Computational Thinking and Activity in STEM Education: What Happens When Math Students Engage with Python Code

Elise Lockwood
Oregon State University

Yale University, December 7, 2018

A Relevant Problem in STEM

- Computing is increasingly becoming an integral component of scientific and mathematical work
- The Next Generation Science Standards explicitly include “Using Mathematics and Computational Thinking”
- There seems to be growing attention on computing in STEM
  - Our department has a “computational” requirement in our major
  - There are increasing job postings for “computational mathematicians”
  - There are newly-developed computational departments (The Department of Computational Mathematics, Science and Engineering at MSU)

A Relevant Problem in STEM

- Some scientists believe that computing complements theory and experimentation as pillars of science (Wing, 2017)
- “By 2020, one of every two jobs in the STEM fields will be in computing” (Association for Computing Machinery, 2014)
- There is a potential relationship between learning computation and learning mathematical and scientific concepts and practices (Eisenberg, 2002; National Research Council, 2011; Wilensky et al., 2014)

A Relevant Problem in STEM Education

- As computing becomes more integrated into work in STEM, there is a need for STEM education researchers to address questions related to computing in STEM
  - What does computational thinking and activity entail within respective STEM disciplines?
  - How can we use computing to help students learn specific STEM concepts?
  - Within STEM disciplines, how do we effectively integrate and teach computing to our students?

A Relevant Problem in STEM Education

- Through an NSF-funded project, I am attempting to address one narrow slice of these broader questions
- I focus on the STEM discipline of Mathematics
  - I study undergraduate students’ work with combinatorics
- I focus on the computational activity of basic Python programming
  - Students create and examine Python code to solve combinatorial problems
- I hope that my study can inform these broader questions related to STEM

Goals for this Talk

- Problematize the need for more STEM Education research related to computational thinking and activity
- Define some key terms
- Focus on the context of undergraduate combinatorics students engaging in Python programming
  - Present affordances and limitations of having students solve combinatorial problems in a computational setting
  - Demonstrate examples of computational thinking in this mathematical context
- Conclude with ideas for potential areas of research for broader STEM Education audiences

Research Goals and Questions

1. Broadly, how does engaging in computational thinking and activity help students learn mathematical or scientific content?
2. Does engaging in basic Python programming tasks
   a. Help to reinforce the relationship between counting processes and sets of outcomes?
   b. Help students successfully solve counting problems?
3. What are affordances and limitations of having students solve counting problems in a computational setting?

Defining Key Terms

- Computational Thinking
- Computing
- Combinatorics

Historical Context of Computational Thinking

- The idea of using computing in learning has a long history (Perlis, 1962; Papert, 1972; 1980)
- In 1996, Papert introduced the term *computational thinking* to refer to "the affordances of computational representations for expressing powerful ideas" (Weintrop, et al. 2016)
- In 2006, Jeannette Wing renewed the modern conversation about CT

Historical Context of Computational Thinking

- Wing initially described CT as “taking an approach to solving problems, designing systems and understanding human behaviour that draws on concepts fundamental to computer science” (2006, p. 33)
- Wing characterized CT broadly, encompassing activities such as:
  - “**thinking recursively**”
  - “**using abstraction** and decomposition when attacking a large complex task or designing a large complex system”
  - “**using heuristic reasoning** to discover a solution”
  - “**making trade-offs** between time and space and between processing power and storage capacity”

Historical Context of Computational Thinking

- Wing's initial descriptions were quite vague and far reaching
- Grover and Pea (2013) reviewed CT literature, and they found "definitional confusion" and widely varying perspectives of CT
- "Clearly, much remains to be done to help develop a more lucid theoretical and practical understanding of computational competencies in children...It is time to redress the gaps and broaden the 21st-century academic discourse on computational thinking" (p. 42)

My Working Definition of Computational Thinking (Wing, 2014)

- **CT is the way of thinking that one uses to formulate a problem and/or express its solution(s) in such a way that a computer could effectively carry it out.**

My Working Definition of Computational Thinking (Wing, 2014)

- **CT is the <u>way of thinking</u> that one uses to formulate a problem and/or express its solution(s) in such a way that a computer could effectively carry it out.**
- CT is about a person’s thinking: "CT is about the thought processes that one goes through before you even write some code, before you’re even creating an artifact” (Wing, 2016)

My Working Definition of Computational Thinking (Wing, 2014)

- **CT is the way of thinking that one uses to formulate a problem and/or express its solution(s) in such a way that a computer could effectively carry it out.**
- Computational thinking can be leveraged in problem formulation (even if a solution is not found), and it can be leveraged in expressing a solution (even if the formulation was not computational)

My Working Definition of Computational Thinking (Wing, 2014)

- **CT is the way of thinking that one uses to formulate a problem and/or express its solution(s) in such a way that a computer could <u>effectively</u> carry it out.**
- There may be multiple pathways to formulate the problem or solutions
- An ultimate goal is correctness, and efficiency must be considered

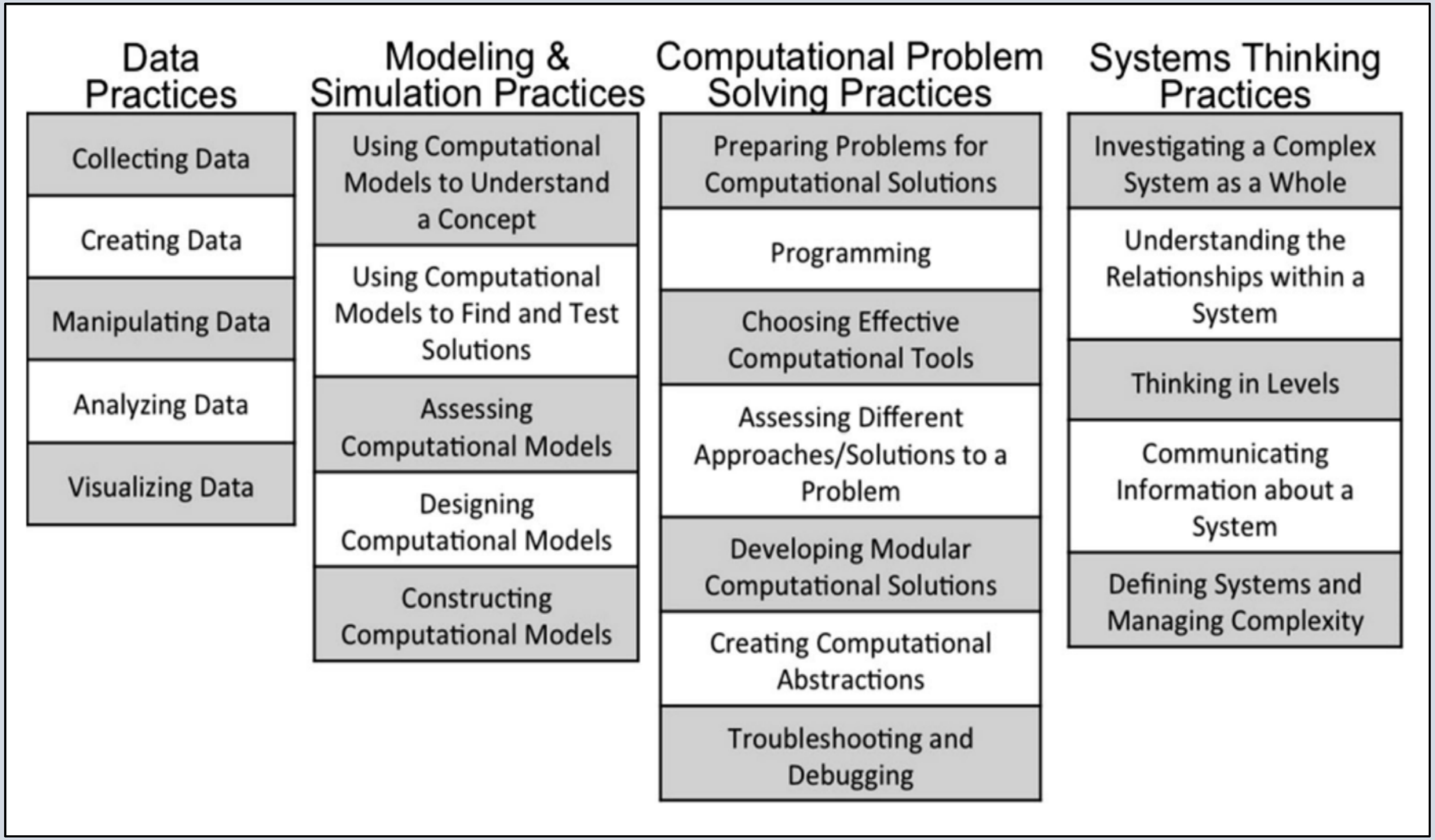
My Working Definition of Computational Thinking (Wing, 2014)

- **CT is the way of thinking that one uses to formulate a problem and/or express its solution(s) in such a way that a computer could effectively carry it out.**

Defining Computing

- “In a general way, we can define computing to mean any goal-oriented activity requiring, benefiting from, or creating computers” (Joint Task Force for Computing Curricula, 2005, p. 9)
- I define computing as *the practice of using tools to perform (mathematical)* ***calculations*** *or to develop or implement* ***algorithms*** *in order to accomplish a (mathematical) goal*
- I use the terms computation and computing interchangeably

Taxonomy of Computational Thinking Practices for Mathematics and Science



Weintrop, et al., 2016

Why Combinatorics?

Why Combinatorics?

- Enumerative combinatorics is a domain of mathematics that determines how many elements in a set satisfy certain constraints
- Such problems are called “counting problems”
- Counting has applications in a variety of areas, including probability, statistics, and computer science (e.g., English, 1991; Kapur, 1970; NCTM, 1989, 2000)
  - # of distinct passwords
  - # of routes through a city grid from point A to point B
  - # of ways to roll to get a sum of 13 when you roll 3 20-sided dice
  - # of DNA sequences of a given length with certain specified gene structures

Your Turn! The 3-Letter Sequences Problem

How many 3-letter sequences can be made using the letters *a, b, c, d, e, f,*

1. If **no repetition** of letters is allowed?
2. If the sequence **must contain** *e*, and **no repetition** of letters is allowed?
3. If the sequence **must contain** *e*, and **repetition** of letters **is** allowed?

Why Combinatorics?

- Counting problems are easy to state…

  but they can be surprisingly difficult to solve

Combinatorics Books Say Counting is Hard

- Martin’s (2001) first chapter is entitled “Counting is Hard”

  He points out that “there are few formulas and each problem seems to be different”
- Tucker (2002) says of his counting chapter, “we discuss counting problems for which no specific theory exists…it is the most challenging and most valuable chapter in this book”

Math Education Research Says Counting is Hard

- Researchers report relatively low success rates (Roa, 2000; Eizenberg & Zaslavsky, 2004; Lockwood & Gibson, 2015; Kavousian; 2006)
- Annin and Lai (2010) say “Mathematics teachers are often asked, ‘What is the most difficult topic for you to teach?’ Our answer is teaching students to count”

Reasons Behind Student Difficulties with Counting

- Counting involves non-routine problems, many of which seem different (Kapur, 1970; Tucker, 2002)
- Counting problems often involve very large sets of outcomes, and they are difficult to verify (Eizenberg & Zaslavsky, 2004)
- Students struggle to understand, justify, and appropriately apply counting formulas (e.g., Batanero, et al., 1997; Lockwood, 2014)

Reasons Behind Student Difficulties with Counting

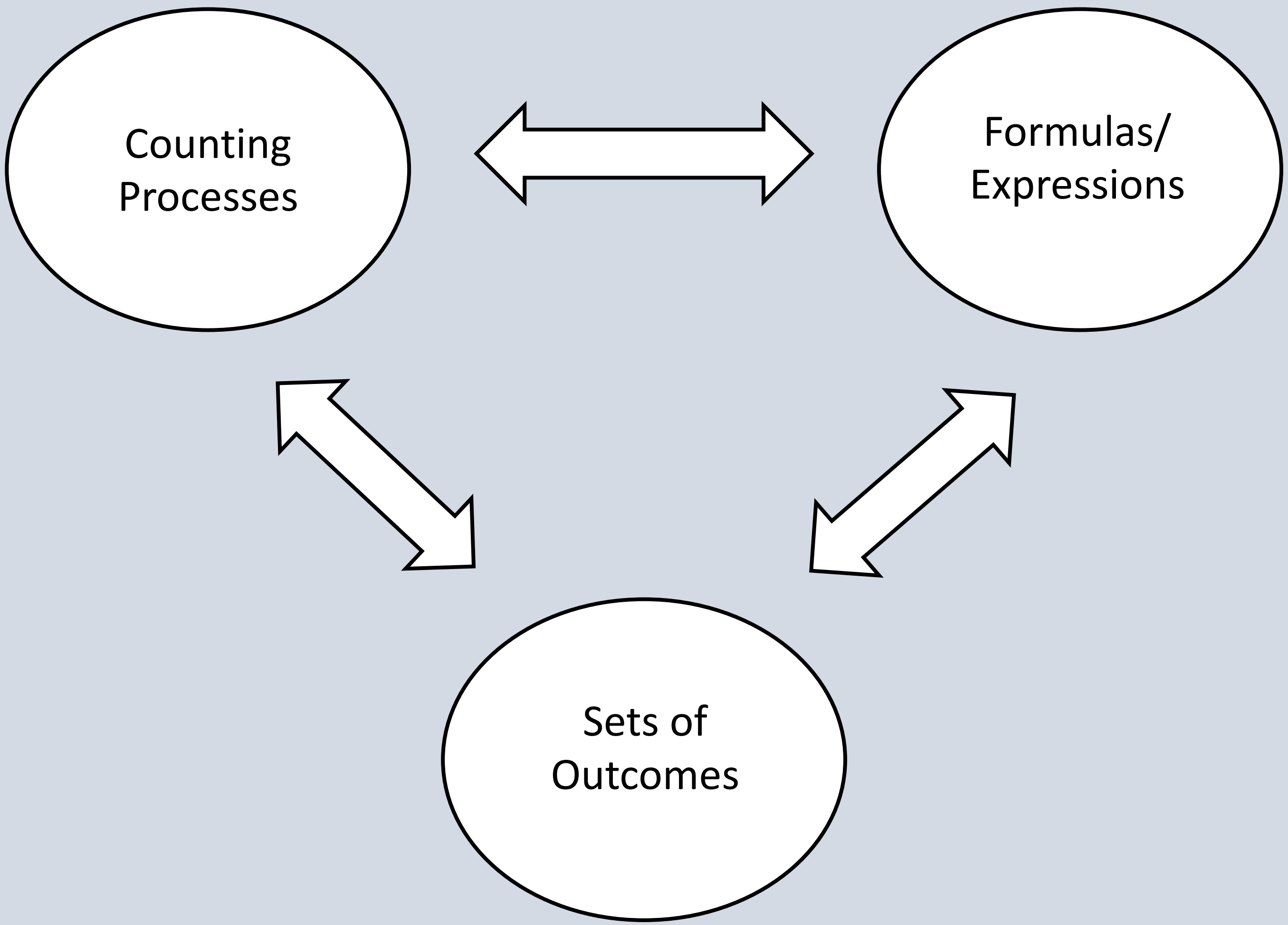
- Lockwood (2014) reports the following exchange when a discrete mathematics student was solving a counting problem:
- *Student*: I’m doing the combination ones because I’m pretty sure order doesn’t matter with combination.
- *Int.*: Why?
- *Student*: I’m not sure about that one (laughs). I just kind of go off my gut for it, on the ones that don’t specifically say order matters or it doesn’t matter.

Characterizing Combinatorial Thinking and Activity

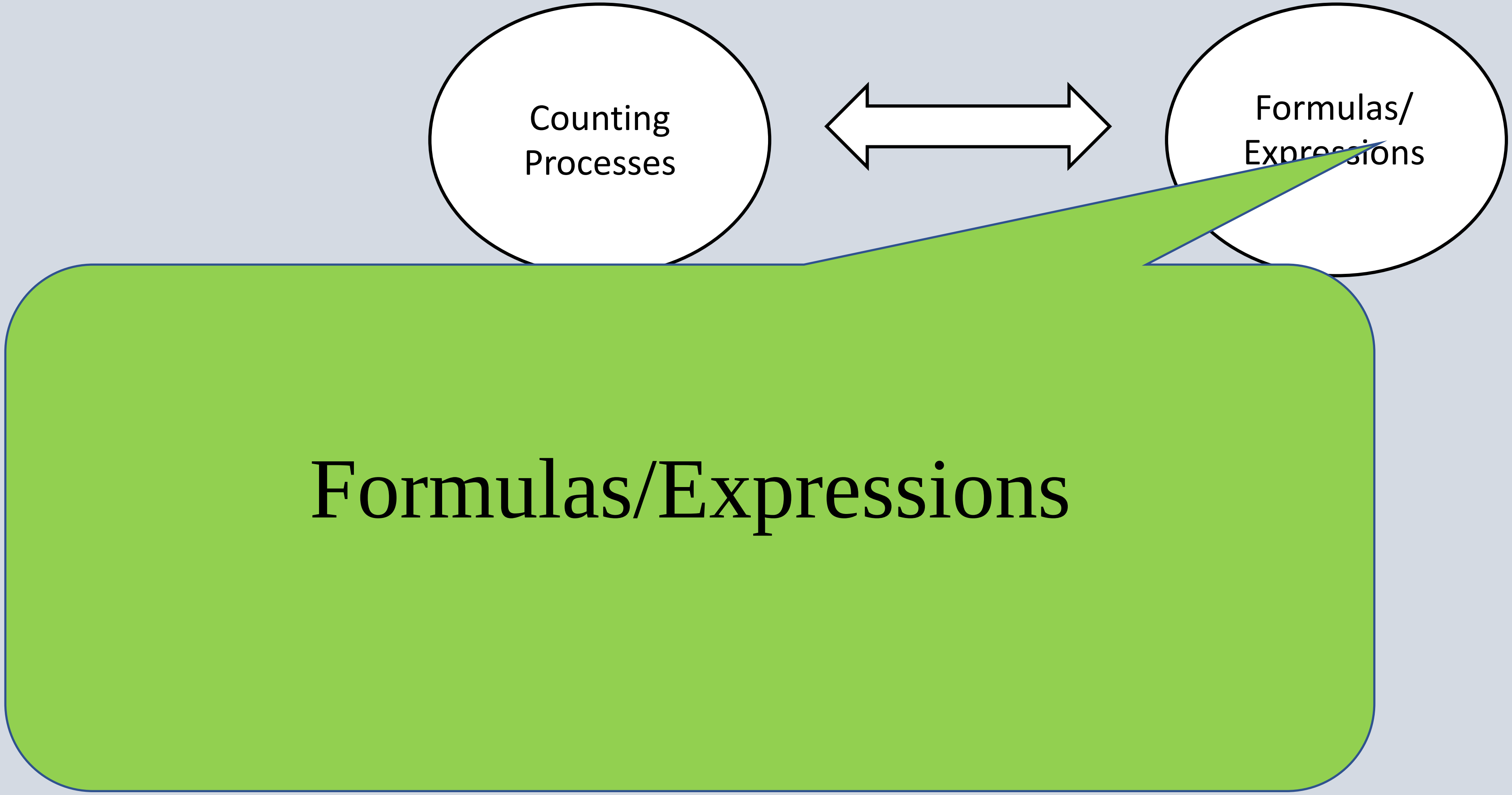
A Model of Students' Combinatorial Thinking

- By **model**, I mean a system for identifying, describing, and explaining certain phenomena related to a mathematical topic (in this case, combinatorial thinking)
- By **students' combinatorial thinking**, I mean my interpretation of students' thinking via their observable language and activity
- The model was theoretically and empirically developed

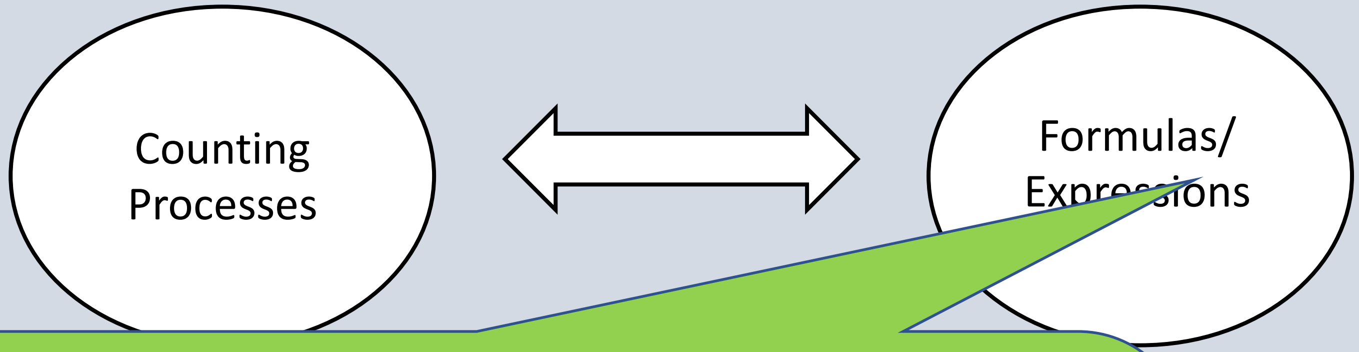
A Model of Students' Combinatorial Thinking



A Model of Students’ Combinatorial Thinking

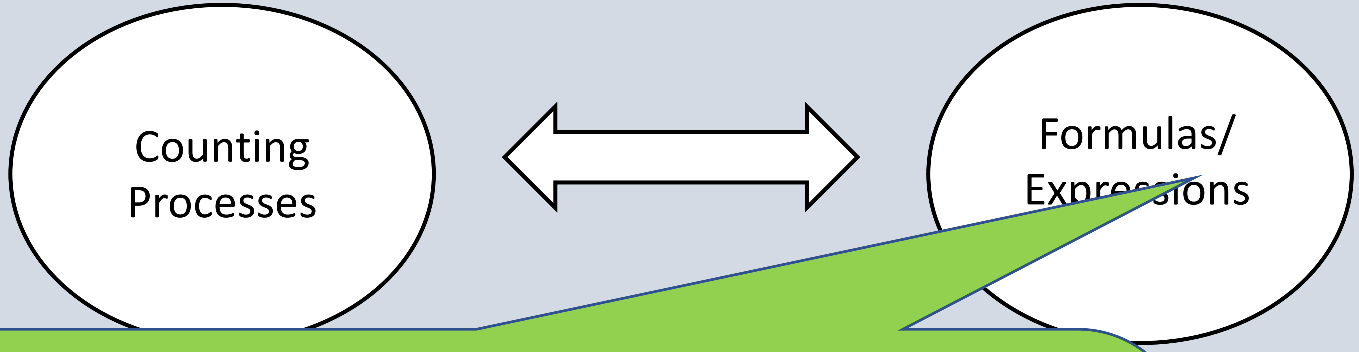


A Model of Students' Combinatorial Thinking



Mathematical expressions that yield some numerical value
(often considered "the answer").

