

Mathematics Learning in a Computer Programming Environment: Co-Actional Phenomenon,
Mathematical Significance, and Computational Thinking

Wendy Ann Forbes

Department of Educational Studies

Submitted in partial fulfillment of the
Doctor of Philosophy in Educational Studies

Faculty of Education, Brock University

St. Catharines, Ontario

© Wendy Ann Forbes 2025

Abstract

This study explored the nature of interactions between learners and their programming environment as they develop and program Exploratory Objects (EOs) for mathematics investigations. This interaction is conceptualized as *co-action*, a reciprocal process in which learners and their environment mutually shape each other. Grounded in the enactivist concept of structural coupling and informed by the theory of stigmergy, the research reconceptualizes learning as an emergent phenomenon that arises through embodied action, sensory feedback, and evolving environmental traces. Using a qualitative phenomenological approach, data were collected from six undergraduate students enrolled in Mathematics Integrated with Computer Applications (MICA) courses at a Canadian university. Data sources included semi-structured interviews, field notes, and exploratory objects (EOs). Thematic analysis, guided by Braun and Clarke's (2006, 2020) framework, revealed six key themes: pedagogical traces, Environmental Semantics and Social Traces, learning strategies and student agency, dynamic problem-solving, emerging perspectives and psychological empowerment, and development of proficiency. Findings highlight how learners' adaptive engagement with programming tools supports the co-emergence of computational thinking, mathematical reasoning, and learner agency. The study offers theoretical and pedagogical insights for integrating computational thinking into mathematics education in ways that are flexible, inquiry-driven, and grounded in students' lived experiences.

Acknowledgements

I would like to begin by sincerely thanking the participants in this study. Your willingness to share your experiences made this work possible. I am deeply grateful for your time, insights, and openness throughout this journey.

A special thank you to my supervisor, Dr. Joyce Mgombelo, whose mentorship has been both intellectually rich and emotionally supportive. You believed in me, and your unwavering support has been instrumental in helping me reach this milestone. To my supervisory committee, Dr. Steven Khan and Dr. Priscila Corrêa, thank you for your thoughtful feedback, encouragement, and academic guidance. Your expertise helped shape and strengthen this dissertation. To my friend and colleague Pinar Sen, you have been an anchor throughout this process. Thank you for always being present, for your shared wisdom, and for standing with me through the highs and lows.

To my friend Joy-Ann Ferguson Reid and Shernett Auld, your presence, deep listening, and willingness to hold space for my frustrations made a profound difference. You reminded me that I didn't have to carry it all alone.

To my dear friend Ann-Marie Dawn Roach, thank you for your unwavering encouragement. Your words, prayers, and tangible support provided fuel for my spirit during times when I felt depleted. Your friendship means the world to me.

To all my friends, thank you for walking with me in different ways. Whether through kind words, laughter, or simply your presence, your support sustained me.

To my family, this work is dedicated to my parents, Veronica Forbes and Delford Forbes. From my earliest days, your love, sacrifice, and belief in me created the foundation on which this accomplishment stands. I also want to express my deepest gratitude to my sister Trudy-Ann

Forbes-Harris and her family, whose unwavering love and encouragement lifted me through the toughest days. To all my siblings, thank you for stepping in at crucial moments to remind me of my strength and purpose. Your belief in me has carried me through.

To my husband, Delran Hall, whom I met along the way, thank you for your understanding and for embracing my work as a part of our shared journey, rather than a barrier between us. Your patience, love, and steady presence have been a profound source of strength.

Finally, I extend my thanks to everyone who, in ways both big and small, contributed to the successful completion of this dissertation. This accomplishment is not mine alone—it belongs to all of us.

Thank you.

