 1, 2, 3, 4, 5, 6, and that represents kids 1, 2, and 3 each having a lollipop, and kids 4, 5, and 6 each having a red balloon. That's a possible outcome.
> *Diana:* Yeah. And then, like, I think, from that outcome, like the one through six, it would go to like 1, 2, 3, 4, 5, 7, because that's like the next one.
> *Int. 1:* And so, again, what does it say, like 1, 2, 4, 5, 6, 7 represent?
> *Diana:* It's like Kid number 1 has a lollipop, Kid number 2 has a lollipop, Kid number 3 has a lollipop, Kid 4 has a balloon, Kid 5 has a balloon, and Kid 7 has a balloon.

This initial approach makes sense, particularly given their prior work, and notably they assigned meaning to each of the columns [Fig. 6]. However, the code in Fig. 7 is not correct, because it would not allow kids 1, 2, or 3 to get lollipops (that is, the numbers 1, 2, and 3 could not appear in the last 3 columns). We drew their attention to this issue, and the students considered how to proceed. They wondered if they might multiply the total by two, but they were not convinced of that approach (and this would not be correct). They seemed unsure of how to proceed, and so we explicitly prompted them to think about a two-stage process that would first distribute

[7] More specifically, by including the conditional statement of "if j > i," Charlotte ensured not only that the elements would not be repeated, so something like 111 would not appear, but also the elements would only appear in strictly increasing order, so 123 and 321 would not both appear.
[8] Details of the expression here are not important; they knew to divide to account for duplicate outcomes.

```
# Task 3a
# How many ways are there to give 3 identical lollipops to 8 different kids
# (if no kid can have more than 1 lollipop)?
arrangements = 0
kids = [1, 2, 3, 4, 5, 6, 7, 8]

for i in kids:
    for j in kids:
        if j > i:
            for k in kids:
                if k > j:
                    arrangements = arrangements+1
                    print(i,j,k)
print(arrangements)
```

```
3 4 5
3 4 6
3 4 7
3 4 8
3 5 6
3 5 7
3 5 8
3 6 7
3 6 8
3 7 8
4 5 6
4 5 7
4 5 8
4 6 7
4 6 8
4 7 8
5 6 7
5 6 8
5 7 8
6 7 8
56
```

Fig. 5. a, b. The students' code and some of the output for the Identical Lollipops problem.

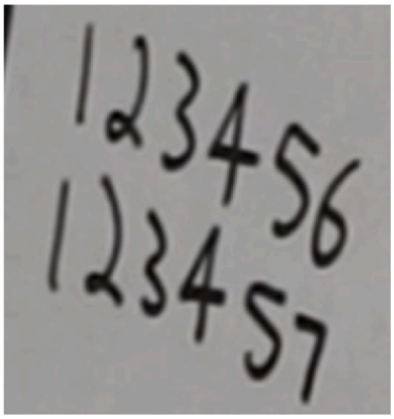


Fig. 6. Diana wrote two outcomes and interpreted them in terms of the problem.

```
# Task 3b)
# How many ways are there to distribute 3 identical lollipops and 3 identical red balloons
# to 8 different kids (if no kid can have more than 1 object).
arrangements = 0
kids = [1, 2, 3, 4, 5, 6, 7, 8]

for i in kids:
    for j in kids:
        if j > i:
            for k in kids:
                if k > j:
                    for l in kids:
                        if l > k:
                            for m in kids:
                                if m > l:
                                    for n in kids:
                                        if n > m:
                                            arrangements = arrangements+1
                                            print(i,j,k,l,m,n)
print(arrangements)
```

Fig. 7. The students' initial coding of the Identical Lollipops and Identical Balloons problem.

lollipops, then distribute balloons. We reminded them of their solution to the Identical Lollipops problem (giving 3 identical lollipops to 8 kids, which they had found to $\frac{8 \cdot 7 \cdot 6}{6} = 56$).

> *Int. 1*: … The 8 times 7 times 6, over 6. That 56, that gives you the number of ways to give three identical lollipops to eight kids, right?
> *Charlotte*: Mm-hmm.

Diana: Yeah.
Int. 1: So, then, I guess, what if you gave the identical lollipops, and then sort of once you gave out the lollipops, you gave out the red balloons? So, like thought of it as sort of a second stage in your process. So, like first give out the lollipops, then give out the balloons.
Diana: So, like giving out the balloons, then you'd have five options of kids? Because like three of them already have objects. So, then that kinda turns into – […] It would be three spots – I'm sorry. And each of those spots represents having a red balloon, and then it would be 5 times 4 times 3, divided by the number of duplicates, which is 6 [writes the expression on the left in Fig. 8]. Because there's three different things being doled out. So –

Charlotte then wrote down the expression $\frac{8\cdot7\cdot6}{6}$ (see Fig. 8). They were unsure of whether they should add or multiply the two expressions in Fig. 8. We then intervened again and asked them to write some code to help them determine whether they would want to add or multiply.

Diana: Yeah. So, I think this will be like our second chunk of three columns [points to $\frac{5\cdot4\cdot3}{6}$] and this is the first chunk of three columns [points to $\frac{8\cdot7\cdot6}{6}$]. So, like it'd be helpful to figure out a way to put this in the second chunk without affecting what's going on over there.

Diana then began to code the problem on the computer, and the students discussed what they should do. In this conversation, Diana observed that once three lollipops were distributed, there were then only five, four, and three options for which kids could get balloons. In trying to defend her reasoning, Diana referred back to the previous problem of giving lollipops to three kids. She re-ran the code from the Identical Lollipop problem, which helped to convinced the students that the $\frac{5\cdot4\cdot3}{6}$ was giving them what they wanted. Here they had a fruitful conversation about whether or not the expression of $\frac{5\cdot4\cdot3}{6}$ was correct, and they looked back at all 56 arrangements of assigning 3 lollipops to 3 kids to convince themselves that in fact, as Diana said, "when kids 5, 7, and 8 have lollipops…then kids 1, 2, and 3 can have balloons." We highlight this as an important example of the students using the computer for insight about encoding – they ran a previous program, examined the outcomes, and decided that it indeed gave them what they intended.