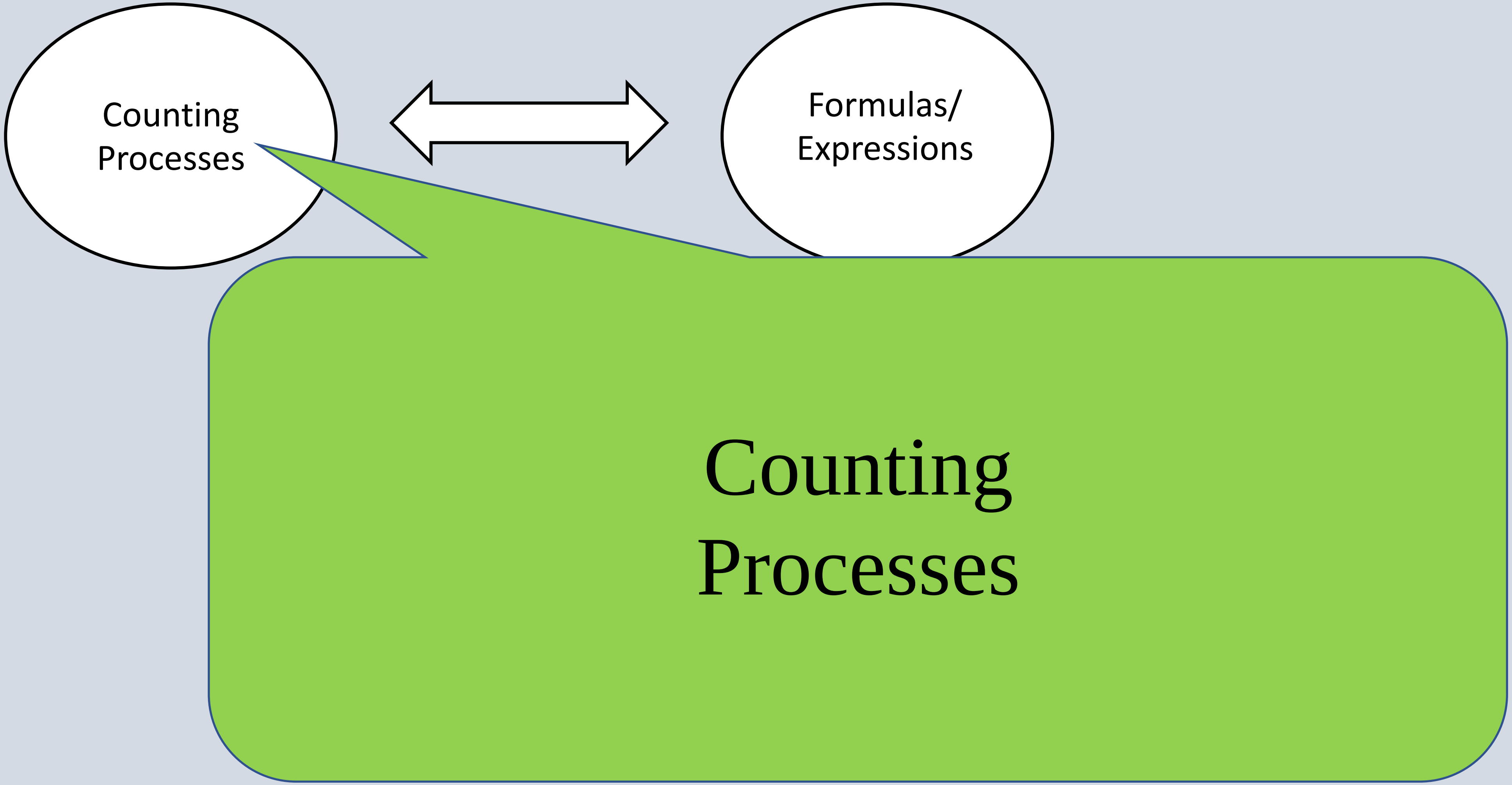
A Model of Students' Combinatorial Thinking



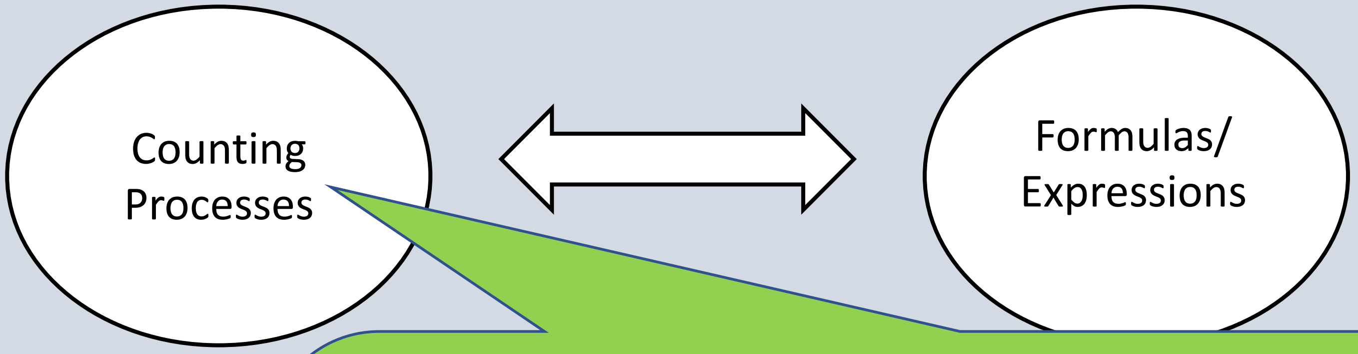
$$\binom{8}{3} \cdot 26^5 \qquad 9 \cdot 13 + 3 \cdot 12 \qquad \binom{5}{2}\binom{8}{3}$$

$$n!$$

A Model of Students' Combinatorial Thinking

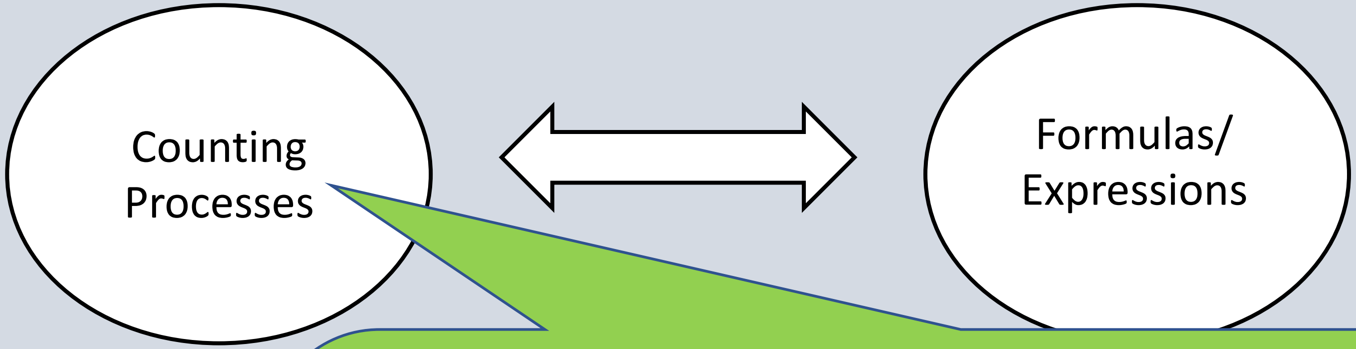


A Model of Students' Combinatorial Thinking



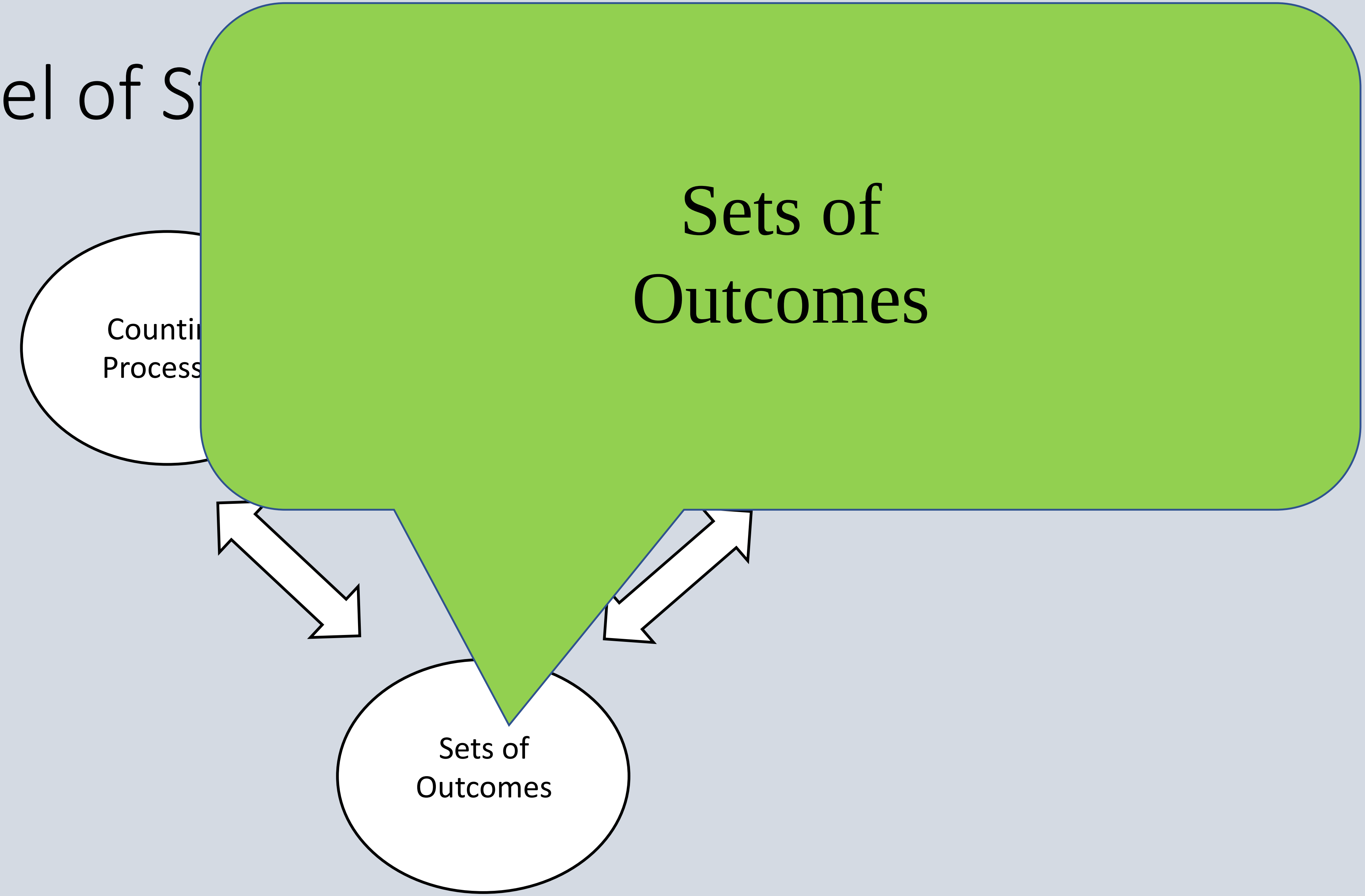
The enumeration process (or series of processes) in which a counter engages as they solve a counting problem.

A Model of Students' Combinatorial Thinking

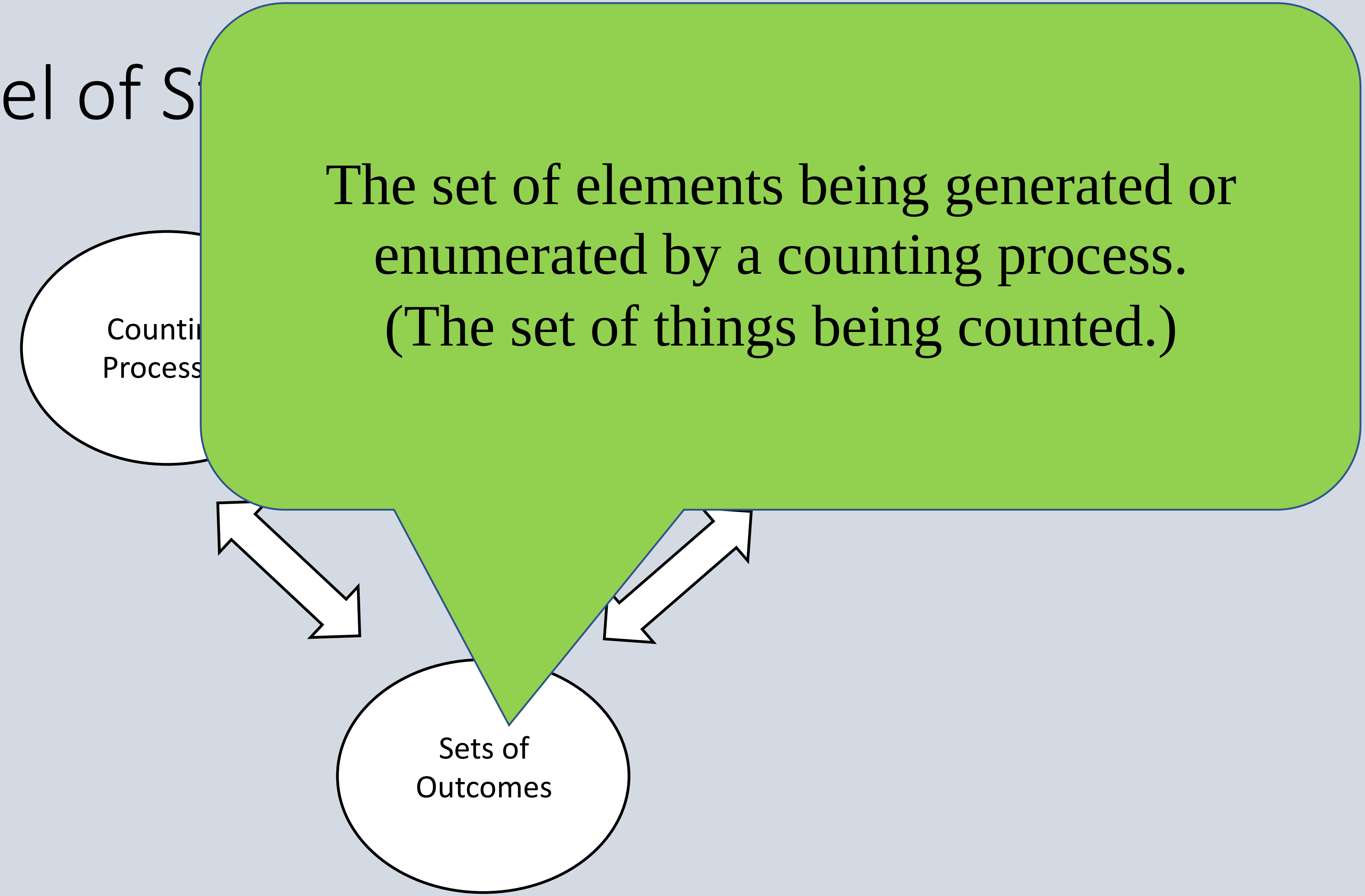


Using a particular case breakdown;
Applying the multiplication principle.