Table of Contents

	Page
Abstract	
Acknowledgements	
List of Tables	
List of Figures	
CHAPTER ONE: INTRODUCTION.....	1
Background of the Problem.....	3
Rationale.....	6
Purpose of the Study.....	9
Research Question.....	11
Significance of the Study.....	12
Overview of the Chapters.....	13
CHAPTER TWO: LITERATURE REVIEW.....	15
Benefits of Learning CT.....	15
Definitions and Skills of Computational Thinking (CT).....	16
CT as a Thought Process for Problem-Solving.....	17
CT as a Problem-Solving Process.....	18
Educational Definitions.....	20
The Evolving Nature of CT Definition.....	22
Mathematical Thinking.....	24
Computer Technology in Mathematics Classroom.....	26
Early Years of Computer Integration in Classrooms.....	28
Behaviouristic Perspective of Computer Integration in the Classroom.....	29
Alternative to Behaviourism.....	31
Programming in Mathematics Education.....	43
Recent Integration of Computational Thinking in the K–12 Mathematics Classroom.....	46
Preservice Teachers Scaffolding CT Integration.....	52
The Programming Learning Environment.....	54
Chapter Summary.....	56
CHAPTER THREE: THEORETICAL AND CONCEPTUAL FRAMEWORK..	58
Enactivism.....	59
Embodied Action.....	63
Structural Coupling.....	64
Co-Action Within a Mathematics-Programming Context.....	71

Co-Action with Tools	75
Theoretical Basis for Identifying Components of the Emerging System of Co-Action	83
Stigmergy	84
Chapter Summary	101
CHAPTER FOUR: METHODOLOGY	102
Brief Overview of Phenomenology	102
Researcher Positionality	104
Participants for the Study	105
Data Collection Methods	106
Data Analysis.....	108
Transcribing the Interview	108
Thematic Analysis (TA).....	109
Limitations.....	116
Ethical Considerations.....	117
Recruitment and Informed Consent	117
Right to Withdraw or Not Answer a Question Without Penalty.....	118
Confidentiality	118
Member Check.....	118
Chapter Summary	119
CHAPTER FIVE: FINDINGS.....	120
Participants	121
Participants' Narratives.....	121
Result of Thematic Analysis	167
Perturbations: Trace-Driven Learning.....	169
Agents' Actions and Perceptions	186
Emergent System Behaviour: Proficiency, Perspectives, and Environment Shaping.....	110
Chapter Summary	226
CHAPTER SIX: DISCUSSION AND IMPLICATIONS	228
How the Themes Address Research Questions	230
Perturbations: Trace-Driven Learning	233
Agents' Actions and Perceptions	239
Emergent System Behaviour: Proficiency, Perspectives, and Environment Shaping.....	256
Implications for Theory	259
Implications for Practice.....	261
Implications for Research.....	265
Conclusion.....	267

References.....	269
Appendix A: Invitation Letter (Email)	293
Appendix B: Information-Consent Letter for Students	296
Appendix C: Semi-Structured Interview Questions (Final Project)	300
Appendix D: Email to Instructor.....	303

List of Tables

Table	Page
1 Comparing Barr et al. (2011) CT With Skills from Computer Science Teacher Association	20
2 Summary of Computational and Mathematical Occasioning Tools	82
3 Analysis Template for Co-Action for Agents Interacting With Environment	99
4 Participant Demographics	122
5 Emerging Themes and Subthemes From Co-Action	168
6 Excerpts From Interview Transcript Showcasing Teaching Strategies and Techniques.....	176
7 Excerpts From Interview Transcript Showcasing Learning Strategies and Techniques Participants Enacted in Response to Perturbations	188
8 Excerpts Showing Participant Perspectives on the Usefulness of Programming in Mathematics Contexts	214
9 Excerpts Illustrating Participants' Dispositions and Mindsets	219
10 Participants' Reflections on the History of Interaction With Programming Concepts	221
11 Subjective Relationship With Programming Software and Environments...	225
12 How the Themes Address the Research Questions.....	231

List of Figures

Figure	Page
1 Heylighen's (2016) Stigmergic Feedback Loop.....	88
2 Parunak's (2006) Basic Architecture of Stigmergy With Template of Analysis for Stigmergic System	90
3 Stigmergic Template for Agent Interacting in a Programming Environment	94
4 Basic Architecture of Stimergy for Agent Interacting in a Programming Environment	98
5 Example of Table Used in the Process of Thematic Analysis	110
6 Analysis Template for Co-Action for Agents Interacting With Environment	114

CHAPTER ONE: INTRODUCTION

Technology shapes the world we live in; therefore, “it is no longer sufficient to wait until students are in college to introduce [computational thinking] concepts” (Barr & Stephenson, 2011, p. 49). This assertion reflects a broader consensus in the literature (Lee et al., 2020; Saad & Zainudin, 2022; Shute et al., 2017; Wing, 2008; Yadav et al., 2017) that has fueled a growing movement to embed computational thinking (CT) across K–12 classrooms, teacher education programs, and educational research.