Diana: Yeah. So, like this expression, which is the lollipops, that like – I mean, I'm gonna go back to our Task 3a really fast. That does account for like – wait what was –? […] [Runs the program] So, this accounts for like all the different kids like having lollipops, so even kids 6, 7, and 8 get a lollipop. And because that goes through all of them, like this comes on a case-by-case basis, so yes, when kids 1, 2, 3 have lollipops, only the kids that aren't 1, 2, 3, that are like 4 through 8 can have balloons.
Charlotte: Right.
Diana: But then when kids 5, 7, and 8 have lollipops, for example, then kids 1, 2, and 3 can have balloons. And such.
Charlotte: Okay.
Diana: Does that make sense?
Charlotte: Yeah.

To explore the idea they had just articulated, we asked them to write down and interpret an outcome. Diana wrote 567123 and said, "that means that kid 5, 6, and 7, they each have a lollipop, and then kid 1, 2, and 3 each have a balloon," showing her understanding of the encoded outcome. The students continued to discuss whether to multiply or add. To argue for multiplication, Diana wrote a second outcome they would want to count, 578134. This convinced herself and Charlotte that they should multiply the number of ways of distributing lollipops by the number of ways to distribute balloons, as 578 could be paired with three of 1, 2, 3, 4, and 6.

Ultimately, the students had a necessary key insight in order to program this problem correctly. Because a kid could only receive at most one object, they realized that they needed to ensure that the options for who got a balloon were distinct from the options for who got a lollipop. To account for this, they incorporated != conditions in their last three loops (Fig. 9).

Diana nicely summarized their approach and what they thought their code would do. They correctly predicted the number of outcomes of the code, which matched their answer.

Diana: … We first used the 'greater than' expression up here because there weren't any like other constraints. Like those are just the lollipops, and we're distributing those first. But then when it gets to the balloons, there are more constraints because you already used three kids, like that have the lollipops. So, it's signifying the 5 times 4 times 3, divided by 6 because we're getting

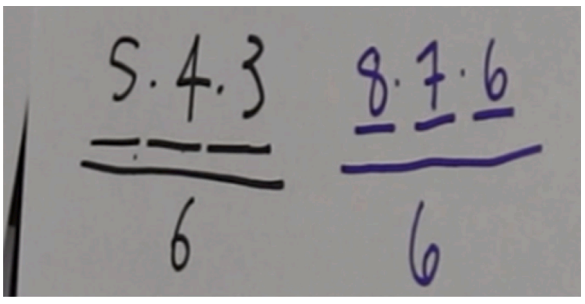


Fig. 8. Diana's expression for giving out balloons to three kids (on the left) and Charlotte's expression for distributing identical lollipops to eight kids (on the right).

```
# Task 3b)
# How many ways are there to distribute 3 identical lollipops and 3 identical red balloons
# to 8 different kids (if no kid can have more than 1 object).
arrangements = 0
kids = [1, 2, 3, 4, 5, 6, 7, 8]

for i in kids:
    for j in kids:
        if j > i:
            for k in kids:
                if k > j:
                    for l in kids:
                        if l != k and l != j and l != i:
                            for m in kids:
                                if m > l and m != i and m != j and m != k:
                                    for n in kids:
                                        if n > m and n != i and n != j and n != k:
                                            arrangements = arrangements+1
                                            print(i,j,k,l,m,n)

print(arrangements)
```

Fig. 9. Charlotte and Diana's code for the Identical Lollipops Identical Balloons problem.

rid of duplicates by like – We're still using the 'greater than' expressions for just the balloon variables, and we're using the 'not equal to' expressions to make them not equal the kids that already have lollipops.

The interviewer asked the students then to look at the output of the computer code to see if it gave them the outcomes they expected. We highlight a couple of observations they made that demonstrate how they were understanding their encoding of outcomes. First, Charlotte noted that she was paying attention to the 1 s, looking to see when 1 s would appear in the balloons group. She and Diana agreed that 1 would show up in the balloons (the last three columns) when Kid 1 was no longer getting a lollipop (1 was not in the first three columns). Then, Charlotte asserted that 234156 was distinct from 123456, saying, "Because the first three columns are lollipops and the second three columns are balloons. So, like the first kid having a lollipop is different than the first kid having a balloon." These exchanges showed that they understood what their code was doing in terms of what outcomes it would produce.

To conclude our discussion of this problem, we briefly share students' insights about their use of code and what they realized about encoding in this problem. We asked if the computer was helpful and if they felt more confident in their answer. In Charlotte's response below, we interpret that she stated that the code helped to visualize the different sections of the outcomes, and she highlighted (in Fig. 10) specific elements of the code that were related to the outcomes she encoded (such as the fact that one of the variables could not equal the first three variables).

Charlotte: Okay. I definitely think that – I mean, I enjoy like just working out the math, but I think that like the code definitely helped to like visualize. Especially this part of – You know, okay, n has to be greater than m, but it also like can't equal the first three variables. So, I think, that kinda helped visualize it, and it kind of helps account for like the two different like three sections. […]

Diana noted how the code seemed to help her realize why multiplication would make sense.

Diana: Yeah. I think on this one, the code did definitely help. After we like realized that these were two different outcomes and you could have the same three kids having lollipops as long as you rearrange the kids that have balloons. That made me realize

```
# Task 3b)
# How many ways are there to distribute 3 identical lollipops and 3 identical red balloons
# to 8 different kids (if no kid can have more than 1 object).
arrangements = 0
kids = [1, 2, 3, 4, 5, 6, 7, 8]

for i in kids:
    for j in kids:
        if j > i:
            for k in kids:
                if k > j:
                    for l in kids:
                        if l != k and l != j and l != i:
                            for m in kids:
                                if m > l and m != i and m != j and m != k:
                                    for n in kids:
                                        if n > m and n != i and n != j and n != k:
                                            arrangements = arrangements+1
                                            print(i,j,k,l,m,n)

print(arrangements)
```

Fig. 10. Charlotte highlighted part of the code that was particularly useful for her.

that, "Okay, yeah. We want it to like – the for-loops to all be indented, and being like a multiplication sign between these three different functions, instead of like trying to add it." Because I knew – it definitely was included in that. You have to have six things in a line, and if you tried to add these different things separately, the only way we know how to do that, is like making a chunk of three on top, and then a chunk of three columns on the bottom [gestures to the screen].

Her response suggests how encoding outcomes contributed to this insight. She realized that in using multiplication (involving six nested for-loops) she would get "six things in a line" (a six-tuple) rather than two three-tuples. Because she had a strong sense of what the outcomes should be (six -character sequences with certain properties), the programming helped to solidify that she should multiply, as six nested loops gave the kinds of outcomes she expected, whereas two sets of three nested loops would not. We interpret that the structure of the code and the outcomes it would produce contributed to her reasoning about which operation made sense in this situation.