A Model of S

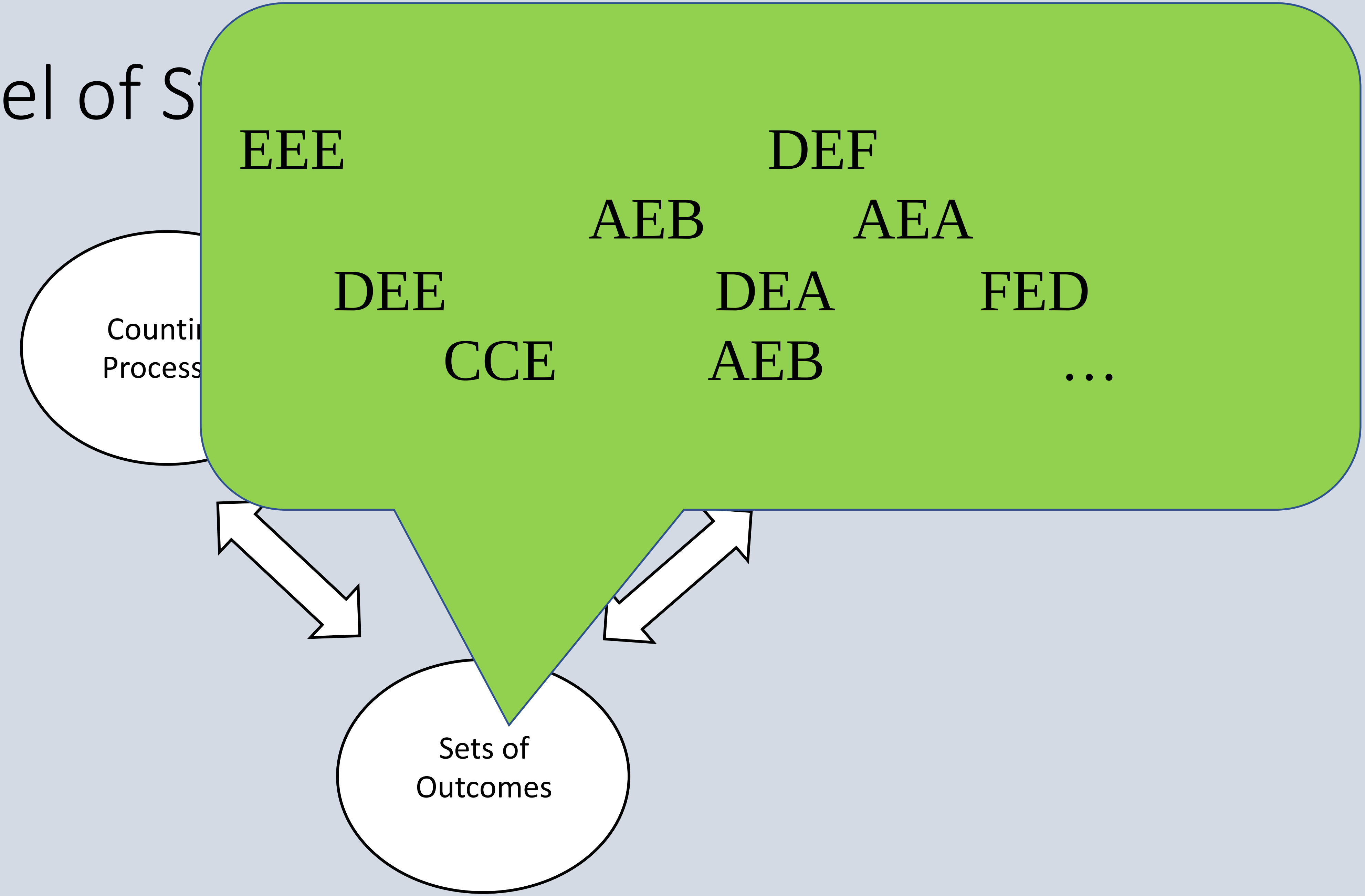


A Model of S



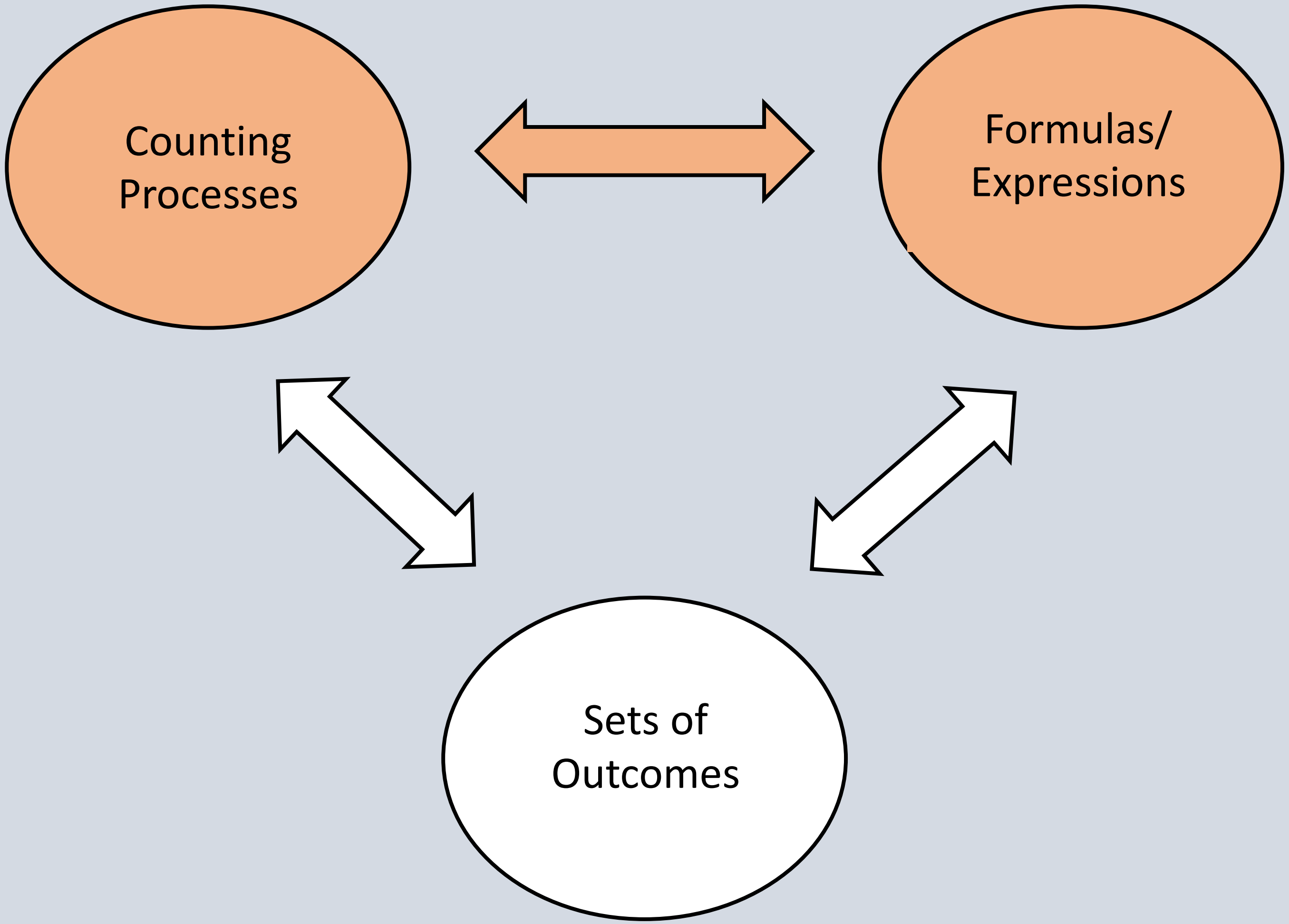
Counti
Process
The set of elements being generated or enumerated by a counting process.
(The set of things being counted.)
Sets of Outcomes

A Model of S



EEE
DEF
AEB
AEA
DEE
DEA
FED
CCE
AEB
...
Counti
Process
Sets of
Outcomes

Counting Processes and Formulas/Expressions



Counting Processes and Formulas/Expressions

- How many 3 letter passwords are there using the letters A, B, C (repetition allowed)?

___ ___ ___

Counting Processes and Formulas/Expressions

- How many 3 letter passwords are there using the letters A, B, C (repetition allowed)?

3 ___ ___ ___

Counting Processes and Formulas/Expressions

- How many 3 letter passwords are there using the letters A, B, C (repetition allowed)?

$\underline{\ 3\ } \cdot \underline{\ 3\ }\ \underline{\quad}$

Counting Processes and Formulas/Expressions

- How many 3 letter passwords are there using the letters A, B, C (repetition allowed)?

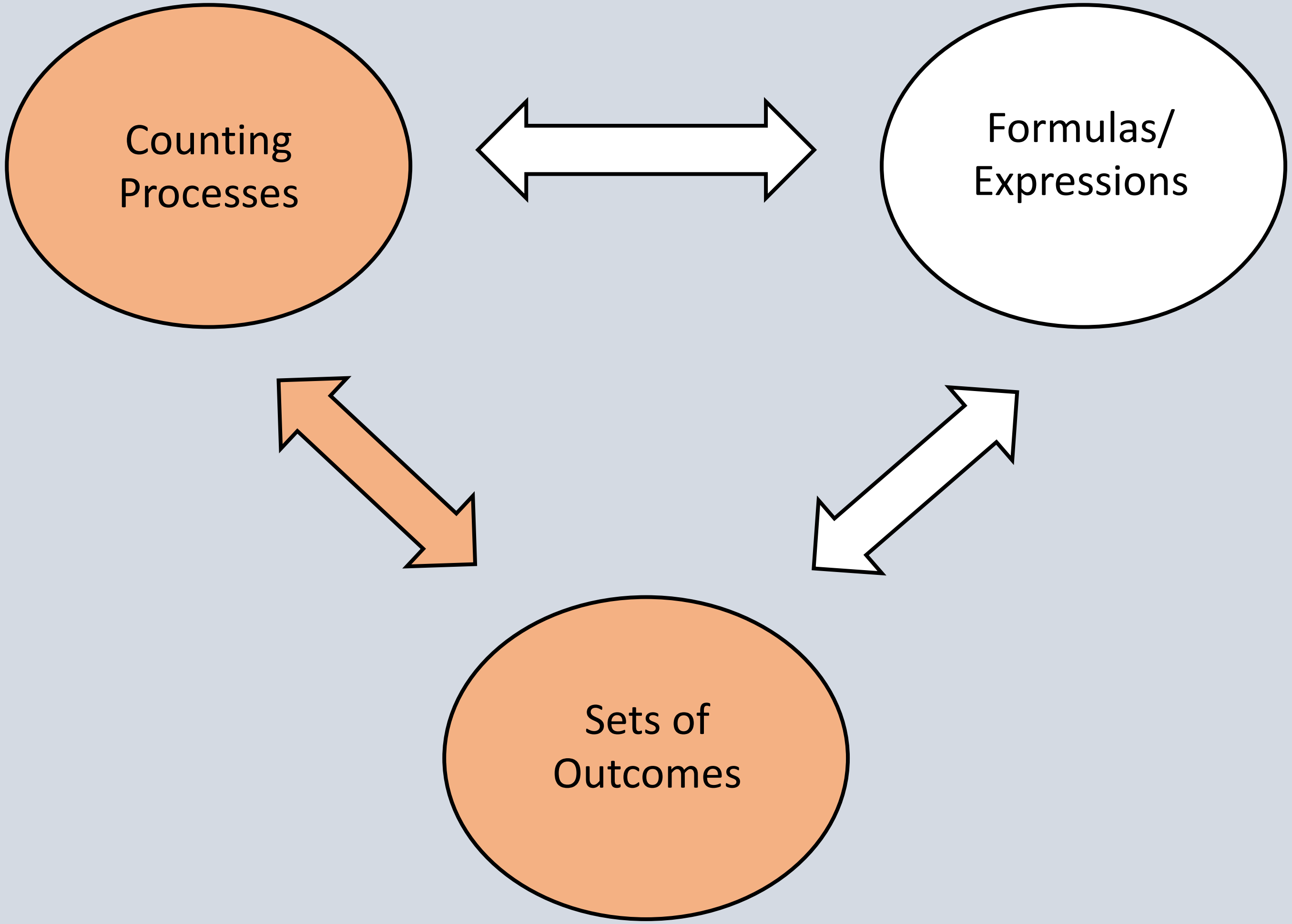
$$\underline{\;3\;} \cdot \underline{\;3\;} \cdot \underline{\;3\;}$$

Counting Processes and Formulas/Expressions

- How many 3 letter passwords are there using the letters A, B, C (repetition allowed)?

$$\underline{\ 3\ } \cdot \underline{\ 3\ } \cdot \underline{\ 3\ } = 3^3$$

Counting Processes and Sets of Outcomes



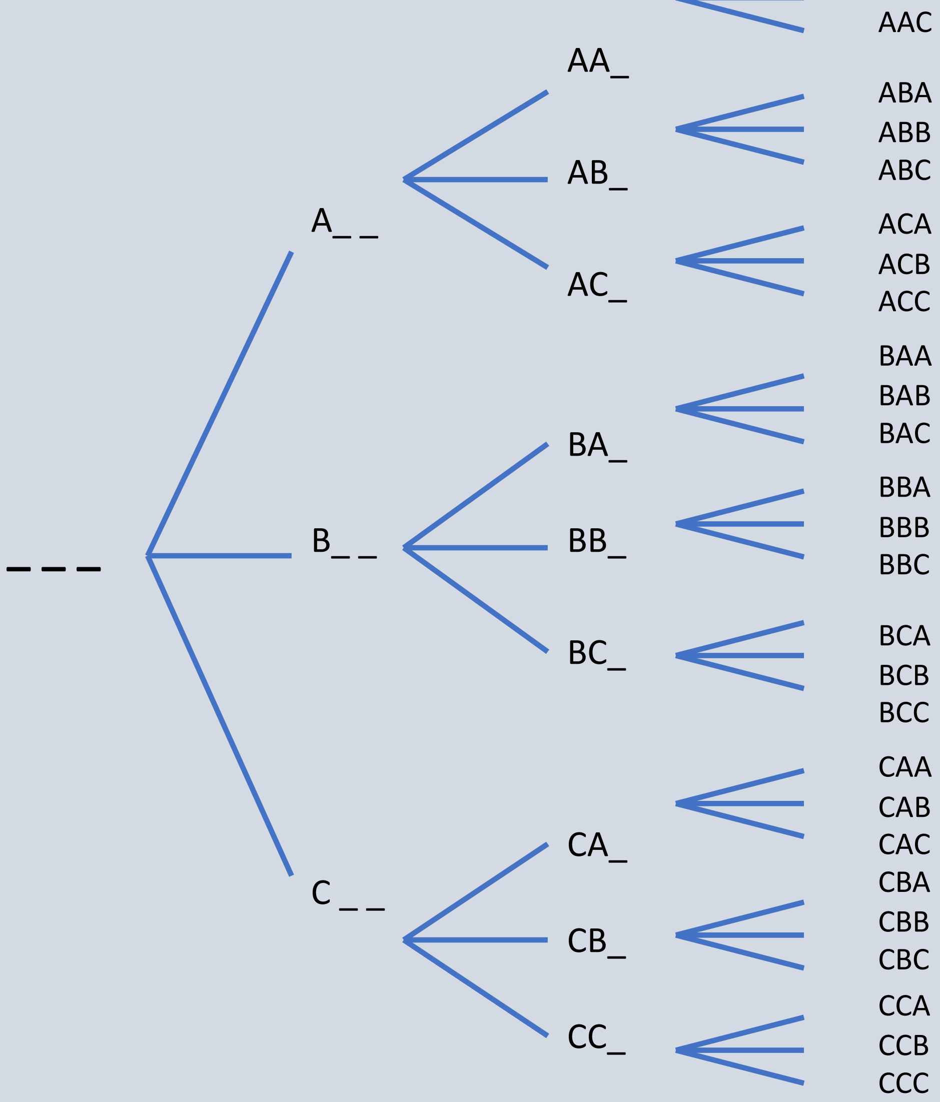
Counting Processes and Sets of Outcomes

- How many 3 letter passwords are there using the letters A, B, C (repetition allowed)?

$$\underline{\ 3\ } \cdot \underline{\ 3\ } \cdot \underline{\ 3\ } = 3^3$$

How does this counting process relate to sets of outcomes?

Tree Diagram



Set of Outcomes

AAA	AAB	AAC
ABA	ABB	ABC
ACA	ACB	ACC
BAA	BAB	BAC
BBA	BBB	BBC
BCA	BCB	BCC
CAA	CAB	CAC
CBA	CBB	CBC
CCA	CCB	CCC

A Different Structure on the Set of Outcomes

AAA	AAB	AAC
ABA	ABB	ABC
ACA	ACB	ACC
BAA	BAB	BAC
BBA	BBB	BBC
BCA	BCB	BCC
CAA	CAB	CAC
CBA	CBB	CBC
CCA	CCB	CCC

A Different Structure on the Set of Outcomes

AAA BBB CCC

	AAB	AAC
ABA	ABB	ABC
ACA	ACB	ACC
BAA	BAB	BAC
BBA		BBC
BCA	BCB	BCC
CAA	CAB	CAC
CBA	CBB	CBC
CCA	CCB	

A Different Structure on the Set of Outcomes

AAA BBB CCC

AAB, BBA ABA, BAB BAA, ABB

AAC, ABC, ACC ACA ACB

BAC, BBC, BCC BCA BCB

CAA, CBA, CCA CAB, CBB, CCB CAC, CBC

A Different Structure on the Set of Outcomes

AAA	BBB	CCC
AAB BBA	ABA BAB	BAA ABB
AAC CCA	ACA CAC	CAA ACC

		ABC
	ACB	
		BAC BBC
BCA	BCB	BCC
	CAB	
CBA	CBB	CBC
	CCB	

A Different Structure on the Set of Outcomes

AAA	BBB	CCC
AAB BBA	ABA BAB	BAA ABB
AAC CCA	ACA CAC	CAA ACC
BBC CCB	BCB CBC	CBB BCC

ABC

ACB

BAC

BCA

CAB

CBA

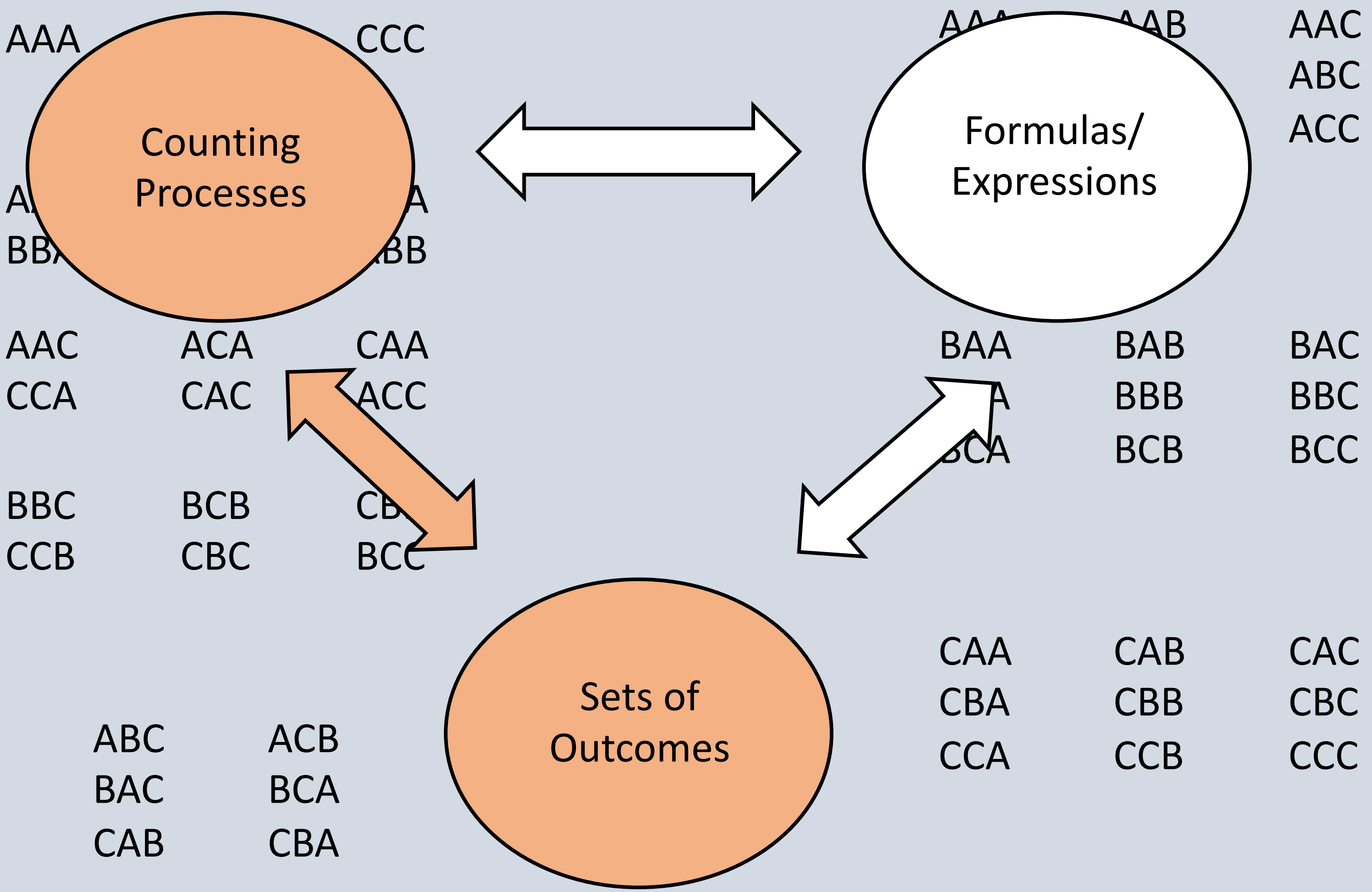
A Different Structure on the Set of Outcomes