Broadly speaking, CT comprises a set of problem-solving processes—such as abstraction, decomposition, and algorithmic reasoning, rooted in computer science but applicable across disciplines (Wing, 2008; Yadav et al., 2017). These cognitive tools help reframe complex problems into computational tasks that can be managed by information-processing systems—human, machine, or both (Wing, 2006, 2010). The push to integrate CT across education levels is supported by research emphasizing its cognitive and educational benefits, particularly in enhancing problem-solving, reasoning, and communication skills (Buteau et al., 2017; diSessa, 2001; Liao & Bright, 1991; Saad & Zainudin, 2022).

CT is especially valuable in mathematics education, where it is increasingly integrated through programming-based strategies and technological tools (Benton et al., 2016; Benton et al., 2017; Misfeldt & Ejsing-Duun, 2015). Rooted in Papert’s (1972, 1980) constructionist philosophy, programming in mathematics education is seen not just as a technical skill but as a creative medium for students to explore and make sense of mathematics through hands-on, meaningful problem-solving. Studies highlight programming’s potential for fostering abstraction, modeling, and algorithmic thinking, particularly through low-floor, high-ceiling platforms like Scratch (Benton et al., 2016; Benton et al., 2017; Weintrop et al., 2016; Yadav et al., 2017).

Despite these potential benefits, and although recent curriculum reforms have incorporated programming to foster CT and support deeper mathematical engagement, pedagogical and theoretical challenges remain. To begin with, there is no universally accepted definition of CT (Shute et al., 2017; Tsarava et al., 2022), and many scholars argue that a stable definition is essential for guiding curriculum development (Selby & Woollard, 2013; Zhang & Nouri, 2019). Additional barriers include limited teacher training, difficulties aligning programming with curriculum goals, challenges in designing meaningful tasks, and the lack of robust methods for assessing CT skills (Misfeldt & Ejsing-Duun, 2015). Furthermore, more needs to be understood about how technology-based interactions can simultaneously support CT development and deepen students' mathematical understanding (Abrahamson & Sánchez-García, 2016; Buteau et al., 2017). Moreover, integrating key mathematical processes—such as problem-solving, modeling, and abstraction—into mathematics instruction is also complex and requires thoughtful design (Herbert, 2021). Research further shows that student underachievement in mathematics is often linked to outdated pedagogical methods rather than a lack of ability (Boaler, 2016; Garzon & Bautista, 2018).

Additionally, research calls for updated theoretical frameworks that better reflect the bidirectional nature of human–technology interaction (Hegedus & Moreno-Armella, 2010). Hegedus and Moreno-Armella (2010) critique traditional one-directional models, including those influenced by constructivism, for failing to capture the dynamic co-action between learners and technology. While Papert's (1980) notion of the computer as a “mathematical speaking entity” emphasizes learning through interaction, further inquiry is needed to fully understand this relationship.

Historically, computers were introduced into education using a behaviourist “amplifier” model focused on drill and practice. Later, constructivist perspectives reimaged computers as

tools for transforming mathematical activities (Yushau et al., 2004). Today, the bidirectional and emergent interactions between learners and technology (Hegedus & Moreno-Armella, 2010; Moreno-Armella, 2010), alongside the complexities mentioned above, call for new research approaches to provide direction for theory and practice.

In response, this study explores the co-action between learners and their environment as they engage with programming for mathematics learning, using an enactivist lens. Co-action refers to the mutual and evolving interaction between learners and their learning environment (Hegedus & Moreno-Armella, 2010). From an enactivist perspective, the focus is on how these interactions generate learning opportunities, emphasizing the guiding factors that shape learner actions. This lens allows for an in-depth exploration of how programming environments can support the development of both CT skills and mathematical competencies.