4.1.4. *Distributing a set of unique objects and a set of identical objects*

We conclude this section on distribution problems by briefly presenting their work on the Unique Lollipops Identical Balloons problem: *How many ways are there to distribute 3 unique lollipops and 3 identical red balloons to 8 different kids (if no kid can have more than 1 object)?* Due to space limitations, we do not go into detail, but we include it here for completeness, showing their work on a distribution problem that involved coordinating distributing both identical and distinct items. The students were able to adjust their code (Fig. 11) to answer this problem correctly, explaining their work below.

Charlotte: So, you kinda just changed – Instead of j having to be greater than i, we just changed it to j not equal to i. Because the j being greater than i was accounting for duplicates, and so since we're allowed to have duplicates this time, then we just wanna make sure that you don't get 1, 1, 1. That's what the j not equal to i – […]
Int. 1: And I know you said this before, but by duplicates, what do you mean?
Charlotte: So, with duplicates it will be like, 1, 2, 3 and 3, 2, 1.

The students demonstrated that they understood what their code would do and how it encoded outcomes. The interviewer asked them what they expected to see as output of the code in Fig. 11. Diana said, "It'll kinda be similar to this, except there'll be more arrangements of the first three columns. So, like it'll probably start out with 1, 2, 3, and then kinda like it did up here." She noted that they would expect 1, 3, 2, and Charlotte noted that they would expect 1, 3, 2 after 1, 2, 8 appeared. The students ran the code and were correct in their prediction. The point is to show that the students could make sense of and use this code effectively. They wrote the code to print six-character sequences with particular properties, and they were able to understand and articulate what those properties were doing.

4.2. *"Category II" combination problems*

We had posited that both their excellent work on the distribution problems and the relatively difficult nature of encoding such Category II problems (Lockwood et al., 2018) would provide a rich opportunity to observe the students' encoding activity. After the students' work on the distribution problems, we gave them several Category II problems in Session 9 to see if they could encode the outcomes in a way that was conducive to counting. Our purpose here is to demonstrate students' correct encoding activity on a problem type that has been shown to be difficult for students to solve and to encode effectively (e.g., Lockwood et al., 2015; Lockwood et al., 2018), and we emphasize that they built on their prior work on distribution problems. Because of space limitations, we describe the students' work on just one Category II problems (the Heads/Tails problem: *I flip a coin 10 times in a row. How many different outcomes have exactly the same numbers of heads and tails?*) The students correctly encoded and solved this problem relatively quickly, and our point is to demonstrate their successful encoding activity.

```
# Task 3c)
# How many ways are there to distribute 3 unique lollipops and 3 identical red balloons to
# 8 different kids (if no kid can have more than 1 object).
arrangements = 0
kids = [1, 2, 3, 4, 5, 6, 7, 8]

for i in kids:
    for j in kids:
        if j != i:
            for k in kids:
                if k != j and k != i:
                    for l in kids:
                        if l != k and l != j and l != i:
                            for m in kids:
                                if m > l and m != i and m != j and m != k:
                                    for n in kids:
                                        if n > m and n != i and n != j and n != k:
                                            arrangements = arrangements+1
                                            print(i,j,k,l,m,n)

print(arrangements)
```

Fig. 11. The students' code for the Unique Lollipops Identical Balloons problem.

In discussing their approach, Charlotte suggested initializing a set with digits 1 through 10 and then just printing five letters. She said, "kind of thinking about the code, just explaining you would have like digits 1 through 10, then you would have a print statement with five letters," which we interpret a similar approach to what they had done on the distribution problems. Diana suggested introducing some additional information and seemed to be viewing as each number representing a head or a tail, and not the positions. She said, "We could have a like element set that has 1 through 10; and we could say that 1 through 5 represents heads, and 6 through 10 represents tails." In the following excerpt, Charlotte acknowledged this response from Diana, but she articulated more clearly that she meant the numbers would represent *positions* and not the actual heads and tails (the underlined portion). Diana seemed to agree with her line of reasoning.

> *Charlotte*: You can do that but it's just gonna make it more complicated. It might just be easier to just say one through – because you wouldn't want one through five only to be heads, and six through ten to only be tails because like here the heads is in position seven. So if you only restricted it to one through five, then these are the only slots you could put heads, you know?
> *Diana*: Well, it could put a two over here; we would just need to like code it correctly to make it do that. We might need to use like not equal to statements, and we might need to use letters, I guess.
> *Charlotte*: I just think it might be easier just to keep it kind of like this just to kind of isolate one variable because you automatically know since it is evenly split half-and-half, like if you have heads in positions you know one, two, four and six then you obviously have tails in positions three, five, seven, eight, nine, ten.
> *Diana*: Oh, I see your line of thinking now!
> *Charlotte*: Do you know what I'm talking about?
> *Diana*: Yeah. Okay. I get now why we could get away with only having five variables in our print statement.
> *Charlotte*: Right.
> *Int. 1*: Okay, and why is that? What are those five variables representing, Diana? Or, sorry, Charlotte, because you were just saying you got –
> *Diana*: Yeah, so they would represent heads because like in this problem, and even the last problems when we didn't really care about it, the numbers that aren't listed there and aren't printed in any specific outcome are the positions of the other like options. So like in this case it would be tails.

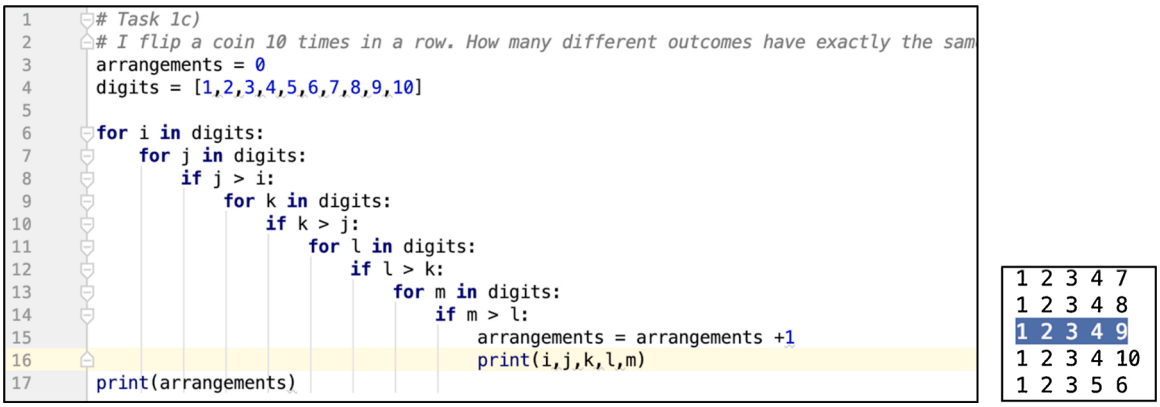
The students ran the code in Fig. 12a, and Charlotte highlighted the outcome in Fig. 12b. She confirmed what she had said about the sequence of numbers representing positions:

> Charlotte: Right, because I mean this is still gonna print out five. So like, even if you just had this one, you're like okay, well, you could say either heads or tails. So yeah, the heads would be in positions 1, 2, 3, 4, and 9; and then tails would be in positions, 5, 6, 7, 8, and 10.

After they had run the program, we asked Diana to explain her process on this problem. We see that the idea of encoding positions (and then ultimately writing code that would produce sequences/sets of numbers with certain properties) remained important for her.

> *Diana*: … In terms of the code, like I didn't really think back to the other problems yet; I was like treating it differently in my head and thinking that we should have ten variables in our print statement, and then like the ten variables in our outcomes. But then, like, when Charlotte suggested the five variable and she explained it a bit more then I understood that like, oh yeah, we like, we kind of know where the other, like, where the tails coin positions are just based on knowing where the heads coin positions are.

This last sentence is particularly noteworthy, given how important it is for students to be able to formulate and use isomorphisms among problems, and, more specifically, to leverage the fact that they can encode certain outcomes as sets of distinct positions. Further, this quote by Diana suggests how the act of programming played into her reasoning. We contend that her consideration of the five versus ten variables in the print statement is particularly important. The students seemed to understand that number of variables in their print statement would correspond to the number of variables in their outcomes (they sometimes referred to these as columns).



```
# Task 1c)
# I flip a coin 10 times in a row. How many different outcomes have exactly the sam
arrangements = 0
digits = [1,2,3,4,5,6,7,8,9,10]

for i in digits:
    for j in digits:
        if j > i:
            for k in digits:
                if k > j:
                    for l in digits:
                        if l > k:
                            for m in digits:
                                if m > l:
                                    arrangements = arrangements +1
                                    print(i,j,k,l,m)
print(arrangements)
```

```
1 2 3 4 7
1 2 3 4 8
1 2 3 4 9
1 2 3 4 10
1 2 3 5 6
```

Fig. 12. a, b: Charlotte's code and a highlighted outcome for the Heads/Tails problem.

This relationship between print variables and the outcomes underscores something important about the nature of the outcomes. The fact that Charlotte suggested that they just needed five variables made Diana think carefully about what information five (as opposed to ten) variables might give them. Ultimately, Diana agreed that five was enough because she could infer what the remaining variables would be, and, thus, a ten-element string could be encoded as a set of five numbers. This episode shows that the students' ability to encode the outcomes contributed to their success on this Category II problem, a problem type that has been shown to be problematic for students.

4.3. Arrangement with repetition problems

As a final example of Charlotte and Diana's work on problems involving encoding, we present their work on the BANANA problem: *How many arrangements are there of the word BANANA?* Our point in presenting this problem is to see how the students engaged in encoding, particularly as they attempted to program this problem. As we will see, a clever encoding of outcomes was not necessary for them to solve the problem correctly, but it was necessary for them to be able to program the problem. This example thus highlights potential ways in which the activity of programming may motivate and elicit sophisticated encoding for students.

In approaching this problem, Diana almost immediately had an insight about a solution strategy, which she implemented and justified correctly, suggesting the expression in Fig. 13a. She was initially unsure about whether to add or multiply the terms in the denominator, but, ultimately, she argued for multiplication. She explained her work in the excerpt below. We do not focus too much on Diana's solution except to note that it was insightful and reflects an equivalence approach to such problems (Section 2.2.3, CadwalladerOlsker, 2013).

> *Diana*: I thought about how, like I knew there were multiple letters of the same letter. And I was thinking about how like if we arranged the word, like, I guess I'll just spell it out like it's written, BANANA and BANANA. Those are like technically two different options in terms of the N, like this could be N1, N2; this could be N2 and N1. But we don't want to count those as distinct [writes work in Fig. 15b]. So, if you're talking about the Ns since there are two of them you divide out that. Because I was thinking about the way you arrange like the multiple letters. And now as I write this out, I think it does make sense to multiply because you have the N switching going on at the same time as you have the As that could be rearranging themselves. So, the multiplier makes sense, because like it's happening at the same time.

Again, this is an excellent solution that Diana reached quickly, and we do not mean to detract from this correct answer. But, for the purposes of this paper, we focus on the students' work in programming the problem, as this demonstrates their encoding activity via the computer. Here, we see some payoff of their hard work on the distribution problems. Right away, Charlotte made a connection to prior problems and talked about programming positions as they had previously.

> Charlotte: I feel like you might be able to make it easy where so if you have – So if you just like ignored the letter B for right now; you're left with five letters. And you could do kind of like what we've done here, and then just assume that the first three columns are the As and the last two columns are the Ns. So then you know the positions of all the letters. So then B, is just in the last spot; and that kind of accounts for yeah, the As obviously being different, and the Ns being different.
> *Int. 1*: Good. […] So you were maybe saying using kind of a similar idea to what you had done last time. So say a little more about that, and then, yeah does what she say make sense Diana?
> *Diana*: Yeah, it does make a bit of sense; but I'm also confused about like the B being on like either end, because it could also be in the middle, but then does that, does your idea for code, does that account for the B being in the middle, maybe in the third spot?
> *Charlotte*: Well, yeah, because it would just, if you had, well I guess how many numbers would you have? There are only six. Hold on. Let's just say 1, 2, 3, 4, 6 [highlights that outcome]. So the three As are in positions 1, 2 and 3. And the Ns are in positions 5 and 6; so then B is in position 4.