AAA	BBB	CCC

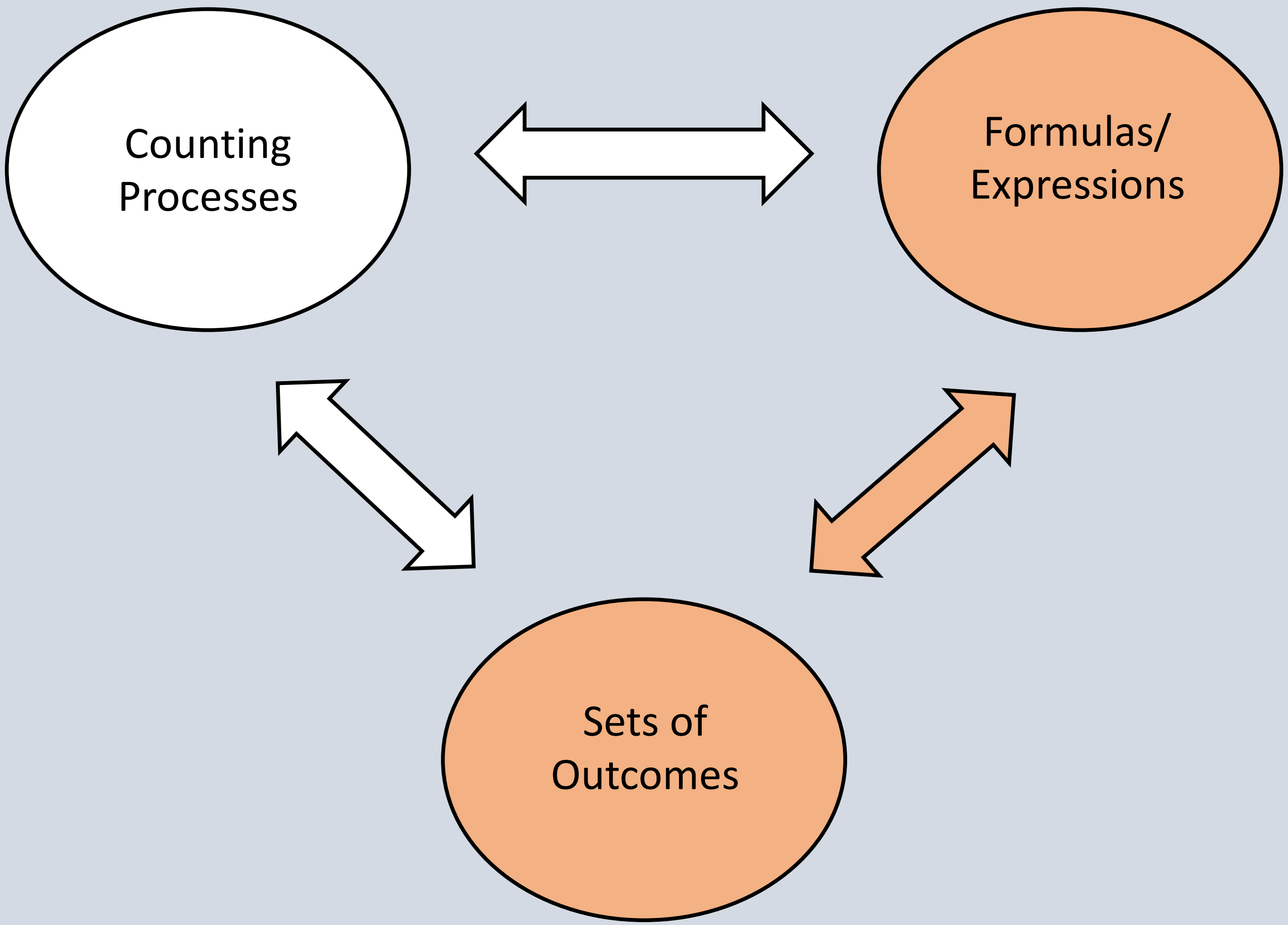
AAB	ABA	BAA
BBA	BAB	ABB
AAC	ACA	CAA
CCA	CAC	ACC
BBC	BCB	CBB
CCB	CBC	BCC

ABC	ACB
BAC	BCA
CAB	CBA

A Different Structure on the Set of Outcomes



Sets of Outcomes and Formulas/Expressions

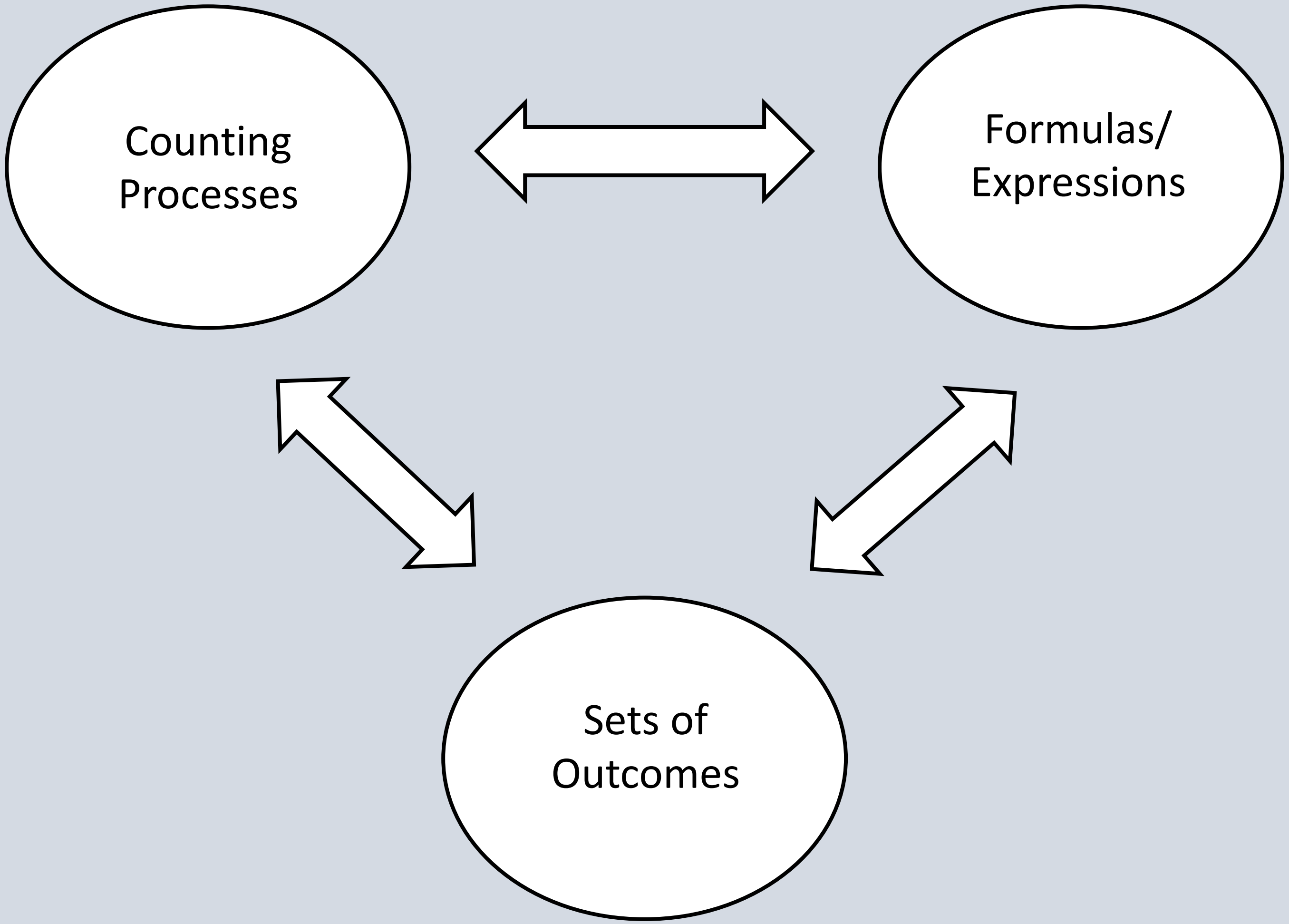


Sets of Outcomes and Formulas/Expressions

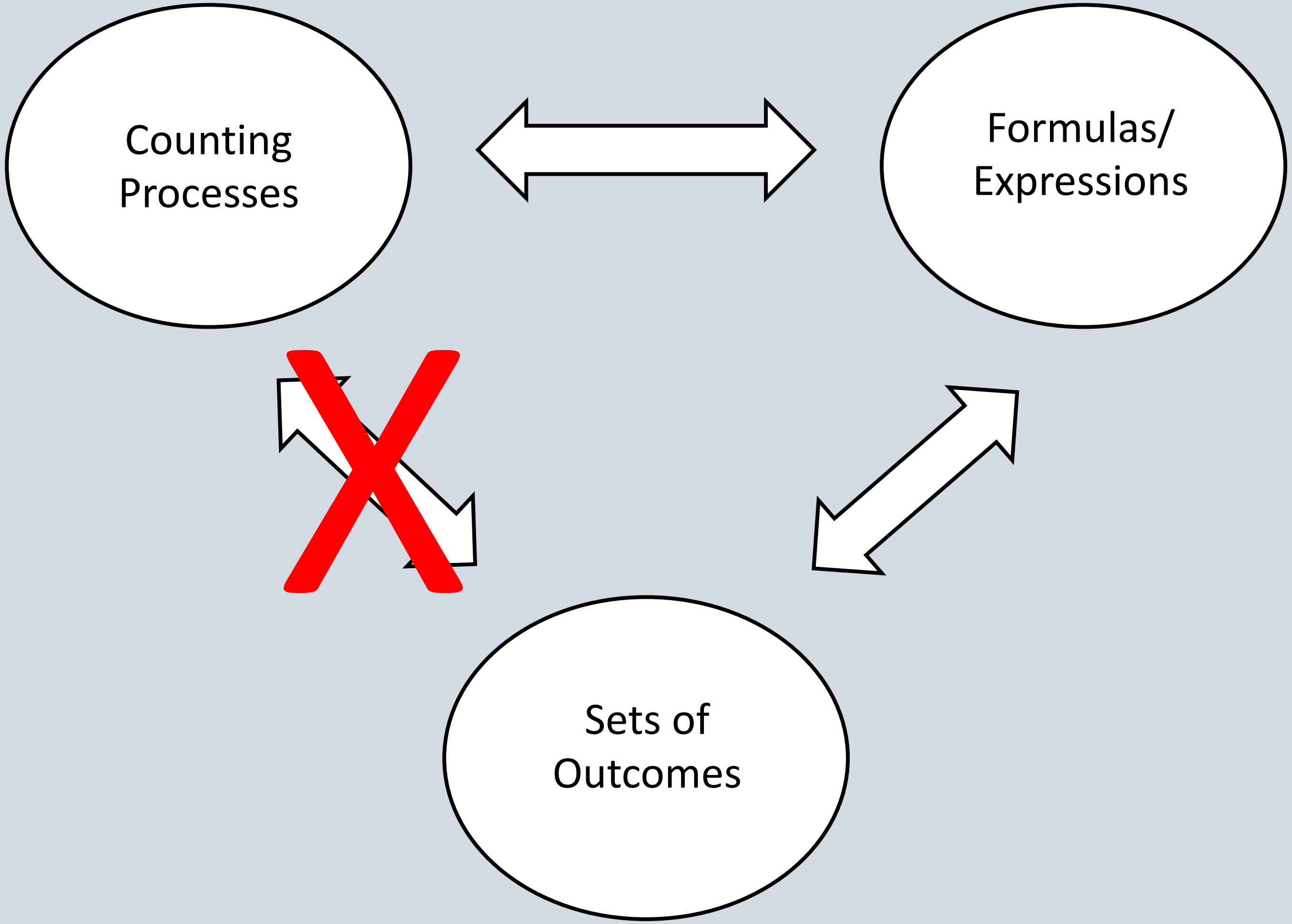
- How many ways are there to arrange *n* objects?

n	Outcomes	Number of outcomes
1	1	1
2	12 21	2 = 1*2
3	123 213 312 132 231 321	6 = 1*2*3
4	1234 2134 3124 4123 1243 2143 3142 4132 1324 2314 3214 4213 1342 2341 3241 4231 1423 2413 3412 4312 1432 2431 3421 4321	24 = 1*2*3*4

A Model of Students' Combinatorial Thinking



A Model of Students' Combinatorial Thinking



Let's revisit the 3-letter sequence problem

How many 3-letter sequences can be made using the letters *a, b, c, d, e, f,*

1. If **no repetition** of letters is allowed?

    # = 6 x 5 x 4

__ __ __

Let’s revisit the 3-letter sequence problem

How many 3-letter sequences can be made using the letters *a, b, c, d, e, f,*

1. If **no repetition** of letters is allowed?
2. If the sequence **must contain** *e*, and **no repetition** of letters is allowed?

   # = 3 x 5 x 4

e ___ ___

___ e ___

___ ___ e

Let's revisit the 3-letter sequence problem

How many 3-letter sequences can be made using the letters *a, b, c, d, e, f,*

1. If **no repetition** of letters is allowed?
2. If the sequence **must contain** *e*, and **no repetition** of letters is allowed?
3. If the sequence **must contain** *e*, and **repetition** of letters **is** allowed?

   # = 3 x 6 x 6

e ___ ___ ___

___ e ___ ___

___ ___ e ___

Let’s revisit the 3-letter sequence problem

- Only reasoning about the counting process (and not the set of outcomes) can be misleading
  - 3 x 6 x 6 overcounts
  - Consider two ways of completing the process

e a e

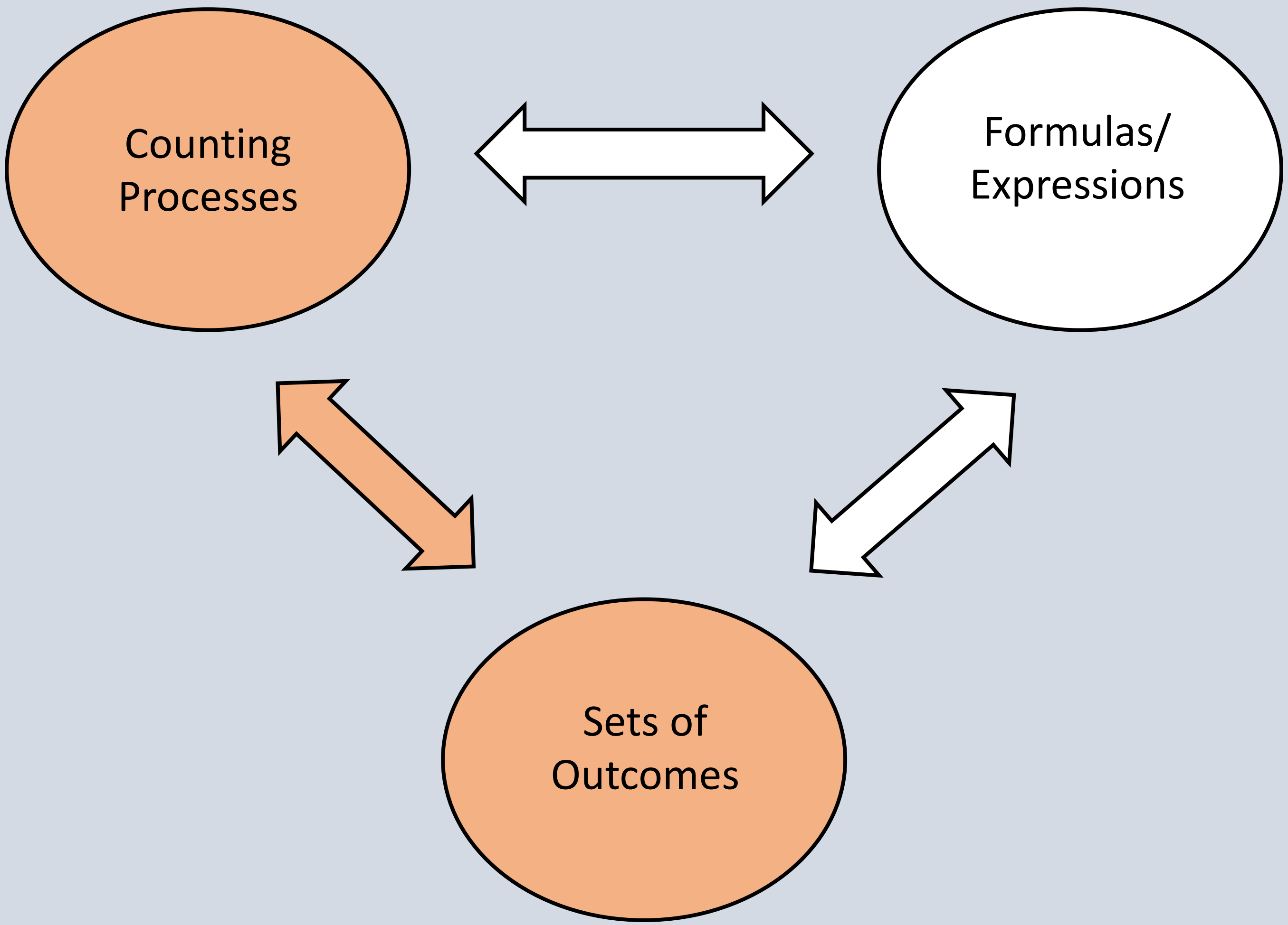
e a e

- The “eae” sequence is counted too many times

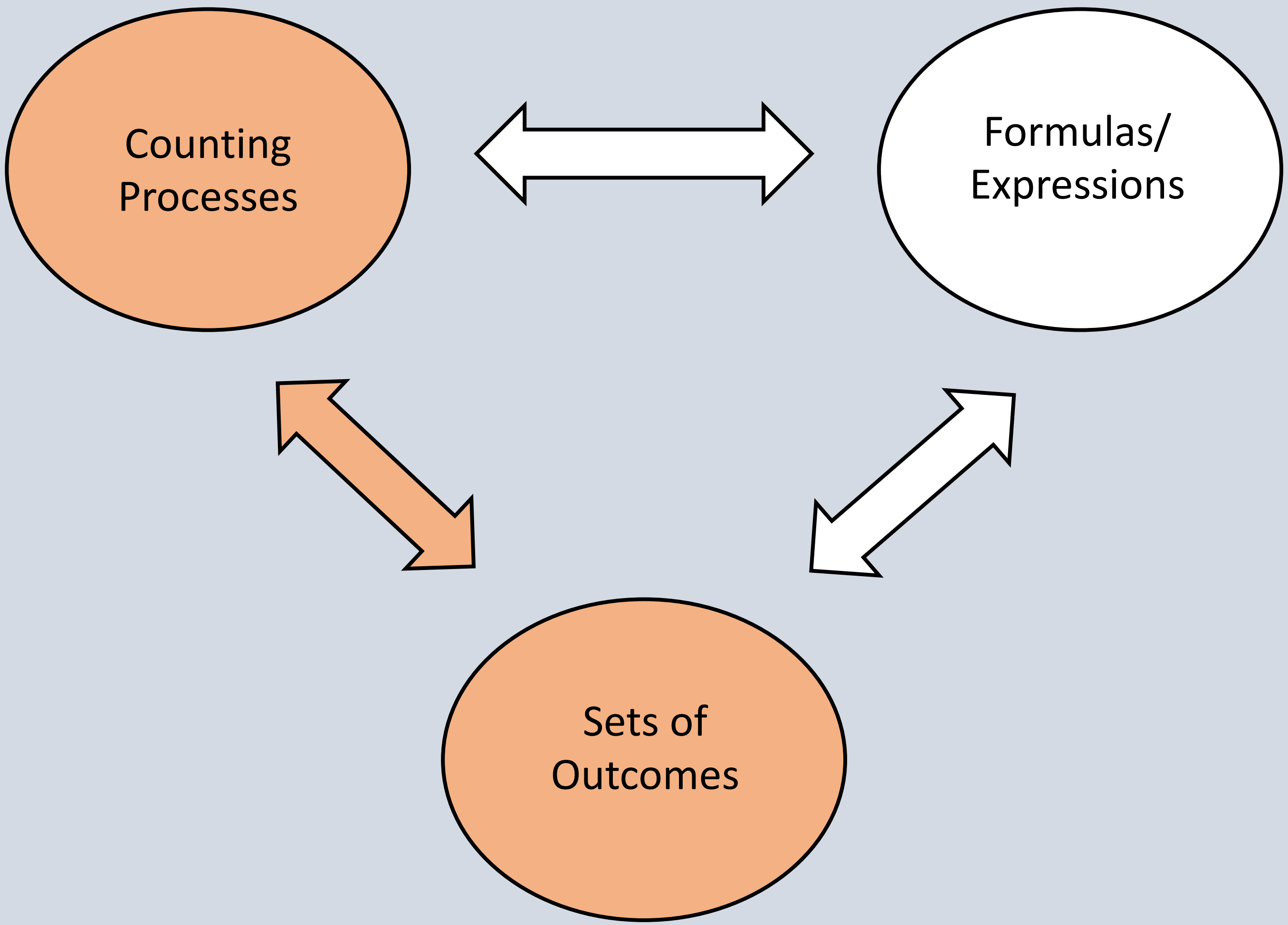
Let’s revisit the 3-letter sequence problem

- There are 3 x 6 x 6 = 108 ways of completing the counting process
- This is true, but this is not equal to the number of distinguishable desirable outcomes
- We have to refine the counting process (the answer is actually 91)
- For me, this offers motivation to help students reason about the relationship between counting processes and sets of outcomes

This relationship is very important!