Background of the Problem

The increasing prevalence of digital technologies such as smartphones, robotics, and networked devices highlights their importance in modern society and underscores the need to emphasize digital literacy. Additionally, educational research points to the cognitive benefits of integrating CT strategies into learning through digital tools (Buteau et al., 2017; diSessa, 2001; Liao & Bright, 1991; Montuori et al., 2024). Thus, governments worldwide have prioritized Information and Communication Technology (ICT) literacy as part of developing 21st-century competencies to address the growing influence of digital technologies and ensure students are prepared for future workforce demands (Almazroa & Alotaibi, 2023; Fadel et al., 2015). Former Canadian Prime Minister Justin Trudeau highlighted the importance of CT as a critical problem-solving tool for equipping students to succeed in an increasingly algorithm-driven world (“Prime Minister Justin Trudeau Talks Importance,” 2016). Similarly, the National Research Council

(NRC) emphasized the need for early CT education (Yadav et al., 2017). In line with these priorities, former U.S. President Barack Obama allocated \$4 billion in 2016 to support CT initiatives in American schools (Wing & Stanzone, 2016).

In response to these global developments, CT themes have been increasingly integrated into curriculum frameworks worldwide (Weintrop et al., 2016). Australia, for example, recommends introducing digital literacy by the second grade, while Finland integrates CT into curricula beginning in first grade (Mason & Rich, 2019). England and New Zealand similarly highlight the importance of digital literacy to support CT development (Bower & Falkner, 2015). Mathematics education, in particular, has been identified as a fertile ground for fostering CT skills due to the shared histories and skills between mathematics and computer science (Misfeldt & Ejsing-Duun, 2015; Tooke, 2001). Furthermore, CT is reshaping mathematics education, with practices like data analytics, modeling, simulation, and computational problem solving becoming central to modern learning environments (Benton et al., 2017; Li & Ma, 2010; Weintrop et al., 2016).

Papert's (1972, 1980) pioneering work in mathematics education argued for the integration of computers as epistemic tools, promoting learning environments that accommodate multiple ways of thinking and problem-solving. In particular, Papert suggests programming as an object-to-think-with for mathematics learning. While initially underutilized, his vision is increasingly embraced, as research and curricula now emphasize the benefit of developing CT skills through coding and programming (Kynigos, 2015; Montuori et al., 2024; Weintrop et al., 2016). In a systematic review of 19 studies conducted between 2006 and 2022, Montuori et al. (2024) found that coding and programming activities, including robotics and virtual tools (e.g., Scratch), foster cognitive development, with notable improvements in problem-solving,

planning, working memory, and response inhibition. Montuori et al. further differentiate coding and programming as distinct yet interconnected aspects of CT within digital literacy. Coding involves writing specific instructions in a programming language to perform a task, emphasizing syntax and communication with machines. Programming, on the other hand, refers to the broader process of problem-solving, encompassing planning, debugging, and evaluating performance. Coding and programming activities in mathematics education have been recognized as effective for developing deep CT skills (Berland & Wilensky, 2015; Brennan & Resnick, 2012; Feurzeig & Papert, 2011; Papert, 1972, 1980; Zhang & Nouri, 2019). In this vein, the Programme for International Student Assessment (PISA) now includes CT in its 21st-century framework, acknowledging that mathematical literacy today requires not only traditional problem-solving but also mathematical reasoning and elements of CT (OECD, 2022).

As part of efforts to integrate CT into curricula, Ontario, Canada, updated its mathematics and science curricula in June 2020 to include coding for Grades 1 to 8. The coding component extends beyond software coding to encompass broader digital literacy skills, following earlier implementations in British Columbia and Nova Scotia, where coding was introduced across all grade levels 3 years earlier (“Ontario Releases,” 2020).

Various perspectives have shaped computer integration in mathematics classrooms, but constructivist frameworks have played a significant role by promoting CT through hands-on coding activities at all grade levels (Bundy, 2007; Kynigos, 2015). However, despite government policies promoting CT and efforts grounded in constructivist approaches, educators continue to face challenges integrating these new pedagogies into mainstream classrooms. Research shows that aligning CT with existing curricula requires rethinking instructional strategies and exploring the interaction between emerging technologies and traditional mathematics education (Drijvers, 2015; Guzdial, 2008; Lee et al., 2020; Yadav et al., 2017). Furthermore, CT’s evolving

definitions and dynamic nature (Shute et al., 2017) add complexity for educators striving to create meaningful and engaging learning experiences.

Ultimately, the rapid evolution of technology necessitates ongoing reflection on how CT reshapes both mathematical epistemology and pedagogy. Researchers suggest that educators and policymakers must reconsider theoretical approaches to account for the interaction possibilities of emerging technologies and their implications for mathematics education (Abrahamson & Sánchez-García, 2016). The continued exploration of innovative pedagogies is essential to equipping students with the CT skills required for future success in an increasingly digital and interconnected world.