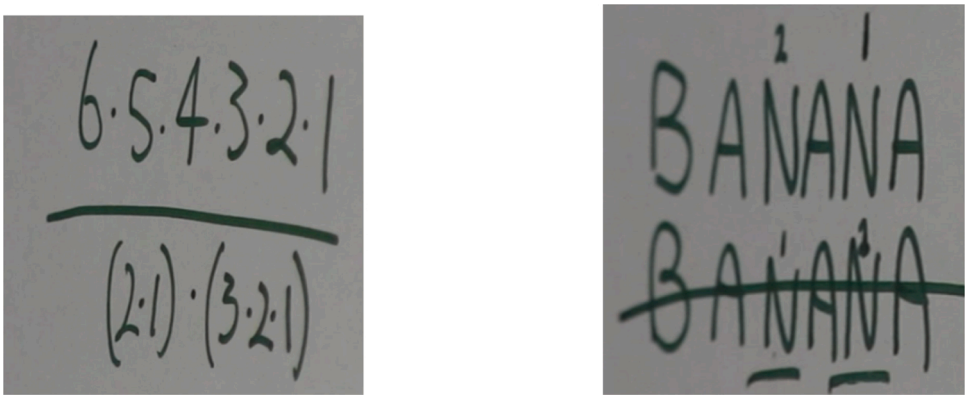


Fig. 13. a, b. Diana's initial idea for the BANANA problem.

Diana: Okay that kind of makes sense.

The students continued their conversation as they began to write their program. Charlotte explained the program and clarified that the numbers did not represent the letters, but rather that they represented the positions for the respective letters.

> *Diana*: And then what does – are you saying that like the six represents the B; and then like four and five represent?
> *Charlotte*: No. So, once you get the actual print statement, these first three columns represent the As; these second columns represent the Ns. So like, for this [1 2 3 4 5], right, the three As would be in positions 1, 2, 3; the Ns would be in positions 4 and 5; and the B would be in position 6.

Charlotte completed the program in Fig. 14, and the first three columns represented As, the fourth and fifth represent Ns. She ran the code and got 6 outcomes, which she recognized was incorrect. The issue, which Diana explained below, is that by using only greater than signs (as they had on the Heads/Tails problem), they would never see a 5 or a 6 in any of the first three columns (which represent A), which meant that they would never have an A in position 5 or 6.

> *Diana*: Like I think for this – this set-up, anyway like if the first three columns are the As you're never gonna have an A that's in position five or six, because the code is like making it so the first columns can't be the bigger numbers.
> *Charlotte*: Right, so you would – you wouldn't wanna use the greater than you would wanna use the not equal to sign.

Importantly, here again they were referring to the columns and positions, which suggests that they were articulating a relationship between how their outcomes were encoded and what values could and should be in each column. If they did not have a solid understanding of what their respective columns represented, we do not think they would have been able to engage so meaningfully with these ideas. The students then started to fix this, incorporating some != conditions as they had on the Identical Lollipops Identical Balloons problem.

The students had more discussion and adjusted their code. They eventually realized that they did not want to count both 1, 5, 6, 2, 3 and 1, 5, 6, 3, 2 as distinct outcomes, and so they realized they need and additional "if m > l" constraint in the last nested for-loop. They continued to write the program, ultimately arriving at the (correct) code and partial output in Fig. 15a and b.

We highlight one additional discussion they had that emphasize their understanding of the complicated encoding on this problem. First, Diana asked about whether they needed an explicit position for the B, which suggests to us that they were having sophisticated conversations about the nature of the outcomes as they related to the print statements and variables.

> *Diana*: So, I have another question, though. What prevents you from printing all six variables and pretending that like B was like the last column?
> *Charlotte*: I don't understand what you're saying.
> *Diana*: Okay, so like, you are printing like what represents the As and the Ns the first three columns are As, the second two columns are Ns. How come didn't you print a sixth column that represents the position of the B?
> *Charlotte*: I mean, you can if you would like to; but you don't have to since there's only one position missing. Then you know that's where B goes.
> *Diana*: Okay. Okay, so you're like, it saves like work, you don't have to type as much.
> *Charlotte*: Right. So like this one represents [highlights 1, 2, 3, 5, 6 in the output] – there's only, you know, one position missing, position 4, you know that's where B goes.

In this discussion, they were considering what columns represented and what information they contained, which suggests sophisticated encoding of outcomes. Charlotte could clearly articulate a mapping between the output such as 12,356 and the arrangement of AAABNN, even without needing to explicitly encode the position of the B. As a final example of the students' encoding and their understanding of these ideas, we had asked Diana to write out an arrangement of the letters in the word BANANA that corresponded to 1, 3, 5, 4, 6 [highlighted in Fig. 16a]. Diana's wrote the outcome in Fig. 16b, and said the following:

> *Diana*: So, there's an A in the first spot, and the third spot, and the fifth spot, and then there's an N in the fourth spot, I wrote an S what the heck, okay that was the fifth spot, then the sixth spot is an N, and so the B goes there.

```
# Task 2)
# How many arrangements are there of the word BANANA?
arrangements = 0
digits = [1,2,3,4,5,6]

for i in digits:
    for j in digits:
        if j > i:
            for k in digits:
                if k > j:
                    for l in digits:
                        if l > k:
                            for m in digits:
                                if m > l:
                                    arrangements = arrangements+1
                                    print(i,j,k,l,m)
print(arrangements)
```

```
1 2 3 4 5
1 2 3 4 6
1 2 3 5 6
1 2 4 5 6
1 3 4 5 6
2 3 4 5 6
6
```

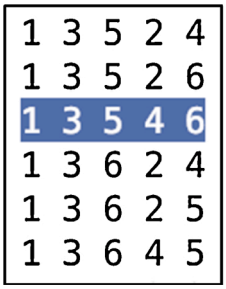
Fig. 14. The students' initial code that generated just six outcomes.

```
# Task 2)
# How many arrangements are there of the word BANANA?
arrangements = 0
digits = [1,2,3,4,5,6]

for i in digits:
    for j in digits:
        if j > i:
            for k in digits:
                if k > j:
                    for l in digits:
                        if l != k and l != j and l != i:
                            for m in digits:
                                if m > l and m != k and m != j and m != i:
                                    arrangements = arrangements+1
                                    print(i,j,k,l,m)
print(arrangements)
```

```
2 4 5 1 6
2 4 5 3 6
2 4 6 1 3
2 4 6 1 5
2 4 6 3 5
2 5 6 1 3
2 5 6 1 4
2 5 6 3 4
3 4 5 1 2
3 4 5 1 6
3 4 5 2 6
3 4 6 1 2
3 4 6 1 5
3 4 6 2 5
3 5 6 1 2
3 5 6 1 4
3 5 6 2 4
4 5 6 1 2
4 5 6 1 3
4 5 6 2 3
60
```

Fig. 15. a, b. The students' final code and some of the output for the BANANA problem.