Leveraging Computing to Reinforcing this Relationship



Leveraging Computing to Reinforcing this Relationship

- When programming, students MUST explicitly articulate a counting process, and they get immediate feedback on the output of that process
- Lockwood and Gibson (2016) found that even partially systematically listing outcomes was positively correlated with students answering counting problems correctly
  - Systematic listing makes a connection between students' counting processes and their sets of outcomes
  - Listing is a combinatorial activity that we value, but it is often infeasible to do by hand

Summary of Motivation

- Based on this model and the desire to reinforce the relationship between counting processes and sets of outcomes, I turn to computing
- My hypothesis is that computing can help students reinforce this important relationship
- We do not know if computing helps or hurts students as they reason about and solve counting problems
- I am motivated primarily to use computing to improve student success in solving combinatorial problems, but I also want to better understand the nature of computational thinking and activity in mathematics

Research Goals and Questions

1. Broadly, how does engaging in computational thinking and activity help students learn mathematical or scientific content?
2. In what ways does engaging in basic Python programming tasks
   a. Help to solidify/reinforce the relationship between counting processes and sets of outcomes?
   b. Help students successfully solve counting problems?
3. **What are affordances and limitations of having students solve counting problems in a computational setting?**

Methods: Teaching Experiments

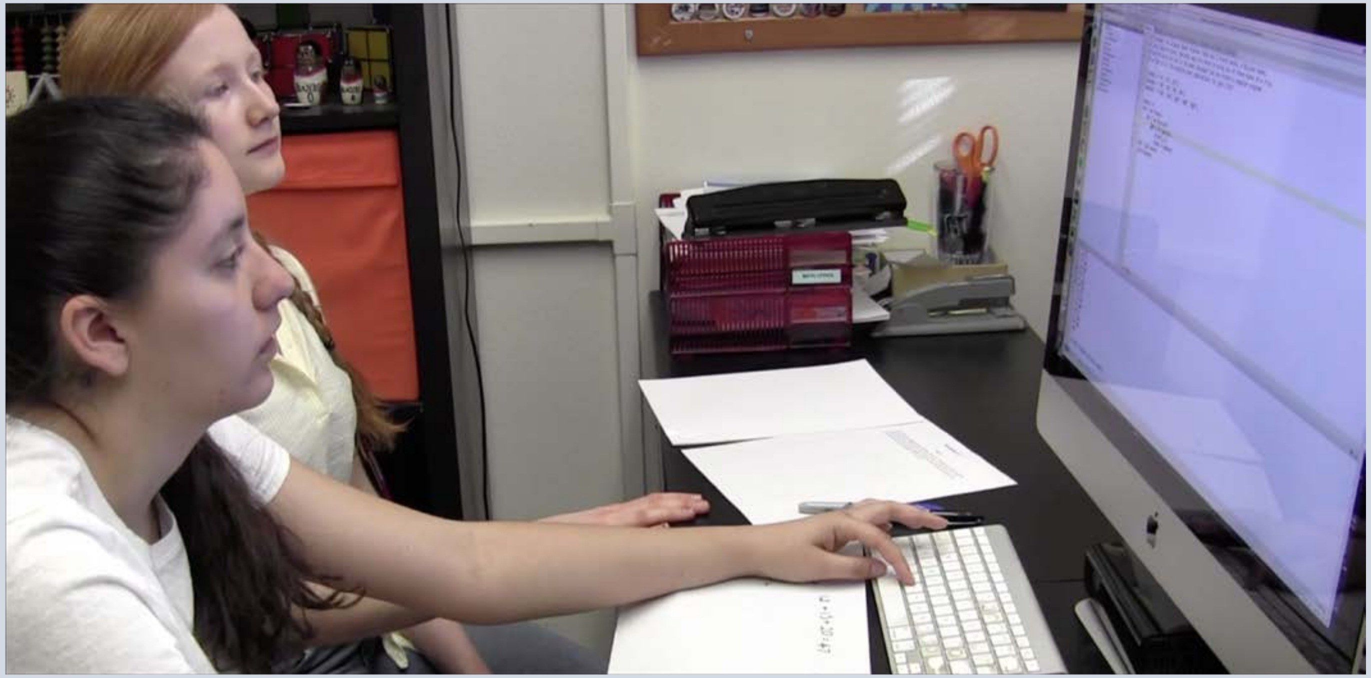
- A main purpose for teaching experiments is “for researchers to experience, firsthand, students’ mathematical learning and reasoning” (Steffe & Thompson, 2000, p. 267)
- The teaching experiment methodology allows for a researcher to explore students’ reasoning over a period of time and to observe how they think about and learn particular mathematical concepts
- Last spring I interviewed three pairs of undergraduate students in paired teaching experiments (two pairs for ~15 hours, one pair for ~10 hours)

Methods: Participants

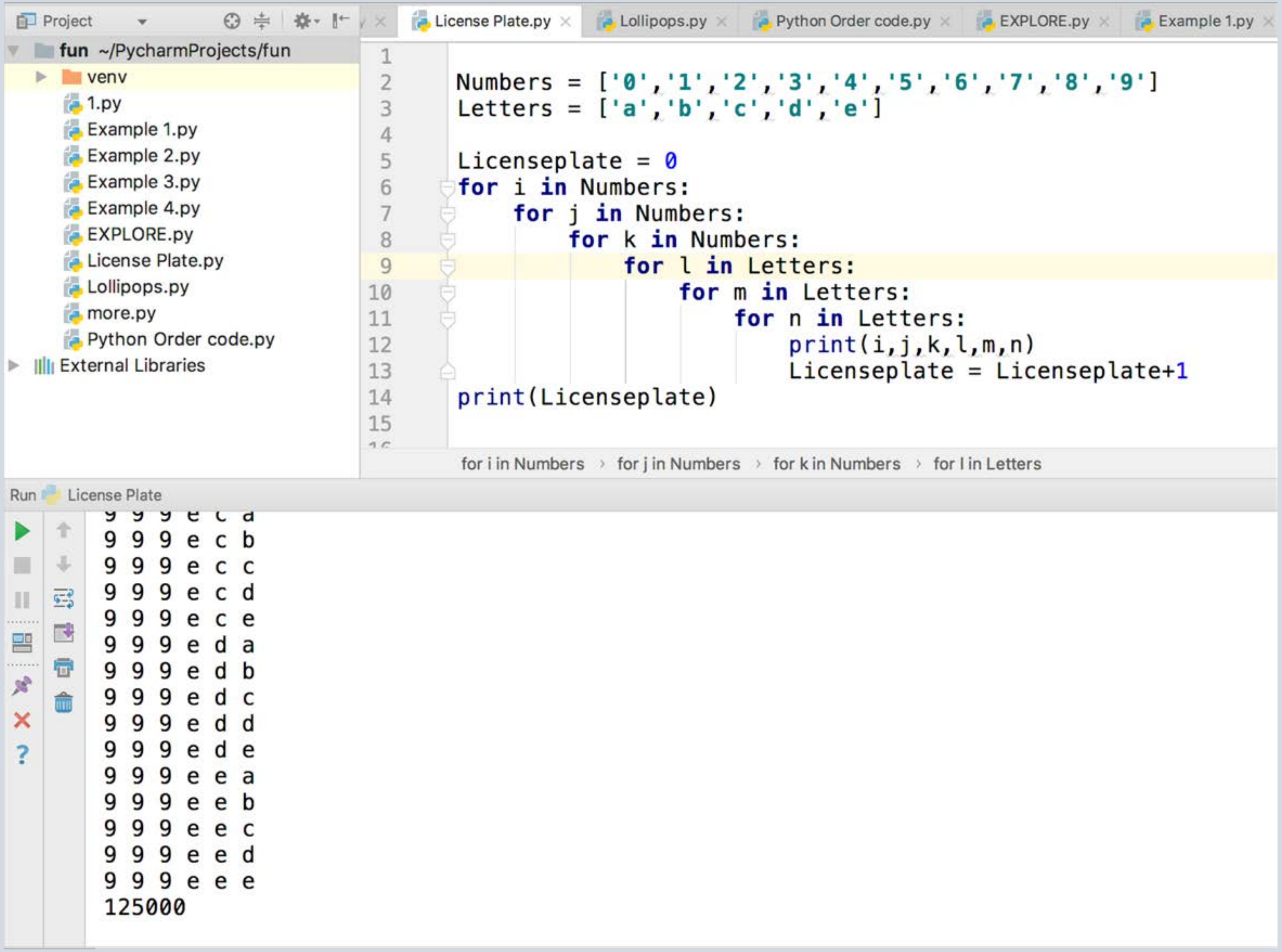
- I present data from one of the pairs of students, who were recruited from vector calculus
- Charlotte and Diana
  - Sophomore and freshman chemistry majors
  - Novice counters (had not taken counting in college)
  - No programming experience in high school or in college

Methods: Data Collection

- The students sat together and worked at a computer, coding in the environment PyCharm
- I videotaped the interviews and recorded the screen



Methods: Data Collection



Methods: Tasks

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

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

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

Can you write some code in order to create a list of all of the ways to arrange 3 of the letters in the word ROCKET?

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?

Methods: Tasks

- I would regularly ask the students to
  - Predict the output of their code and/or the total number of outcomes
  - Discuss the overall structure of the outcomes
  - Write an expression for the total number of outcomes
- Students incorporated both by-hand work and computer work
  - Sometimes they used programming to solve the problem (or to get a start on the problem)
  - Sometimes they solved the problem by hand and then coded the solution

Your Turn! Consider this Python Code

```
Numbers = [1, 2, 3, 4, 5, 6]

Counter = 0

for i in Numbers:
    for j in Numbers:
        if j != i:
            for k in Numbers:
                if k != i and k != j:
                    print(i, j, k)
                    Counter = Counter + 1
print(Counter)
```

- What are the first few outcomes going to look like? Why?
- How will the entire set of outcomes be structured or organized?
- What is an expression that represents the total number of outcomes?

Your Turn! Consider this Python Code

```python
Numbers = [1, 2, 3, 4, 5, 6]

Counter = 0

for i in Numbers:
    for j in Numbers:
        if j != i:
            for k in Numbers:
                if k != i and k != j:
                    print(i, j, k)
                    Counter = Counter + 1
print(Counter)
```

- There are 6 x 5 x 4 = 120 total outcomes
- The set of outcomes is organized into 6 large chunks of 20

Methods: Data Analysis

- All interviews were transcribed, and we created enhanced transcripts
- The research team used qualitative analysis software MaxQDA to examine the data and identify affordances
- For the results presented in this talk, I focused on episodes that
  - Highlighted the nature of the interviews
  - Demonstrated affordances
  - Offered examples of computational thinking

Results

- I will discuss four affordances of solving combinatorics problems in a computational setting
    1. Students get **immediate feedback** on the output of their code
        - Experimentation, verification, and justification
    2. Students can **survey entire sets of outcomes**
        - Identify patterns and mathematical structure
    3. Students get opportunities to **connect multiple mathematical representations**
        - Code, lists of outcomes, tree diagrams, mathematical expressions
    4. Students gain insight about **key distinctions** between combinatorial problems
- I hope to present examples of computational thinking

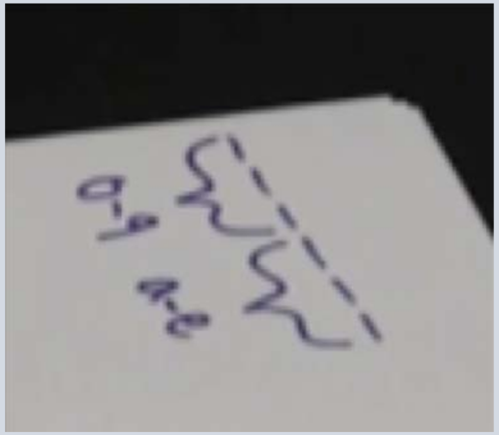
The License Plate Problem

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

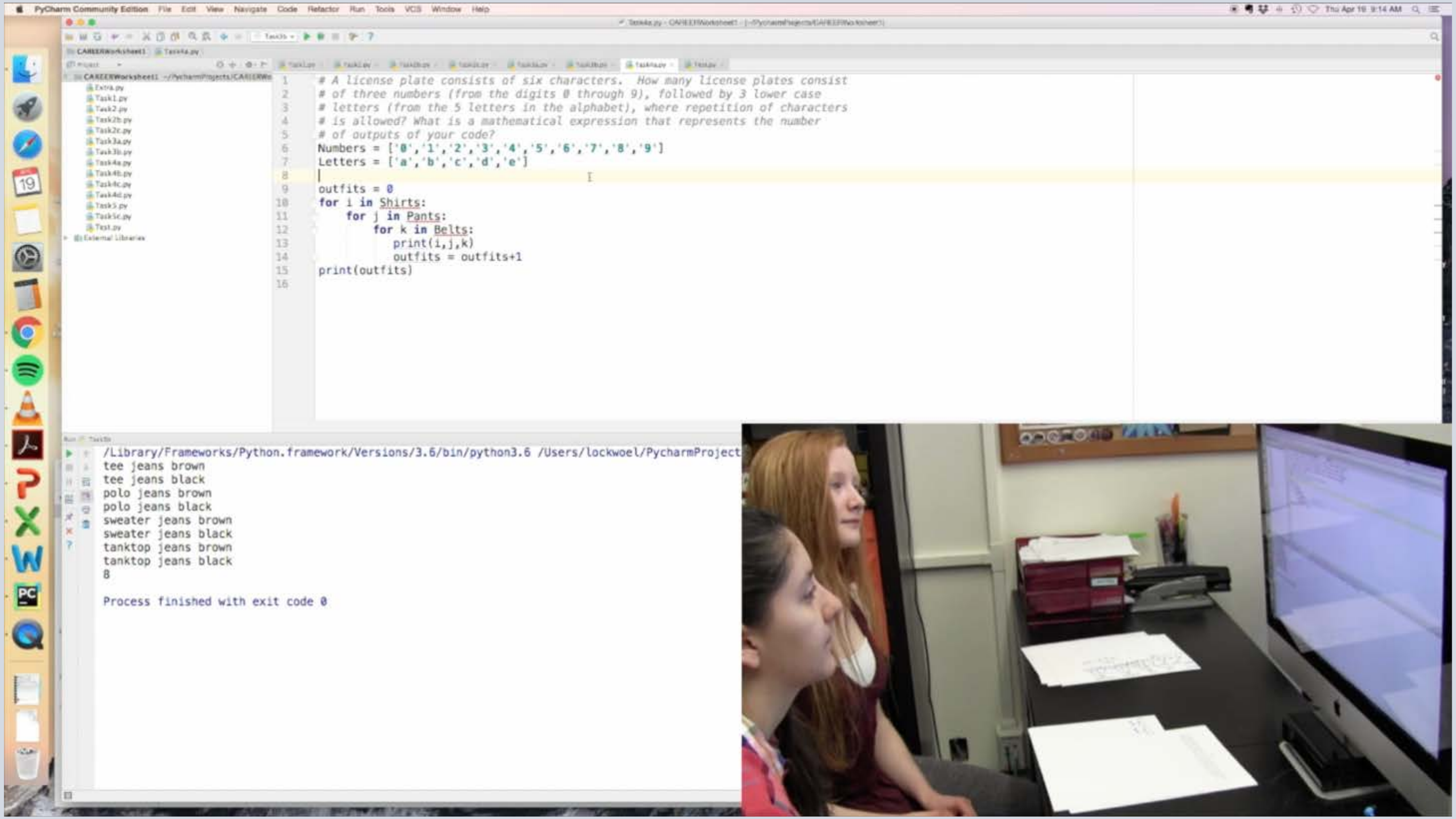
- The answer is 10 x 10 x 10 x 5 x 5 x 5 = 125,000

Charlotte and Diana’s work on the License Plate Problem

- *Charlotte*: Okay. So, we have six characters. We’ll just kind of just visualize it first…And we’ll say these three are sort of, zero to nine, and these three are –
- *Diana*: *a* to *e*?
- *Charlotte*: Yeah. *a* to *e*. Okay. So, write the code to solve this problem. What is the mathematical expression that represents the number of outcomes – Okay. Okay. Where do we start?

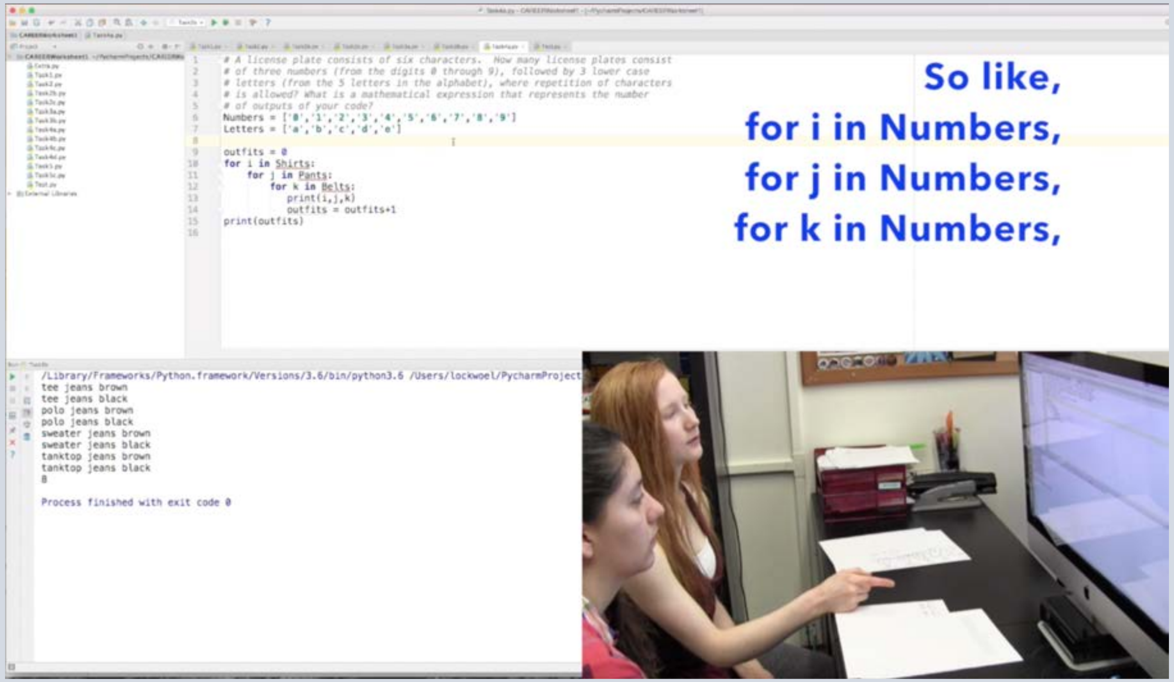


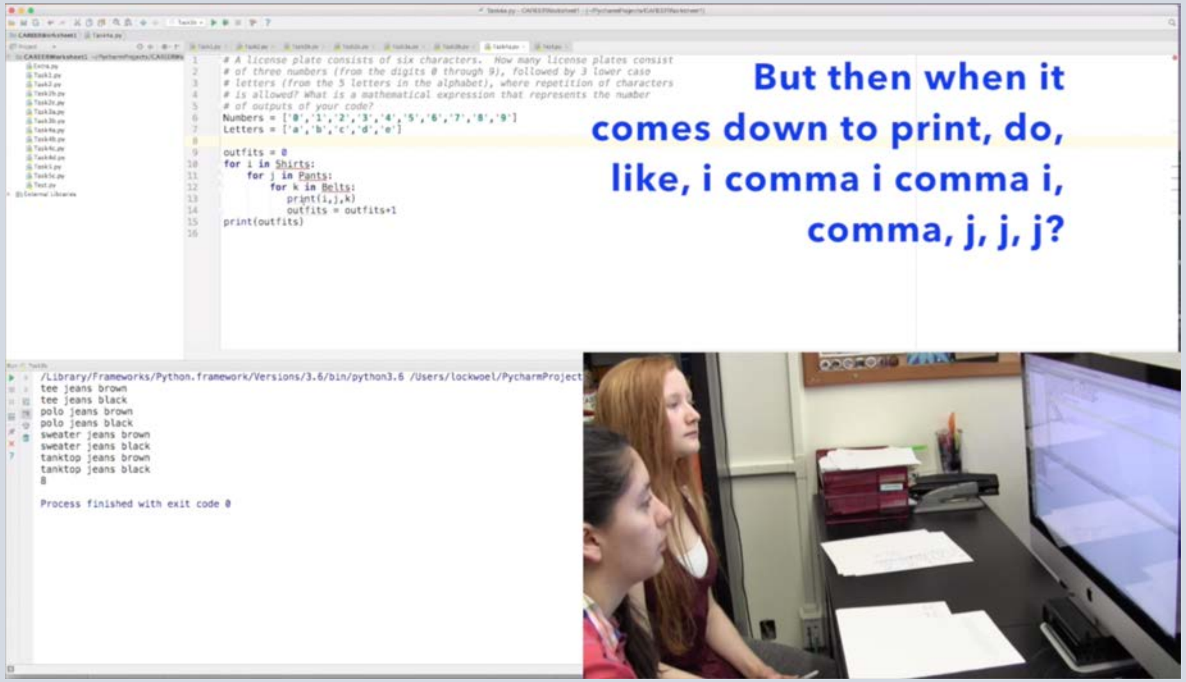
Charlotte and Diana’s work on the License Plate Problem