Rationale

Since many countries, backed by national and global policies, recognize CT as a critical competency for thriving in the 21st century (Bower & Falkner, 2015), its integration into mathematics education has become a growing priority (Misfeldt & Ejsing-Duun, 2015). This shift raises important questions for educators and researchers about how students learn in digital mathematics environments and what pedagogical frameworks best support such learning (Drijvers, 2001; Shvarts et al., 2021). The momentum was reignited by Wing's (2006) influential statement: "to reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability" (p. 33). This renewed emphasis reflects a growing expectation that students become not only consumers of digital technology but also empowered creators through meaningful engagement with CT processes (Barr & Stephenson, 2011; Cabrera, 2019; Gadanidis, G., Brodie et al, 2017; Gadanidis, Cendros et al., 2017).

This recognition has spurred researchers to investigate how CT can be meaningfully embedded in mathematics education across grade levels (Benton et al., 2017; Buteau et al., 2017; Weintrop & Wilensky, 2015; Zhang & Nouri, 2019). While Papert's early work highlighted

programming's potential to advance both CT and mathematical thinking, contemporary studies emphasize the need for further empirical research to understand how CT can be effectively implemented within STEM education (Benton et al., 2017; Lee et al., 2020). Addressing this need requires not only innovative teaching practices, but also theoretical models that account for the dynamic and evolving nature of learners' interactions with digital tools.

Traditional constructivist approaches often overlook how cognition arises through the coordination of actions within dynamic environments. Towers and Martin (2015) call attention to this gap, while Hegedus and Moreno-Armella (2010) emphasize the significance of "bi-directional" interaction between learners and responsive digital tools, an interaction that can ground and advance mathematical understanding. Additionally, Shvarts et al. (2021) note that student engagement with digital mathematics environments involves "a complex system of multiple perception-action loops" (p. 452) involving agents and their environment, a dynamic that deeply shapes how learners conceptualize mathematical ideas. Vergnaud (2013) reinforces this notion by arguing that conceptualization—the process of exploring evolving mathematical objects and their relationships—unfolds within these very loops. Abrahamson and Sánchez-García (2016) similarly emphasize that conceptual knowledge emerges through students' situated interactions with their environment within task spaces. Hegedus and Moreno-Armella (2010) argue that studying co-action is essential to student learning as they interact with digital environments. Meanwhile, many scholars, including Abrahamson and Sánchez-García (2016), Alqahtani and Powell (2016), and Shvarts et al. (2021), call for a re-evaluation of learning theories to better address the interactive nature of digital technologies in mathematics education. Abrahamson and Sánchez-García (2016) caution against the "misalignment between theory of learning and emerging practices to which it should apply" (p. 204).

Hence, this study focuses on exploring co-action between learners and their environment as they interact with programming environments for mathematical investigation through the embodied lens of enactivism, which attends to the bi-directional nature of interaction between agents and their environment.

In addition, Bower and Falkner (2015) advocate for CT activities that reflect real-world problem solving, while others call for mathematics learning environments that mirror authentic disciplinary practices such as conjecturing, justifying, and reflecting (Boaler, 2016; Grootenboer, 2009). Increasingly, programming platforms are being used to foster these mathematical practices (Abrahamson & Sánchez-García, 2016; Buteau et al., 2017; Gueudet et al., 2014). However, understanding the co-actional nature of the learner's interaction with their environment using an appropriate framework is essential not only for understanding cognition and learning but also for effectively integrating CT tools into teaching and learning (Abrahamson & Trninic, 2015; Hegedus & Moreno-Armella, 2010; Shvarts et al., 2021).

Furthermore, Lye and Koh (2014) recommend conducting additional research in naturalistic settings to better understand how programming contexts influence mathematics learning. In their review of 27 intervention studies conducted over 5 years, they examined how CT develops through programming in both K–12 and higher education settings. Their findings emphasize the need for more classroom-based studies that investigate CT practices and perspectives in authentic educational environments. Specifically, they advocate for research situated in constructionist contexts, where learners engage in meaningful, problem-driven tasks and have opportunities to reflect on their learning through design and interaction.