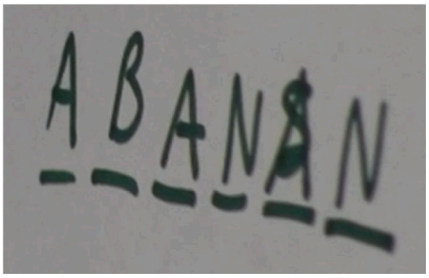


Fig. 16. a, b. Diana's mapping between a sequence and an arrangement of letters.

Both Charlotte's and Diana's demonstrated ability to relate an encoded sequence of numbers with an arrangement of the letters in the word BANANA suggests that they were effectively encoding outcomes in a sophisticated way. They could articulate an isomorphism between sequences of numbers satisfying certain constraints and arrangements of the letters in the word BANANA. Their work on this final problem type shows how the students' work on encoding outcomes persisted in an arrangement with restricted repetition problem, and it also highlights implications for why programming might be particularly effective for promoting encoding activity, which can help to strengthen students' combinatorial thinking and activity. Indeed, we are not making the claim that the students needed the computer to solve problems or to engage in encoding, and in fact Diana solved the BANANA problem quickly and correctly without the computer. The point of this example is to demonstrate and highlight how encoding arose in their programming activity. When they programmed, the need for encoding was made more acute, and the extra step of asking students to program facilitated their engagement in encoding outcomes.

5. Discussion, implications, and avenues for future research

In this section, we attempt to summarize what the results demonstrated in terms of the research questions, and we offer implications and avenues for future research. To answer the first research question *(In what ways did the encoding activity of a pair of undergraduate students who were working within a computational setting affect their combinatorial thinking and activity?),* we recall the episode that was especially important in Charlotte and Diana's encoding activity. That is, a breakthrough in their successful encoding occurred when they realized that the positions in an outcome (or a column in the output of a computer program) could contain and represent information. This transpired on the Unique Hats problem, and Diana realized that she did not need to write pairs of objects, but, rather, could write a sequence of numbers so the positions of the numbers encoded information.

Once they gained the insight that the positions themselves could convey information, they had new opportunities to develop and use a strategy of encoding outcomes as sequences or sets of positions. This was particularly useful given that they had previously been able to solve problems in which they could count sets and sequences of numbers (Lockwood & De Chenne, 2020). Thus, another fundamental part of their successful encoding was that they could translate outcomes in distribution, Category II combination, and arrangement with restricted repetition problems to problems with which they were familiar. They did this to great effect, and they adjusted their code to solve complex problems that involved distributing multiple sets of unique and identical objects. The students' work thus offers some detailed insight into what ideas and activities facilitated the students' successful encoding on increasingly complex problems.

A related point of discussion is that successfully encoding outcomes involves a considerable amount of choice and flexibility. When encoding outcomes, students get to make decisions about how to represent an outcome that could have an impact on their ability to solve that problem correctly, and we saw this play out repeatedly for Charlotte and Diana. Such flexibility can give the students ownership over their mathematical process, potentially helping them to feel empowered in their mathematical activity. An implication of these results, then, is that we should try to give students opportunities to engage in flexible, creative, and productive encoding of outcomes. This is important both for their success in solving counting problems, and also for empowering them as having some control over the mathematical processes they implement. To help students encode counting problems successfully, we might offer direct and targeted questions such as those we saw in the results – researchers and teachers could give students tasks that involve sophisticated encoding and model such encoding for them. This may involve having teachers explicitly discuss mappings and isomorphisms, demonstrating how students could leverage isomorphisms meaningfully to encode outcomes judiciously and solve new classes of problems. Additional research studies could investigate effective ways to teach students to encode outcomes, including designing specific sets of tasks and materials with this goal in mind. Such work could be undertaken both in computational and strictly by-hand settings.

In terms of the second research question, *(What role did the computer program and output play in their encoding activity?)*, we gain some insight into why writing and running a program and producing output can be useful in encoding outcomes. In particular, in the computer programs the students wrote, there was a print statement with some number of variables; then, when run, the program produces outputs with exactly that number of columns. This is not itself a surprising or groundbreaking insight, but for Diana and Charlotte, this was particularly useful. Understanding the importance of *positions* was incredibly important for the students. Often counting problems can be solved by making arguments about positions, or "spots," something Reed & Lockwood (in press) refer to as *positional reasoning*. The fact that the output of Python code of these nested for-loops is produced in columns strongly reinforces this idea there are positions that represent different stages in a counting process, and it can also reinforce the idea that certain positions (or columns) can carry meaning. The fact that the computer code's output was organized into columns reinforced this idea, and it underscored this valuable insight that they could also let the columns (or, positions) in the output stand for something. Indeed, they referred repeatedly to columns in the outputs of the code, and they often used the columns in creative ways. Thus, we argue that the simple mechanics of way the computer produced outputs reinforced what was a fundamental idea.

Another point of discussion is the extent to which the by-hand work and the computer together contributed to the students' reasoning and activity. We have intentionally framed both within the computational setting (as noted in Section 2.3.2), because we argue that even the students by-hand work was, in this teaching experiment, motivated and framed within an overall goal of programming a solution to the problem. In the particular case of the students' breakthrough about positions being able to convey information, the key insight came about during by-hand work, and there is no reason to believe these ideas could not be elicited completely independently of a computer. We are not claiming that the computer was necessary or even played into Diana's insight in that moment. However, we would argue that additionally being asked to program the problem did, at times, have an effect on their motivation to find a way to encode outcomes. For instance, Diana and Charlotte ran into some computational limitations in their coding, such as when they could not figure out how to write code that would output hat-people pairs. We suggest that once they successfully saw that sequences of numbers could represent an outcome (where the columns represented people getting unique hats on the Hats problem), they extended and used that strategy effectively, to the point where on subsequent problems the columns represented people getting identical lollipops, unique balloons, or the position of an A. Because they could use essentially similar code on these problems – nested loops with some conditional statements to constrain the outputted sequences – we propose that they saw value in translating outcomes for the new problem into these sequences of numbers that they knew how to program. Again, we are not claiming that they could not have had such insights without programming, but we hypothesize that their programming activity even implicitly motivated them to engage in the sophisticated encoding that we observed.