Affordances: Immediate feedback

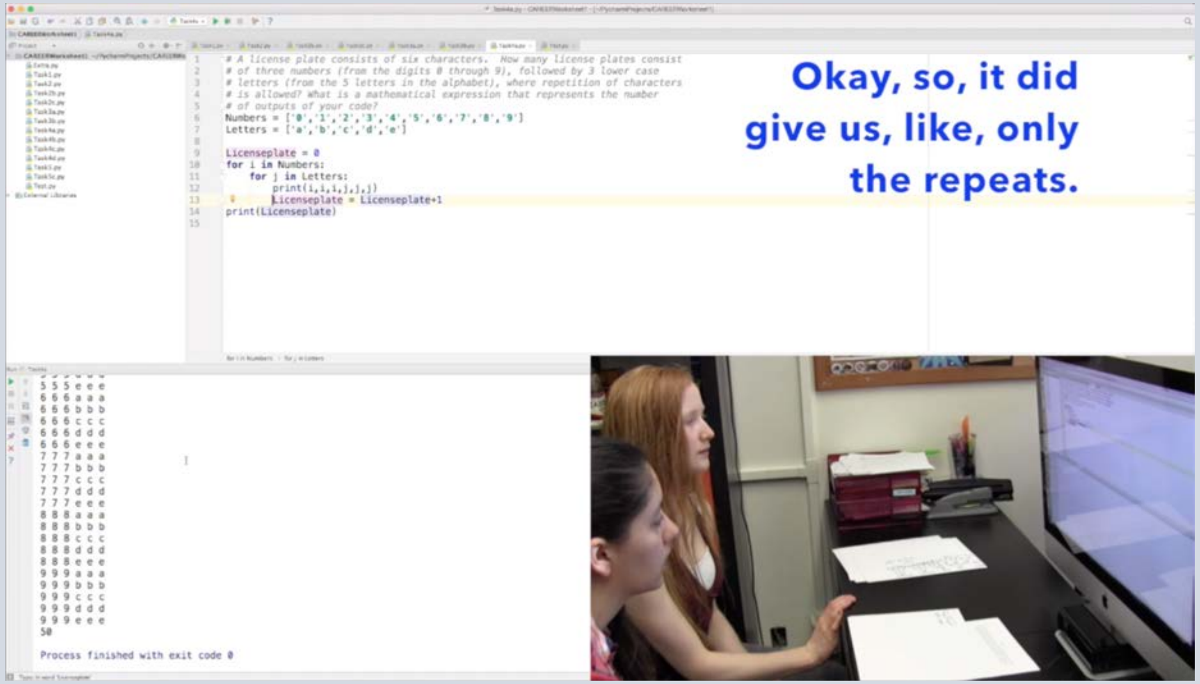
- The feature of the computer's **immediate feedback** facilitated **experimentation** with multiple strategies

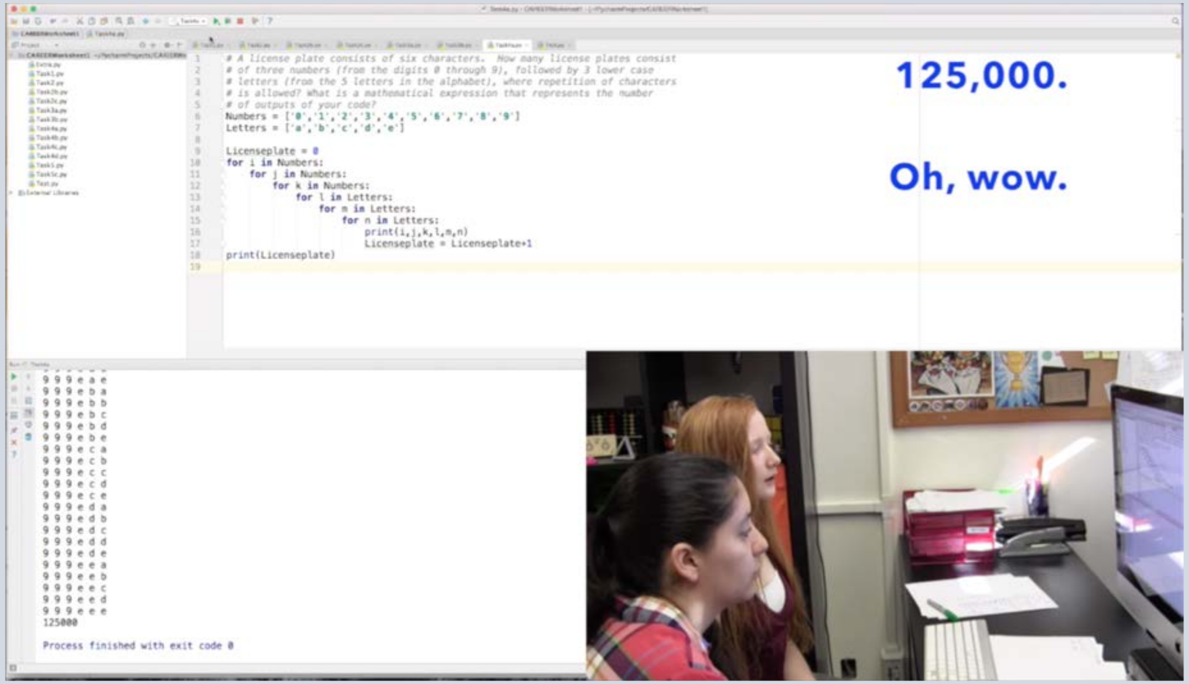




Affordances: Immediate feedback

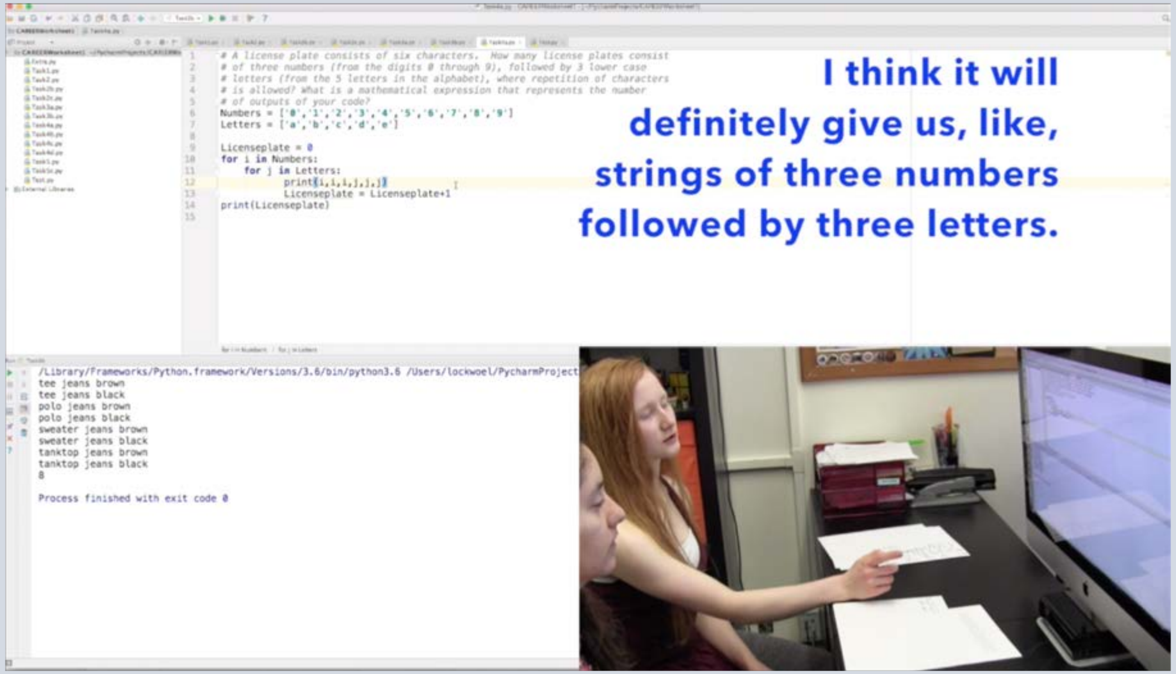
- The feature of the computer's **immediate feedback** provided structural and numerical **verification**

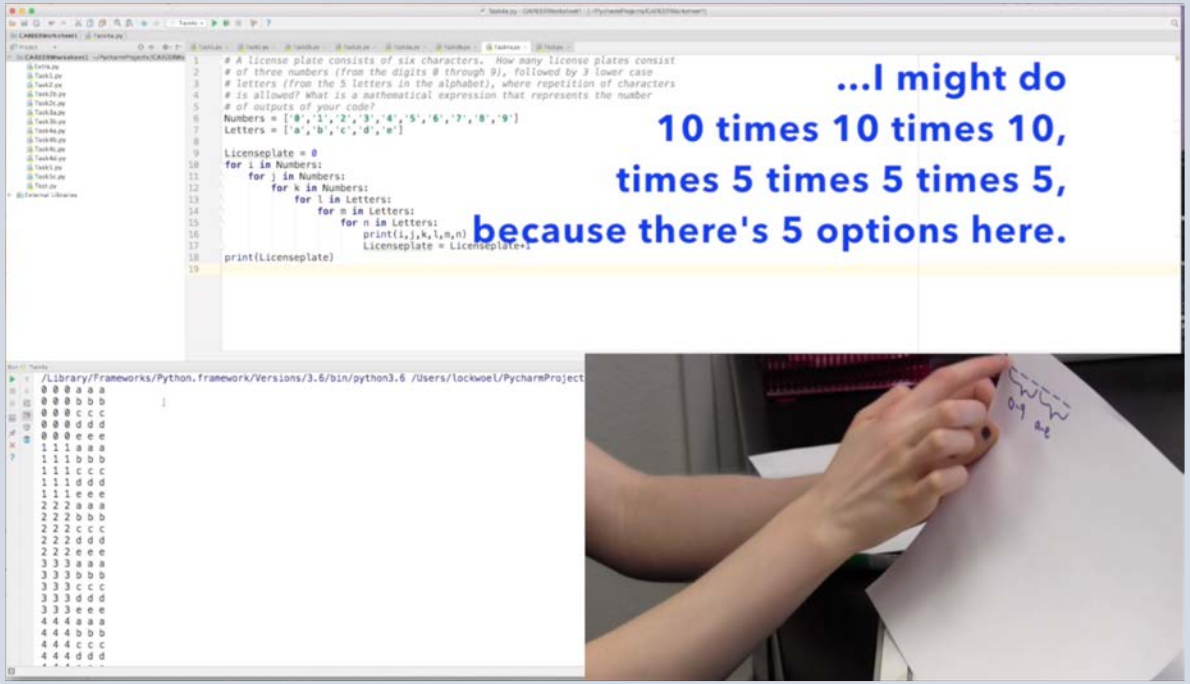




Affordances: Connecting Representations

- The computational setting allows for **connections between multiple mathematical representations**
- Code, lists of outcomes, mathematical expressions



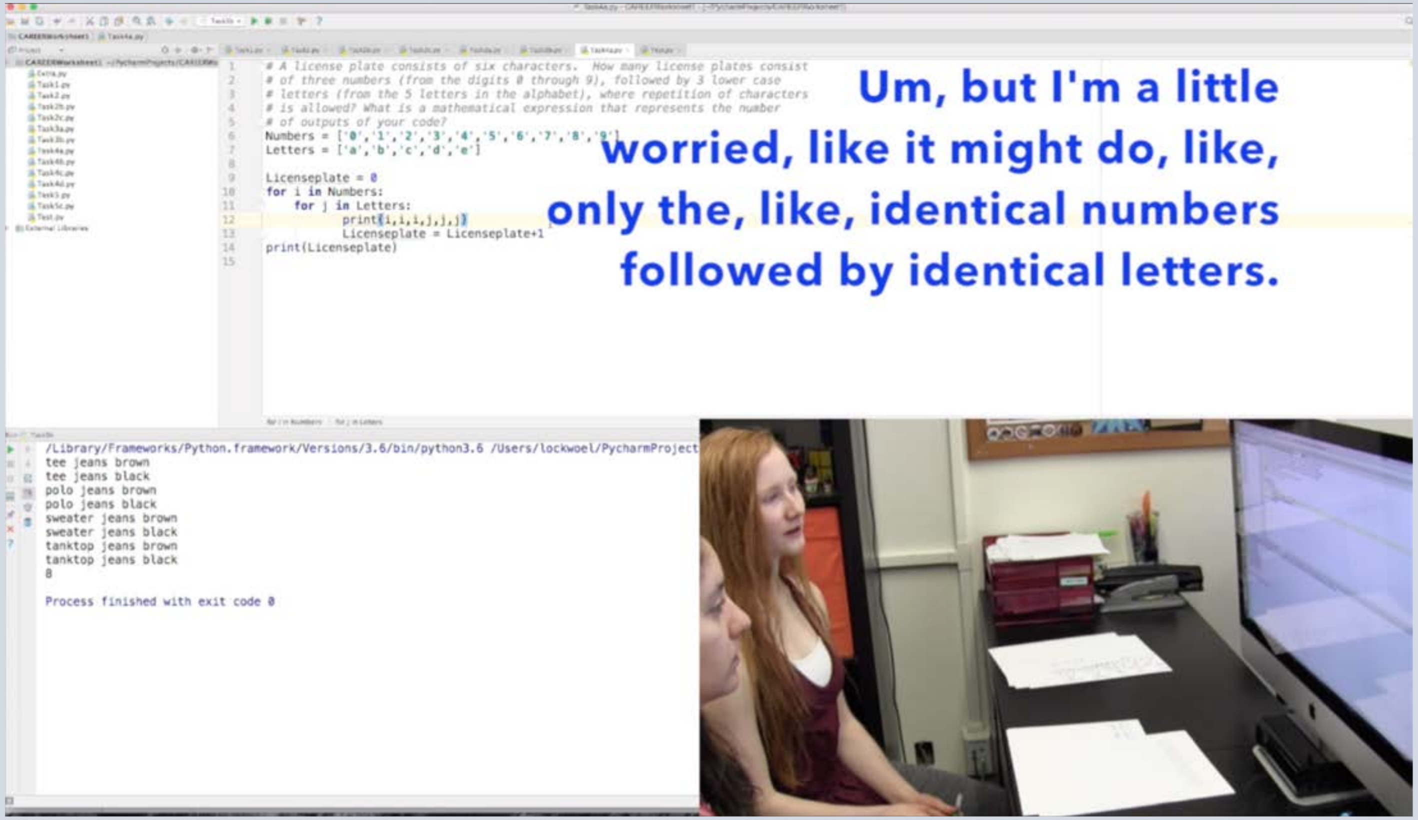


Affordances: Summary of this episode

1. The computational setting provides **immediate feedback** for the students to see the results of their programming, which
   - Facilitated **experimentation** of multiple strategies
   - Provided numerical and structural **verification**
2. The computational setting allows for **connections between multiple mathematical representations**

An example of Computational Thinking

- I suggest that this exchange is an example of computational thinking in mathematics



The Heads and Tails Problem

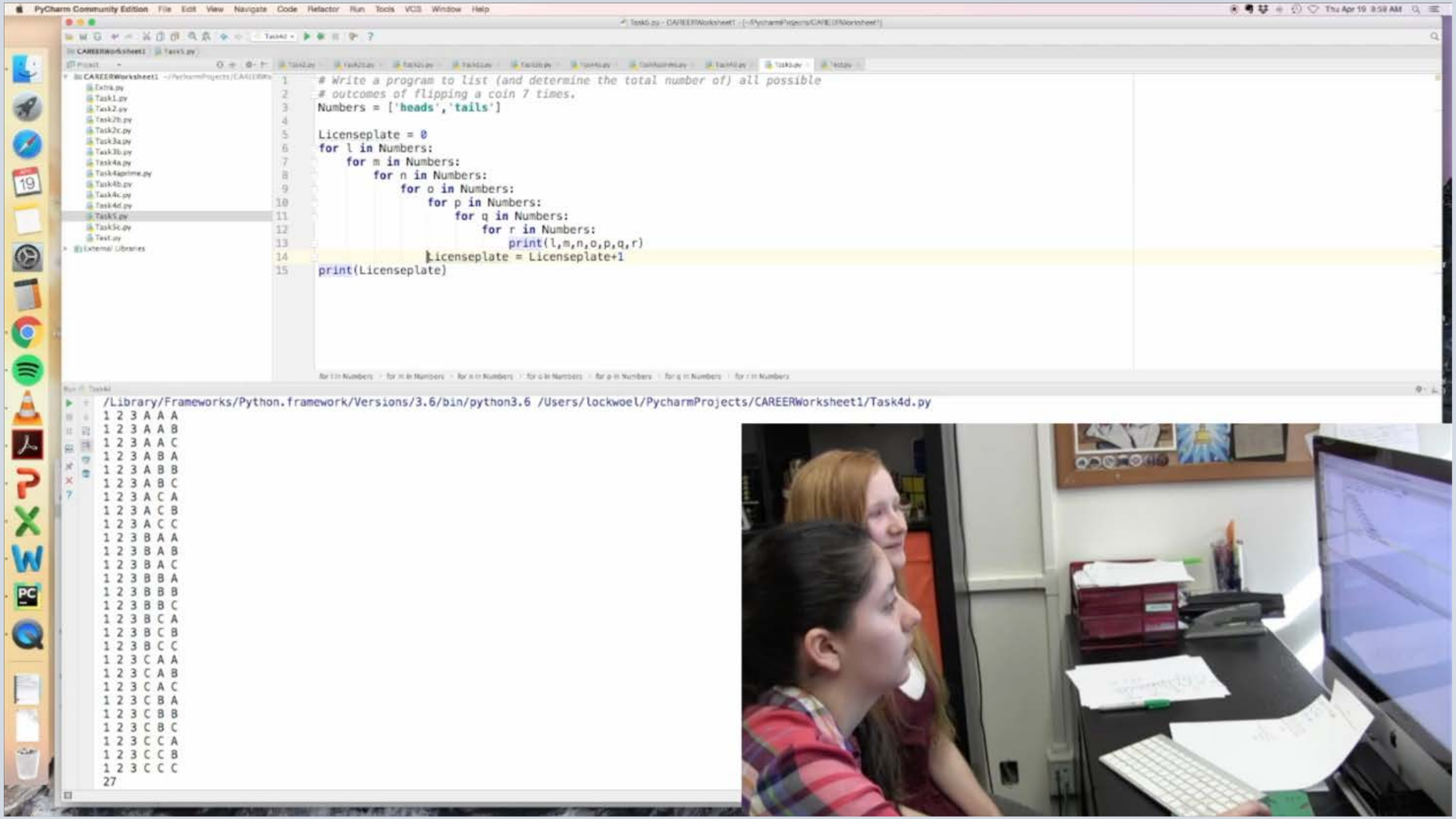
- *Write a program to list (and determine the total number of) all possible outcomes of flipping a coin 7 times.*

Charlotte and Diana’s Work on the Heads and Tails Problem

- *Write a program to list (and determine the total number of) all possible outcomes of flipping a coin 7 times.*