Aligned with these recommendations, the context of this study is a series of three Mathematics Integrated with Computer Applications (MICA) courses—MICA I, II, and III.

These courses integrate both block-based and text-based programming, enabling students to explore pure and applied mathematical concepts through dynamic, visual, and interactive programs they design. These programs take the form of either exploratory objects (EOs) or learning objects (LOs). Exploratory objects are original programming tools created to investigate mathematical concepts or problems, while LOs are interactive teaching tools designed to support elementary or secondary mathematics instruction. The MICA context fosters creativity, problem-solving, and intellectual engagement, encouraging students to construct knowledge through interaction with authentic tasks (Papert, 1980). As such, it aligns closely with Lye and Koh's (2014) call for research in rich, reflective programming environments. In this study, participants' interactions with the programming environment during exploratory objects (EO) development served as the focal point for examining how computational thinking and mathematical learning emerged in tandem.

Purpose of the Study

Exploring the integration of CT into classrooms beyond computer science involves addressing complex questions (Drijvers, 2015; Guzdial, 2008). These include: What are the specific potentials of ICT for teaching and learning? What do non-computer science students understand about computing? What challenges will they face? What tools offer the most significant benefits for learners accessing CT? And how should educators structure learning environments to make computing skills accessible? Applying research findings in mathematics practice can help explore new learning possibilities, reduce student failure, and enhance overall interest in the subject (Boaler, 2016; Garzon & Bautista, 2018).

Given the need to answer these questions and the growing presence of technological resources in mathematics classrooms, researchers are making empirical recommendations to

bridge the gap between theory and practice. Drijvers (2015) analyzed six cases from leading studies in mathematics education, revealing that successful ICT integration depends on three key factors: the design of digital tools and tasks that exploit the tool's pedagogical potential, the teacher's role, and the educational context. Additional research highlights that, given the situated and emergent nature of both mathematics and CT, meaningful conceptual development emerges from exploring the interactions between learners and their environment through appropriate frameworks (Abrahamson et al., 2020; Abrahamson & Sánchez-García, 2016; Brennan & Resnick, 2012; Misfeldt & Ejsing-Duun, 2015; Papert, 1980; Turkle & Papert, 1990).

While theories such as constructionism (Papert, 1980) and the instrumental approach (Verillon & Rabardel, 1995) have provided valuable insights into the learning potential of technology in mathematics education, research must keep pace with advances in digital tools and developments in cognitive science (Abrahamson & Sánchez-García, 2016; Shvarts et al., 2021). Although coding is widely regarded as a means of integrating CT into mathematics, Li and Ma (2010) note that many studies focus on computer applications without adequately considering how learners interact with programming environments. Yadav et al. (2017) emphasize that while CT is familiar within computer science education, K–12 educators, administrators, and teacher educators often lack clarity about what CT entails (p. 55). Some experts argue that greater effort is needed in teacher education to address these challenges at the K–12 level (Rich et al., 2019; Yadav et al., 2017). There is also a need for further research to understand how post-graduate students engage with CT tools and the competencies they develop through this interaction (Gadanidis, G., Brodie et al, 2017; Gadanidis, Cendros, et al., 2017; Grover & Pea, 2013). Abrahamson and Sánchez-García (2016) argue that the lack of bold research on interactive mathematics learning hinders the development of evidence-based policies for integrating technological environments into education.

Consequently, the purpose of the study was to explore the nature of interactions between learners and their programming environment as they develop EOs for mathematics investigations. Specifically, it investigated the co-actions that emerged among students enrolled in MICA courses at an Ontario university, where students designed and implemented interactive environments for mathematics investigations.

Co-action, in this context, refers to the dynamic relationship wherein a learner's perceptually guided actions shape the learning environment, while the environment, in turn, shapes the learner's sensorimotor capacities, fostering further perceptual actions (Varela et al., 2016).

Research Question

To fully articulate the study's purpose, the following outlines the study's guiding question (Clark & Creswell, 2014):

1. What is the nature of co-action between mathematics learners and their environment as they engage in doing mathematics through programming?
 - 1.1 What perturbs learners in a mathematical programming environment?
 - 1.2 How do learners respond to the perturbations within a mathematics programming environment? (How are they shaped by the programming environment?)
 - 1.2.1 In what ways do perturbations in the programming environment prompt computational thinking?
 - 1.2.2 In what ways do perturbations in the programming environment prompt mathematical thinking.
 - 1.2.3 What perspectives do the learners form as they do mathematics within a programming environment.
 - 1.3 How do learners shape the programming environment for mathematics learning?