An implication of this work, then, is that there may be something particularly powerful about combinatorial encoding within a computational setting. The computer interface and features served to reinforce the students' big idea about encoding, which is that positions (or columns) could themselves encode information. Further, having to program a solution may necessitate sophisticated encoding in ways that by-hand solutions may not. If indeed it is a goal for us to have students engage in encoding activity, this setting of combinatorics and encoding provides motivation for students to be careful and thoughtful about how they encode outcomes. Conversely, our results also suggest that if we want to foster computational activity for students, this context of combinatorics and encoding outcomes offers a promising topic in which to integrate students' mathematical thinking and activity within a computational setting.

In terms of avenues for future research, we feel there is more to study about students' different solution strategies toward counting problems, particularly when by-hand and computational solutions elicit different strategies. We saw this most clearly on the BANANA problem, where Diana successfully enacted a labeling approach by-hand, and Charlotte implemented a selection approach when programming the problem. It would be interesting to explore more deeply whether there are differences between students' by-hand solutions and their computational solutions, and to investigate how students think about those respective solution methods. It is often good for students to have multiple ways to solve a problem, and we have evidence for how the computer may offer opportunities for distinct approaches. Thus, further research could explore, particularly if by-hand and computational approaches involve distinct ways of encoding outcomes. In addition, researchers could explore how students might engage in such encoding activity in more naturalistic, classroom settings. Because our work occurred in a teaching experiment setting, future work could explore whether and how students' reasoning might develop in situations when they do not have as much sustained, repeated exposure to encoding as they have in a teaching experiment.

CRediT authorship contribution statement

Elise Lockwood: Data curation, Conceptualization, Writing - original draft. **Adaline De Chenne:** Writing - review & editing.

Acknowledgment

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

Appendix A. For readers who may not be familiar with Python code, Appendix A shows code we wrote that lists all of the outcomes for each problem discussed in this paper. This code also reflects the students' approaches to these problems

Problem name and statement	Solution
Unique Hats: Suppose you have three unique hats and five friends. How many ways are there to distribute the three hats among your friends, if no one can have more than one hat?	$P(5,3) = 5 \cdot 4 \cdot 3 = 60$
arrangements = 0 Friends = [1, 2, 3, 4, 5] **for** f1 **in** Friends: **for** f2 **in** Friends: **if** f2 != f1: **for** f3 **in** Friends: **if** f3 != f1 **and** f3 != f2: arrangements = arrangements + 1 print(f1,f2,f3) print(arrangements)	
Lollipop: How many ways are there to give 3 identical lollipops to 8 different kids (if no kid can have more than 1 lollipop)?	$\binom{8}{3} = \frac{8 \cdot 7 \cdot 6}{3!} = 56$
arrangements = 0 kids = [1, 2, 3, 4, 5, 6, 7, 8] **for** i **in** kids: **for** j **in** kids: **if** j > i: **for** k **in** kids: **if** k > j: arrangements = arrangements + 1 print(i,j,k) print (arrangements)	
Identical Lollipop Identical Balloon: How many ways are there to distribute 3 identical lollipops and 3 identical red balloons to 8 different kids (if no kid can have more than 1 object).	$\binom{8}{3} \cdot \binom{5}{3} = 560$
arrangements = 0 kids = [1, 2, 3, 4, 5, 6, 7, 8] **for** i **in** kids: **for** j **in** kids: **if** j > i: **for** k **in** kids: **if** k > j: **for** l **in** kids: **if** l != k **and** l != j **and** l != i: **for** m **in** kids: **if** m > l **and** m != i **and** m != j **and** m != k: **for** n **in** kids: **if** n > m **and** n != i **and** n != j **and** n != k: arrangements = arrangements + 1 print(i,j,k,l,m,n) print (arrangements)	
Unique Lollipop Identical Balloon: How many ways are there to distribute 3 unique lollipops and 3 identical red balloons to 8 different kids (if no kid can have more than 1 object).	$P(8,3) \cdot \binom{5}{3} = 3,360$
arrangements = 0 kids = [1, 2, 3, 4, 5, 6, 7, 8] **for** i **in** kids: **for** j **in** kids: **if** j != i: **for** k **in** kids: **if** k != j **and** k != i: **for** l **in** kids: **if** l != k **and** l != j **and** l != i: **for** m **in** kids: **if** m > l **and** m != i **and** m != j **and** m != k: **for** n **in** kids: **if** n > m **and** n != i **and** n != j **and** n != k: arrangements = arrangements + 1 print(i,j,k,l,m,n) print (arrangements)	
Heads/Tails: I flip a coin 10 times in a row. How many different outcomes have exactly the same numbers of heads and tails?	$\binom{10}{5} = 252$
arrangements = 0 digits = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] **for** i **in** digits: **for** j **in** digits: **if** j > i: **for** k **in** digits: **if** k > j: **for** l **in** digits: **if** l > k: **for** m **in** digits: **if** m > l: arrangements = arrangements + 1 print(i,j,k,l,m) print(arrangements)	
BANANA: How many arrangements are there of the word BANANA?	$\binom{6}{3} \cdot \binom{3}{2} \cdot \binom{1}{1} = 60$
arrangements = 0 digits = [1, 2, 3, 4, 5, 6] **for** i **in** digits: **for** j **in** digits: **if** j > i: **for** k **in** digits: **if** k > j: **for** l **in** digits: **if** l != k **and** l != j **and** l != i: **for** m **in** digits: **if** m > l **and** m != k **and** m != j **and** m != i: arrangements = arrangements + 1 print(i,j,k,l,m) print(arrangements)	