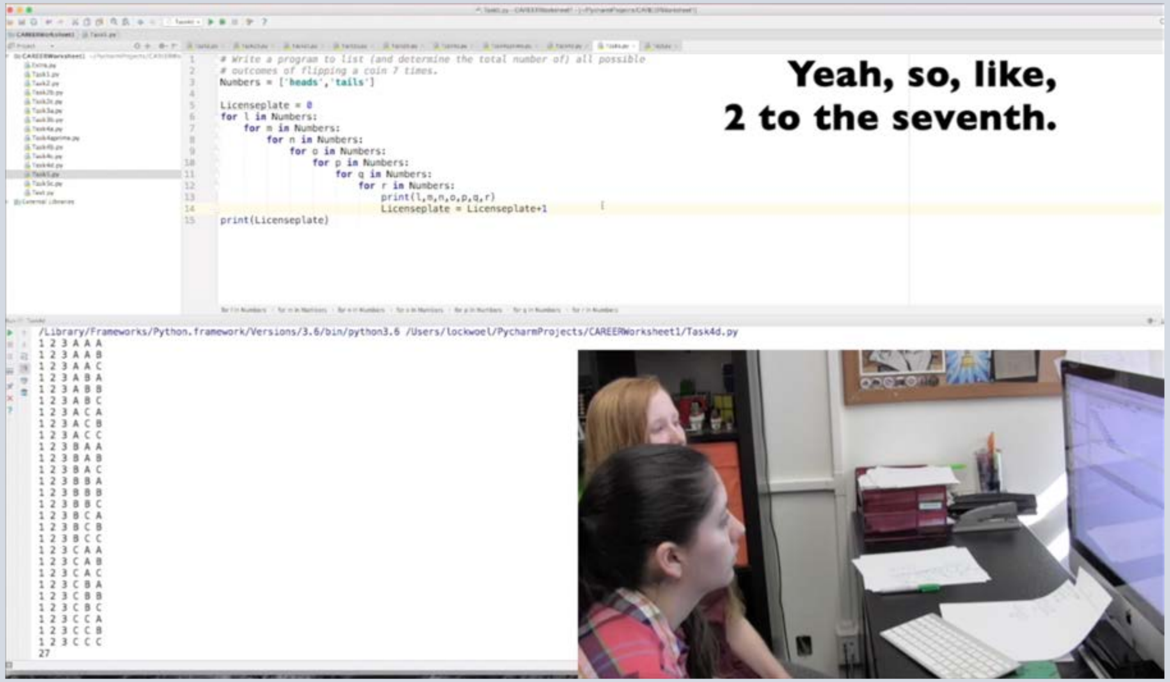
```
Numbers = ['heads','tails']

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

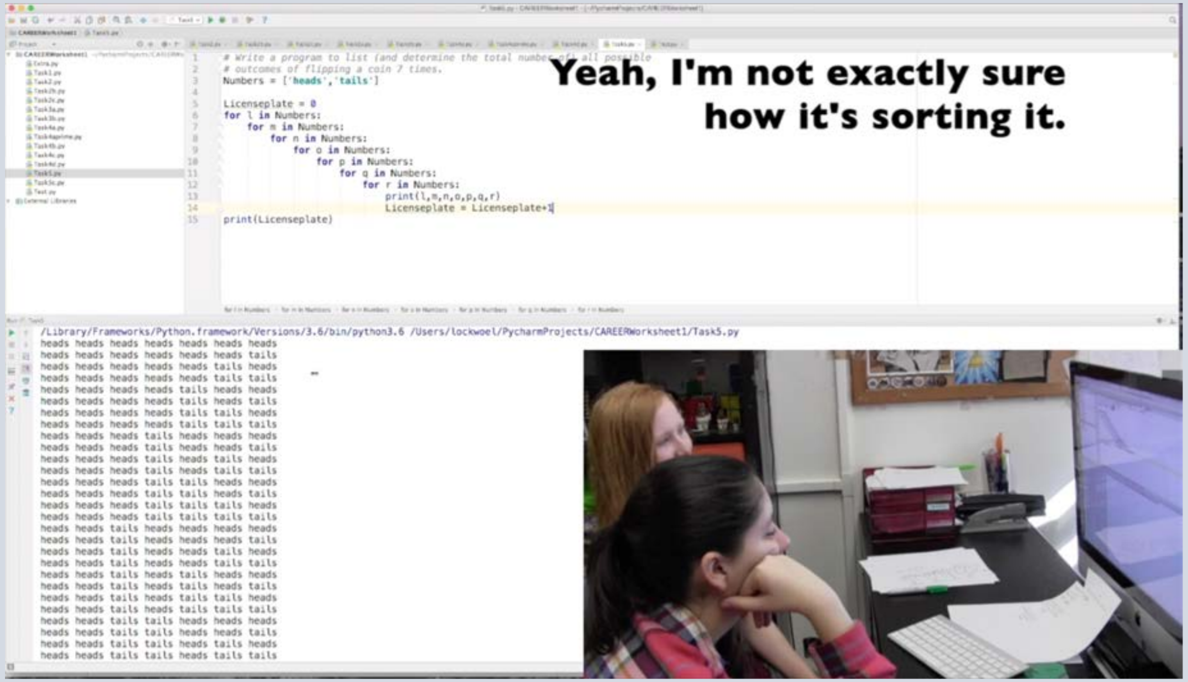
Charlotte and Diana’s Work on the Heads and Tails Problem



Affordance: Reinforce the relationship between counting processes and sets of outcomes

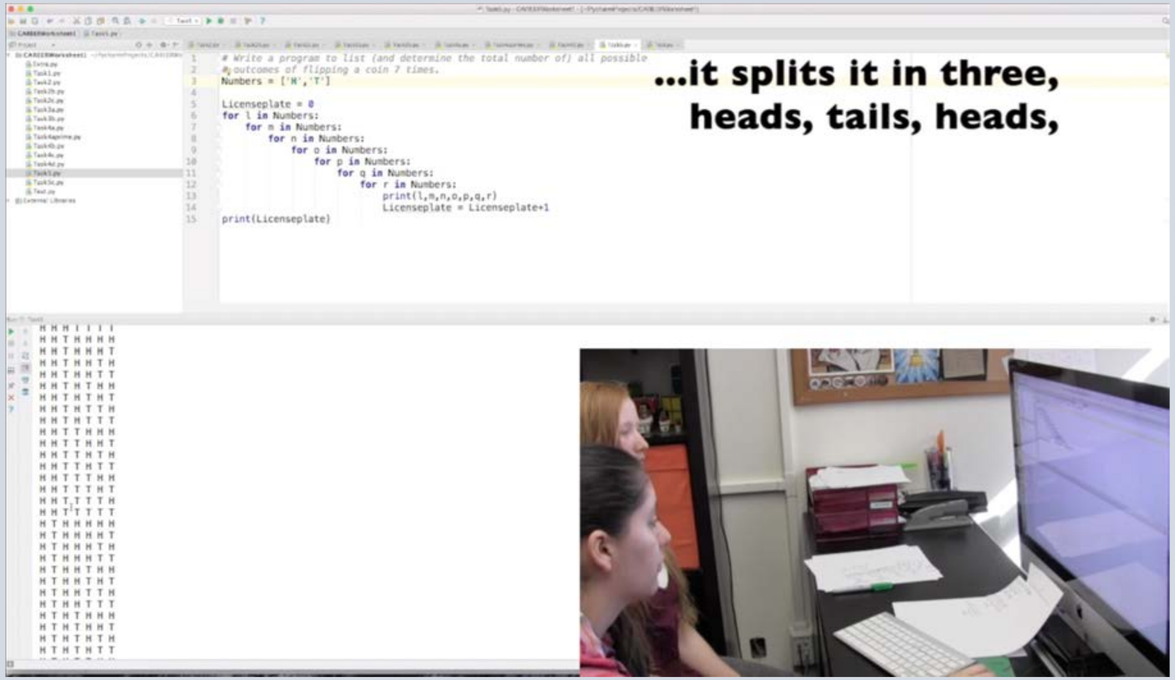
- The students initially got the correct answer, but there is evidence that they did not understand **how that process organized the set of outcomes.**

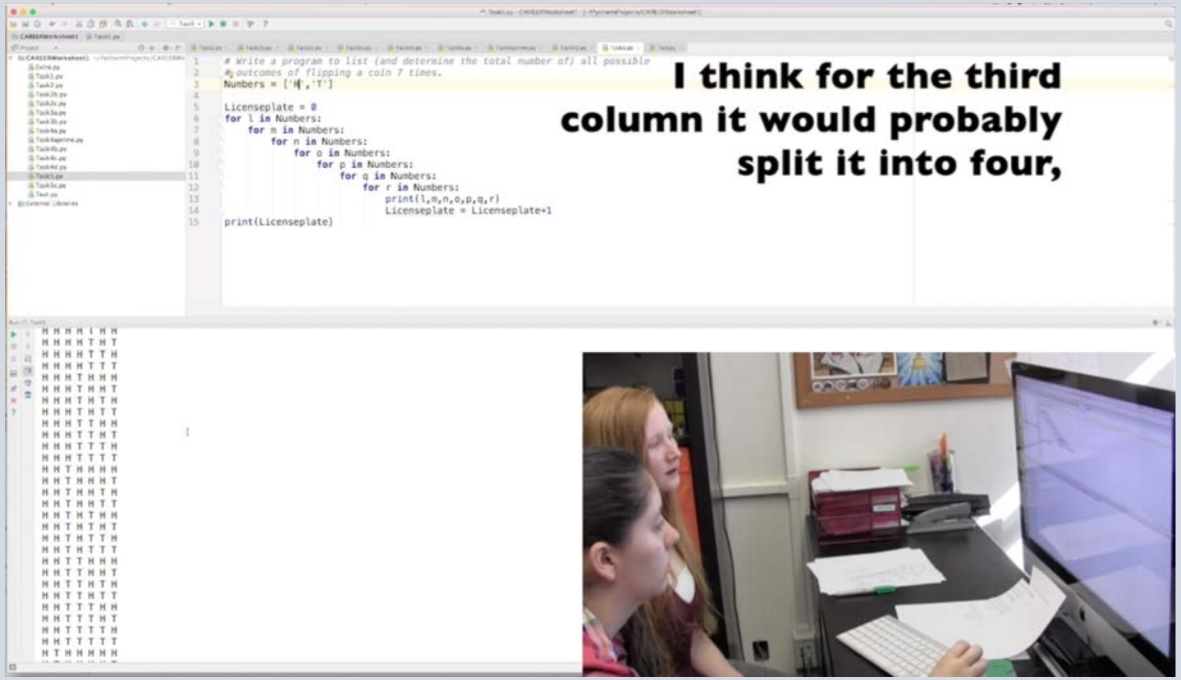




Affordances: Surveying Entire Set of Outcomes

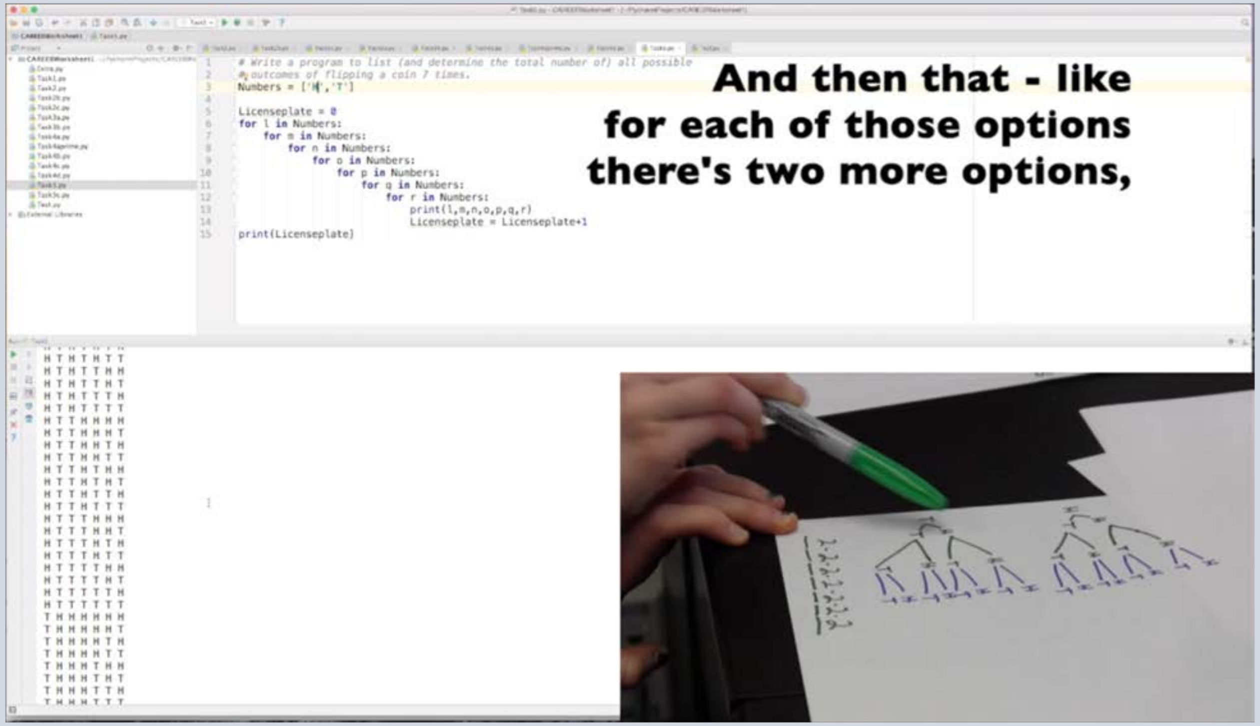
- By **surveying** and scrolling through the complete list of outcomes, they began to notice **structures and patterns** within the list
- They identified an important **mathematical property** in their list





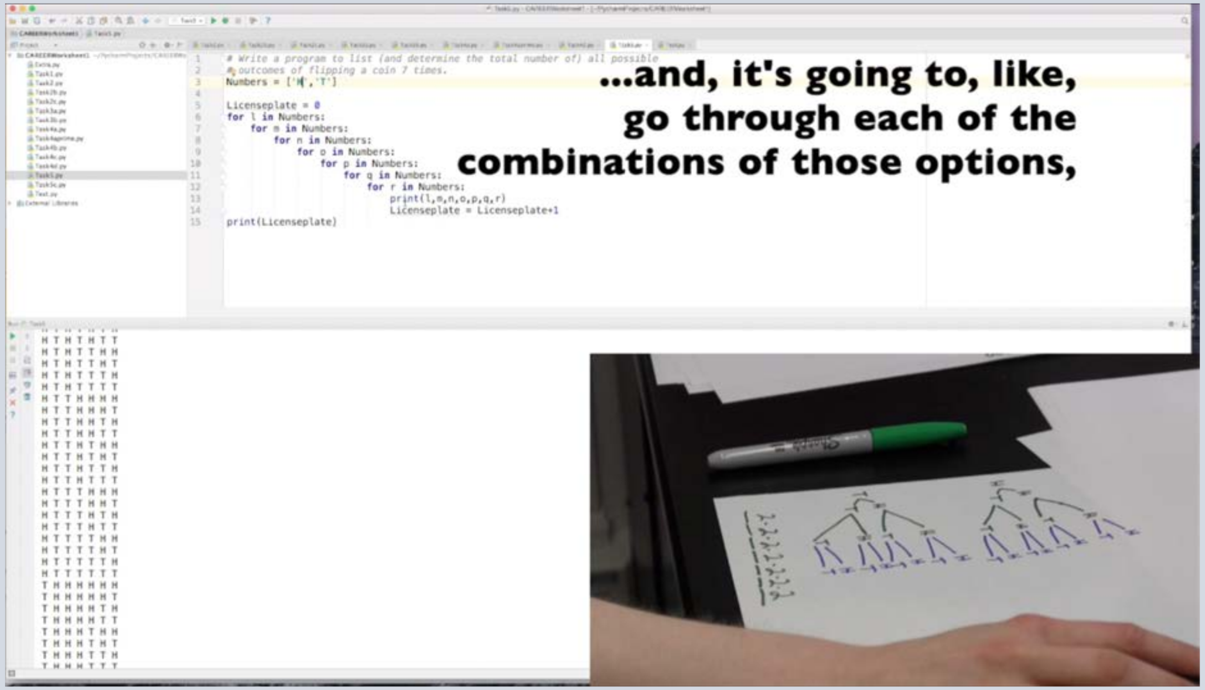
Affordances: Surveying Entire Set of Outcomes

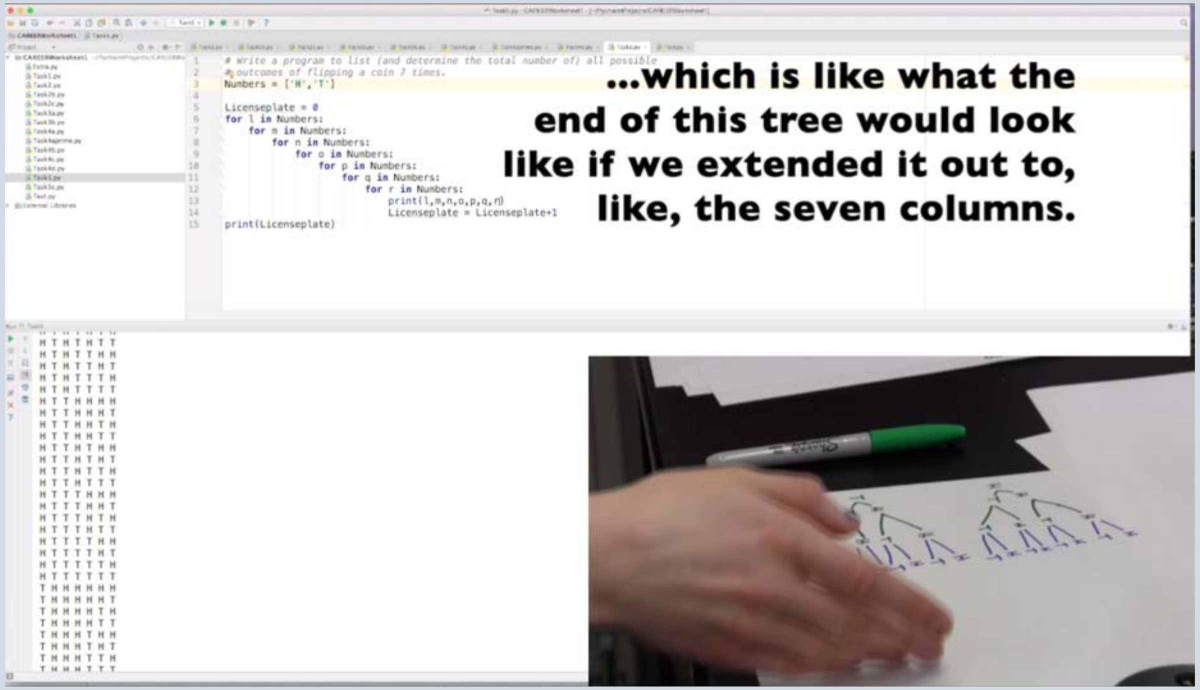
- By **surveying** and scrolling through the complete list of outcomes, they began to notice **structures and patterns** within the list
- They identified an important **mathematical property** in their list



Affordances: Connecting Representations

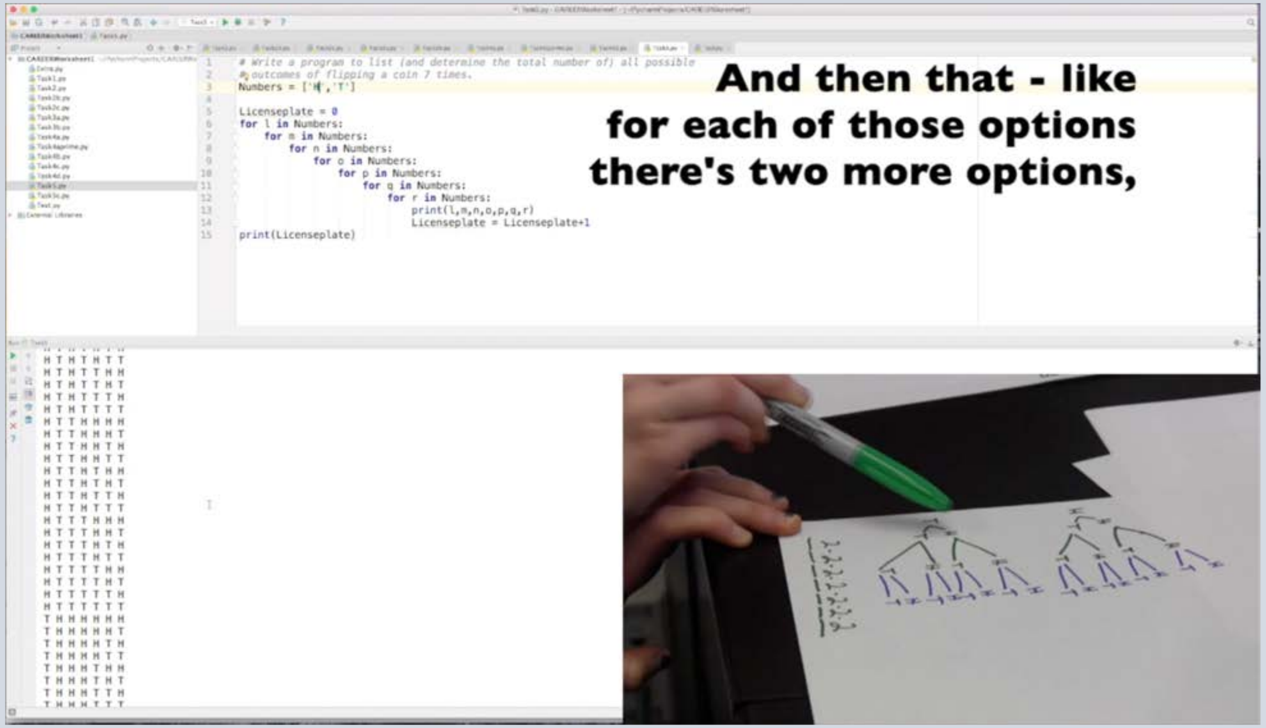
- The computational setting allows for **connections between multiple mathematical representations**
- Code, list of outcomes, tree diagram, mathematical expression





Affordances: Connecting Representations

- The computational setting allows for **connections between multiple mathematical representations**
- The students formulate a **justification** for why their expression works

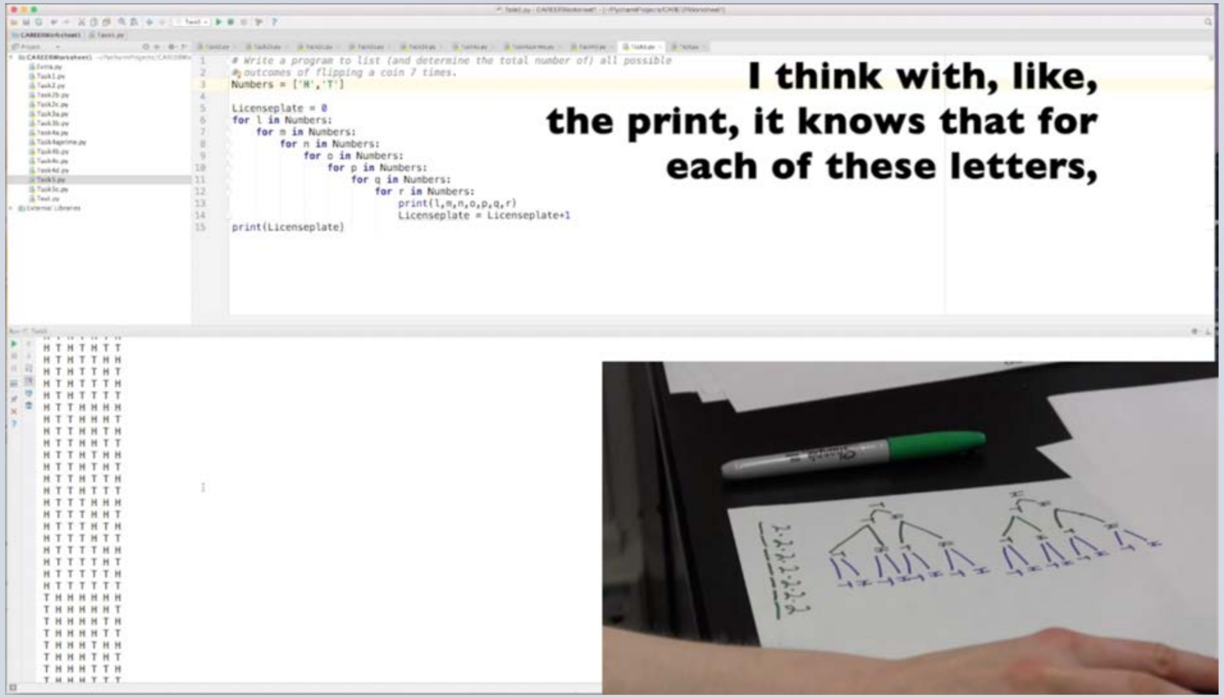


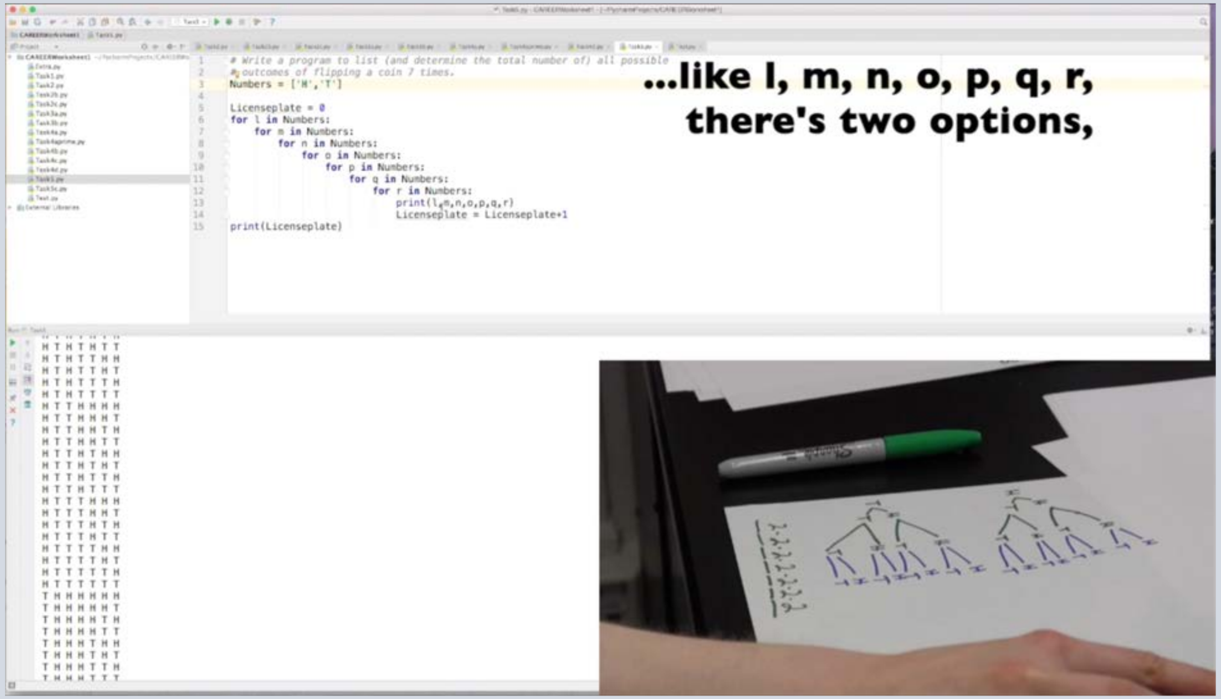
Affordances: Summary of this episode

1. The computational setting provides a **complete, surveyable list** of the set of outcomes, which
   - Allowed students to see **structures and patterns** in the outcomes
   - Such structure facilitated deeper **justification** of their response
2. The computational setting allows for **connections between multiple mathematical representations**

An example of Computational Thinking

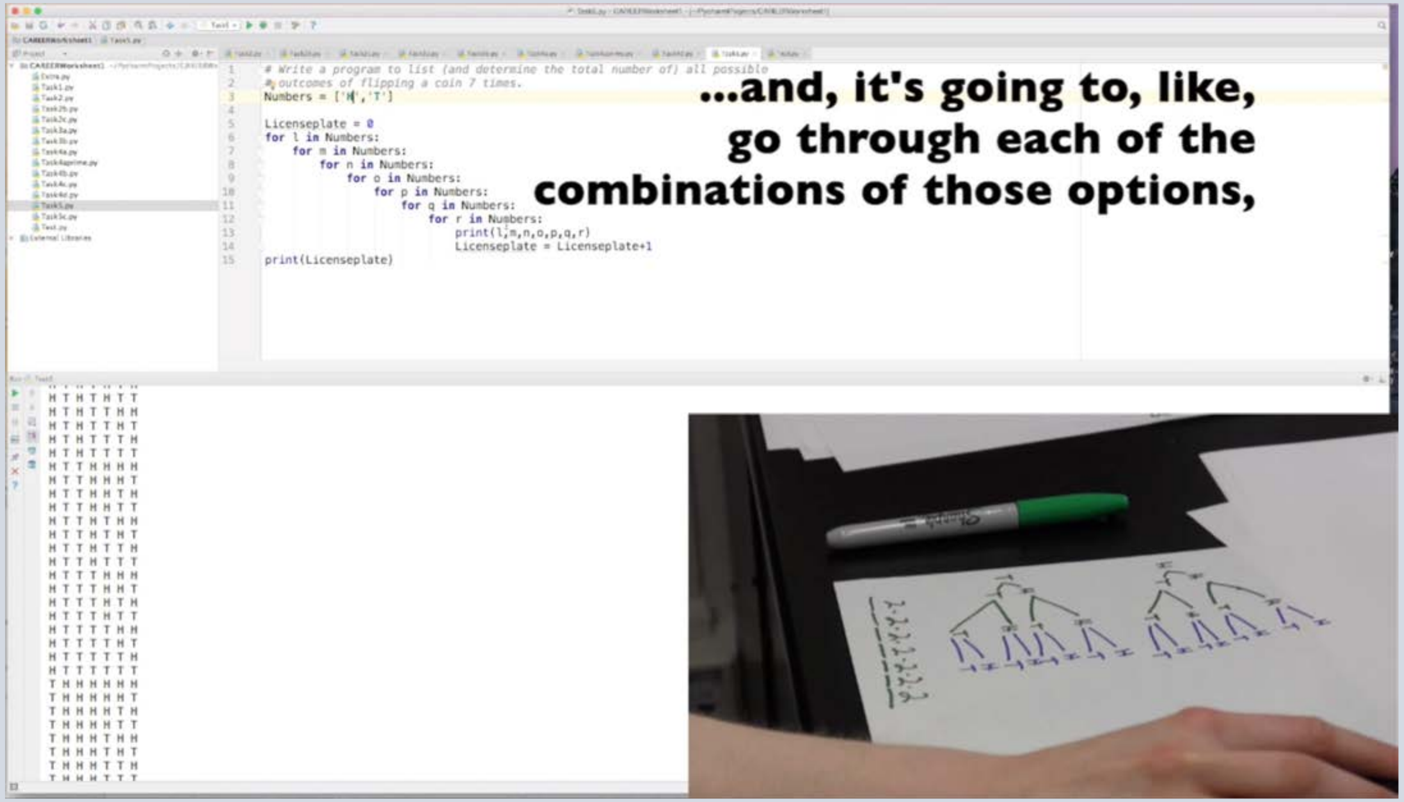
- I suggest that this is an example of computational thinking in mathematics





An example of Computational Thinking

- I suggest that this is an example of computational thinking in mathematics



Discussion: Examples of Computational Thinking?

- What do the two examples I offered suggest about computational thinking in combinatorics?
- The kind of thinking I want students to develop is the ability to **think about an algorithm or process and anticipate (or actually produce) what the output of that process will be**
- Is this computational thinking? Algorithmic thinking? Something else?
- Their CT did not arise just through computational activity – the questions about anticipating output and structure of outcomes also facilitated such thinking

Results so far

- I set out to discuss four affordances of solving combinatorics problems in a computational setting
  1. Students get **immediate feedback** on the output of their code
     - Experimentation, verification, and justification
  2. Students can **survey entire sets of outcomes**
     - Identify patterns and mathematical structure
  3. Students get opportunities to **connect multiple mathematical representations**
     - Code, lists of outcomes, tree diagrams, mathematical expressions
  4. Students gain insight about **key distinctions** between combinatorial problems
- I aimed to present examples of computational thinking

Results so far

- I set out to discuss four affordances of solving combinatorics problems in a computational setting
  1. Students get **immediate feedback** on the output of their code
     - Experimentation, verification, and justification
  2. Students can **survey entire sets of outcomes**
     - Identify patterns and mathematical structure
  3. Students get opportunities to **connect multiple mathematical representations**
     - Code, lists of outcomes, tree diagrams, mathematical expressions
  4. Students gain insight about **key distinctions** between combinatorial problems
- I aimed to present examples of computational thinking

Computing helped students identify key distinctions between problem types

- There are some fundamental problem types, permutations and combinations are a common distinction
- Very minor changes in code correspond to these different problem types, and this was meaningful for students