References

Annin, S. A., & Lai, K. S. (2010). Common errors in counting problems. *Mathematics Teacher, 103*(6), 402–409.
Auerbach, C., & Silverstein, L. B. (2003). *Qualitative data: An introduction to coding and analysis*. New York: New York University Press.
Batanero, C., Navarro-Pelayo, V., & Godino, J. (1997). Effect of the implicit combinatorial model on combinatorial reasoning in secondary school pupils. *Educational Studies in Mathematics, 32*, 181–199.
Benton, L., Saunders, P., Kalas, I., Hoyles, C., & Noss, R. (2018). Designing for learning mathematics through programming: A case study of pupils engaging with place value. *International Journal of Child-Computer Interaction, 16*, 68–76.
Blikstein, P. (2018). *Pre-College computer science education: A survey of the Field.* Retrieved from. Mountain View, CA: Google LLC https://goo.gl/gmS1Vm.
Buteau, C., & Muller, E. (2017). Assessment in undergraduate programming-based mathematics courses. *Digital Experiences in Mathematics Education, 3*, 97–114. https://doi.org/10.1007/s40751-016-0026-4.
CadwalladerOsker, T. (2013). The labeling strategy: Moving beyond order in counting problems. *Problems, Resources, and Issues in Mathematics Undergraduate Studies, 23*(10), 921–934.
Cetin, I., & Dubinsky, E. (2017). Reflective abstraction in computational thinking. *Journal of Mathematical Behavior, 47*, 70–80.
DeJarnette, A. F. (2019). Students' challenges with symbols and diagrams when using a programming environment in mathematics. *Digital Experiences in Mathematics Education, 5*, 36–58. https://doi.org/10.1007/s40751-018-0044-5.
Disessa, A. A. (2018). Computational literacy and "The Big picture" concerning computers in mathematics education. *Mathematical Thinking and Learning, 20*(1), 3–31. https://doi.org/10.1080/10986065.2018.1403544.
Drijvers, P., Doorman, M., Boon, P., Reed, H., & Gravemeijer, K. (2010). The teacher and the tool: Instrumental orchestrations in the technology-rich mathematics classroom. *Educational Studies in Mathematics, 75*, 213–234. https://doi.org/10.1007/s10649-010-9254-5.
Dubois, J. G. (1984). Une systematique des configurations combinatoires simples. *Educational Studies in Mathematics, 15*(1), 37–57.

Fenton, W., & Dubinsky, E. (1996). *Introduction to discrete mathematics with ISETL*. New York: Springer-Verlag.
Feurzeig, W., Papert, S., & Lawler, B. (2011). Programming-languages as a conceptual framework for teaching mathematics. *Interactive Learning Environments, 19*(5), 487–501.
Hadar, N., & Hadass, R. (1981). The road to solving a combinatorial problem is strewn with pitfalls. *Educational Studies in Mathematics, 12*, 435–443.
Hickmott, D., Prieto-Rodriguez, E., & Holmes, K. (2018). A scoping review of studies on computational thinking in K-12 mathematics classrooms. *Digital Experiences in Mathematics Education, 4*, 48–69. https://doi.org/10.1007/s4075-017-0038-8.
Hollenbrands, K. F. (2007). The role of a dynamic software program for geometry in the strategies high school mathematics students employ. *Journal for Research in Mathematics Education, 38*(2), 164–192.
Hoyles, C., & Noss, R. (2015). A computational lens on design research. In S. Prediger, K. Gravemeijer, & J. Confrey (Eds.), *Design research with a focus on learning processes: An overview on achievements And challenges*. ZDM, (47)6, 1039 -1045.
JetBrains. (2017). *PyCharm* [online] JetBrains. Available at: https://www.jetbrains.com/pycharm.
Kapur, J. N. (1970). Combinatorial analysis and school mathematics. *Educational Studies in Mathematics, 3*(1), 111–127.
Kaput, J. J., & Thompson, P. W. (1994). Technology in mathematics education research: The first 25 years in JRME. *Journal for Research in Mathematics Education, 25*(6), 667–684.
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*, 154–171. https://doi.org/10.1007/s40751-017-0031-2.
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. *Journal of Mathematical Behavior, 32*, 251–265. https://doi.org/10.1016/j.jmathb.2013.02.008.
Lockwood, E. (2014a). A set-oriented perspective on solving counting problems. *For the Learning of Mathematics, 34*(2), 31–37.
Lockwood, E. (2014b). Both answers make sense! Using the set of outcomes to reconcile differing answers in counting problems. *Mathematics Teacher, 108*(4), 296–301.
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*, 303–346. https://doi.org/10.1007/s40753-019-00108-2.
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., & Reed, Z. (2018). Reinforcing mathematical concepts and developing mathematical practices through combinatorial activity. In E. W. Hart, & E. J. Sandefur (Eds.), *Teaching and learning of discrete mathematics worldwide: Curriculum and research* (pp. 93–110). Cham, Switzerland: Springer.
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., 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.
Lockwood, E., Wasserman, N. H., & McGuffey, W. (2018). Classifying combinations: Do students distinguish between different categories of combination problems? *International Journal of Research in Undergraduate Mathematics Education, 4*(2), 305–322. https://doi.org/10.1007/s40753-018-0073-x.
Maher, C. A., Powell, A. B., & Uptegrove, E. B. (Eds.). (2011). Combinatorics and Reasoning: Representing, Justifying, and Building Isomorphisms. New York: Springer.
Mamona-Downs, J., & Downs, M. (2004). Realization of techniques in problem solving: The construction of bijections for enumeration tasks. *Educational Studies in Mathematics, 56*, 235–253.
Muter, E. M., & Uptegrove, E. B. (2011). Representations and connections. In C. A. Maher, A. B. Powell, & E. B. Uptegrove (Eds.), *Combinatorics and reasoning: Representing, justifying, and building isomorphisms* (pp. 105–120). New York: Springer.
NGSS Lead States. (2013). *Next generation science standards: For States, By States*. Washington, DC: The National Academies Press.
Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York: Basic books.
Reed, Z. & Lockwood, E. (In press). Leveraging a categorization activity to facilitate productive generalizing activity and combinatorial reasoning. To appear in Cognition and Instruction.
Sinclair, N., & Patterson, M. (2018). The dynamic geometrisation of computer programming. *Mathematical Thinking and Learning, 20*(1), 54–74. https://doi.org/10.1080/10986065.2018.1403541.
Speiser, R. (1997). Block towers and binomials. *Journal of Mathematical Behavior, 16*(2), 113–124.
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*. Mahwah, NJ: Lawrence Erlbaum Associates.
Tarlow, L. D. (2011). Pizzas, towers, and binomials. In C. A. Maher, A. B. Powell, & E. B. Uptegrove (Eds.), *Combinatorics and reasoning: Representing, justifying, and building isomorphisms* (pp. 121–132). New York: Springer.
Tucker, A. (2002). *Applied combinatorics* (4th ed.). New York: John Wiley & Sons.
Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science and Education Technology, 25*, 127–147. https://doi.org/10.1007/s10956-015-0581-5.
Wing, J. M. (2016). *Computational thinking: 10 years later*. Retrieved from https://phys.org/news/2016-03-years.html#jCp.