```
Numbers = [1, 2, 3, 4, 5, 6]

Counter = 0

for i in Numbers:
    for j in Numbers:
        for k in Numbers:
            print(i, j, k)
            Counter = Counter + 1
print(Counter)
```

```
Numbers = [1, 2, 3, 4, 5, 6]

Counter = 0

for i in Numbers:
    for j in Numbers:
        if j != i:
            for k in Numbers:
                if k != i and k != j:
                    print(i, j, k)
                    Counter = Counter + 1
print(Counter)
```

Permutations

```
Numbers = [1, 2, 3, 4, 5, 6]

Counter = 0

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

Combinations

Limitations – Computing is Not a Silver Bullet

- We don’t want students to become so reliable on computing that they neglect by-hand listing
- Numerical verification can be helpful, but it can be misleading
- We gave students the 3-letter sequences with the common error
  - For one pair, computing helped them recognize the potential error
  - For another pair, they solved it incorrectly mathematically and then correctly programmed their incorrect mathematical solution

- My goal in developing materials is to leverage the best aspects of non-computational and computational pedagogical interventions

Conclusions and Next Steps

- I have identified some affordances and limitations of having students solve counting problems in a computational setting
- The rigor of programming offers a novel way to have students think carefully about counting processes and sets of outcomes
- Using basic Python programming may be a good method for introducing students to some fundamental but important combinatorial ideas
- Moving forward, I will use these results to develop tasks for use in classrooms

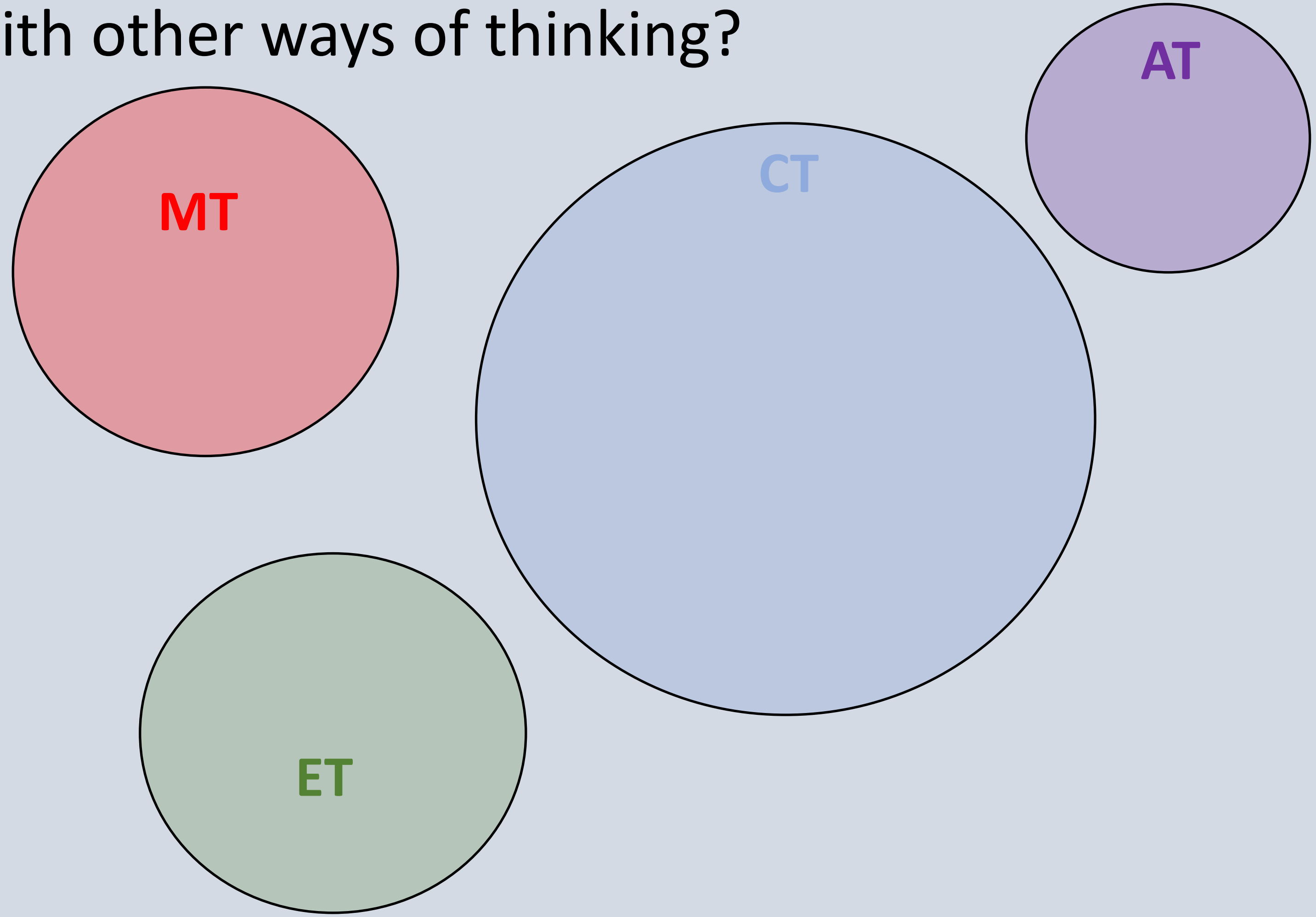
Future Research Endeavors in Computing in STEM

- There are opportunities for research related to computing (I believe) needs more attention

1. What is CT? – Theoretical investigations into the nature of CT itself
2. How does computational thinking and activity help students learn mathematical or scientific content?
3. How does computational thinking and activity help students develop mathematical or scientific practices?

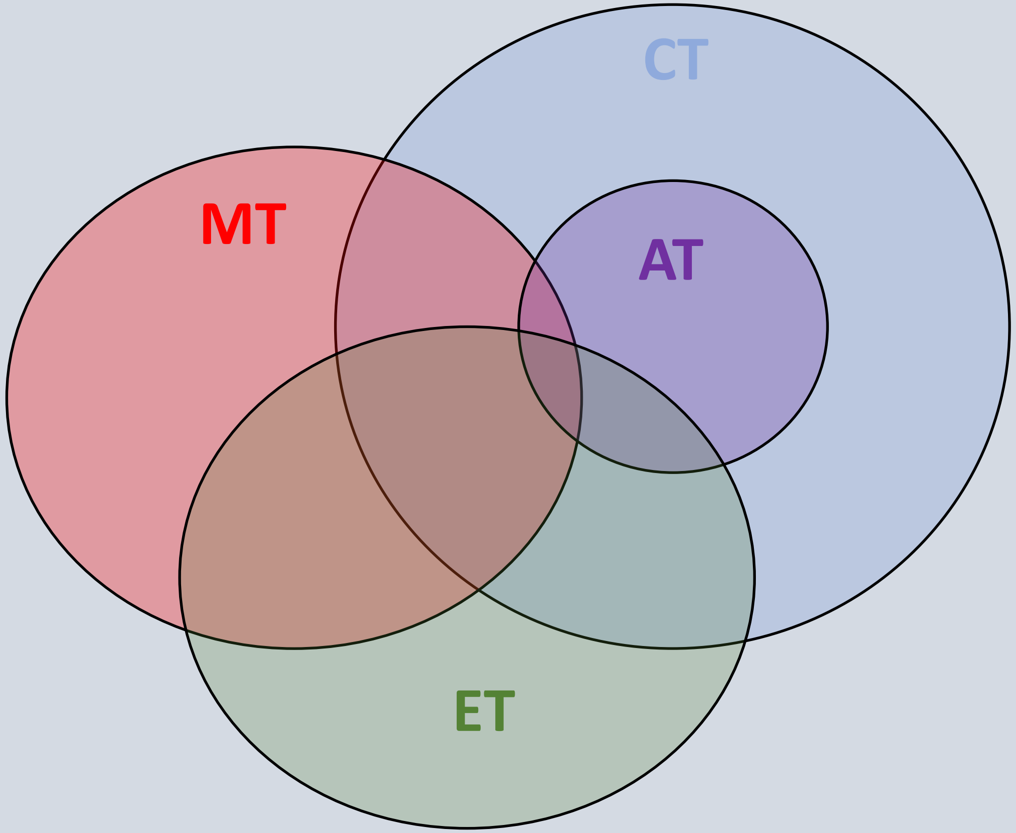
1. Theoretical investigations into the nature of CT itself

- How does CT interact with other ways of thinking?
  - Mathematical Thinking
  - Engineering Thinking
  - Algorithmic Thinking



1. Theoretical investigations into the nature of CT itself

- How does CT interact with other ways of thinking?
  - Mathematical Thinking
  - Engineering Thinking
  - Algorithmic Thinking



1. Theoretical investigations into the nature of CT itself

- Potential Research Questions:

- What is CT?
- What “counts” as CT and what doesn’t? Can there be computation without CT?
- How might CT be taught, learned, and assessed?
- Must computing involve a machine?

2. How does computing help students learn mathematical or scientific content?

- In this talk, I have offered one example of this kind of research

2. How does computing help students learn mathematical or scientific content?

- Potential Research Questions:

- How does engaging in a computational thinking and activity affect students’ understanding of a given mathematical or scientific concept
- How much (and what kinds of) computation is necessary to see gains in student understanding?
- How can computation practically be implemented into undergraduate and graduate classrooms?

3. How does computing help students develop mathematical or scientific practices?

- By practices I mean broad, overarching activities or perspectives that we want to foster in students
- Mathematical practices include problem solving, generalizing, justifying, proving, etc.
- Here “practices” could also entail dispositions or beliefs
- The ability to recover from mistakes, perseverance, beliefs about oneself, identity as a scientist or mathematician, etc.

3. How does computing help students develop mathematical or scientific practices?

- Potential Research Questions:

- How can computational thinking and activity help to develop desirable dispositions and attitudes?
- In what ways does the debugging process during programing affect students’ ability to recover from mistakes?
- Do students who engage in computational thinking and activity develop better and longer lasting problem solving skills?

Conclusion

- There are a variety of definitions and characterizations of computational thinking, but “it is time to redress the gaps and broaden the 21st-century academic discourse on computational thinking” (Grover & Pea, 2013)
- Educational research affords many methods and approaches we could use to study questions and phenomena related to computing in STEM
- The time is ripe to engage in exciting research related to computational thinking and activity in STEM!

Thank you!

Elise.Lockwood@oregonstate.edu

Acknowledgment:

NSF DUE #1650943