Significance of the Study

This study advances current understanding of how CT can be meaningfully integrated into mathematics education through programming environments, especially at the postsecondary level. While global policy initiatives increasingly position CT as a foundational 21st-century competency (Barr & Stephenson, 2011; Bower & Falkner, 2015), empirical research examining how learners interact with programming tools in naturalistic educational contexts remains limited (Lye & Koh, 2014). Addressing this gap is especially critical in mathematics education, where CT skills such as abstraction, decomposition, modeling, and algorithmic reasoning can support deeper engagement with mathematical content (Benton et al., 2017; Papert, 1980; Weintrop et al., 2016).

Drawing on enactivism (Varela et al., 2016) and stigmergy (Heylighen, 2015, 2016; Parunak, 2006), this study conceptualizes learning as an emergent process shaped through dynamic interaction between the learner and environment, where significance arises through embodied action and environmental traces guide ongoing engagement. In this view, programming environments are not merely tools for delivering content, but interactive emerging spaces where learners construct knowledge through embodied and iterative problem-solving. As such, the study contributes to educational theory by offering an alternative to one-directional learning models and provides practical insights into how CT and mathematics can be integrated through instructional design that supports learner agency and innovation.

Moreover, the findings have implications for teacher education, curriculum development, and policy by offering a deeper understanding of how programming tasks can occasion meaningful learning experiences. In particular, it speaks to calls for frameworks that attend to the bidirectional and emergent nature of technology-mediated mathematics learning (Abrahamson &

Sánchez-García, 2016; Shvarts et al., 2021). It also complements research efforts aimed at equipping teacher candidates with the pedagogical strategies and confidence needed to integrate CT into their future classrooms (Gadanidis, Cendros et al., 2017). While this study centres on postsecondary learners, its findings can inform broader initiatives in K–12 education to develop more responsive, interactive, and conceptually rich mathematics learning environments.

Overview of the Chapters

This dissertation is organized into six chapters. In Chapter 1, I introduce the study by outlining the background, rationale, purpose, research questions, and significance for the study. I explain why exploring the interaction between learners and programming environments is critical for understanding how CT supports mathematics learning.

In Chapter 2, I review the relevant literature on CT, mathematical thinking, and technology integration in mathematics education. I trace the historical and theoretical roots of these ideas, including constructivist and constructionist frameworks, and highlight key gaps.

In Chapter 3, I present the conceptual framework guiding the study. I draw on enactivism and stigmergy to explore how learners and environments co-evolve through reciprocal interaction. Concepts like embodied action, structural coupling, and co-action help me make sense of the dynamics I observed in programming-based mathematics learning.

In Chapter 4, I describe the methodology of the study. I explain my phenomenological approach, detail participant selection and data collection methods, and outline how I conducted thematic analysis. I also discuss ethical considerations, limitations, and the steps I took to ensure credibility and trustworthiness.

In Chapter 5, I present the findings from the study. I begin with narrative portraits of participants, annotated using the analytical framework, then share the major themes that emerged

from my analysis. These themes illustrate how students developed learning strategies, mathematical understanding, and computational thinking through their engagement with the programming environment.

Finally, in Chapter 6, I interpret the findings in relation to the research questions and theoretical framework. I discuss the implications for theory, practice, and future research. I conclude by reflecting on the study's contributions to the evolving conversation around CT in mathematics education and the importance of understanding learner–environment interactions.

CHAPTER TWO: LITERATURE REVIEW

Understanding how learners interact with their environment while engaging in programming for mathematical investigations is essential for exploring the role of CT in mathematics education. This chapter reviews the literature on the benefits of CT, its evolving definitions, and the associated skills and practices that have emerged over time. Additionally, it examines mathematical thinking (MT) and how the integration of computer technology in the classroom has evolved across various paradigms in cognitive science, teaching and learning, and philosophical worldviews. These shifts have influenced mathematics education and contributed to the growing emphasis on programming as a means of fostering CT.

Despite the promising benefits of integrating programming into mathematics education, questions remain regarding the bidirectional nature of learners' interactions with dynamic environments like programming. Existing theoretical frameworks have yet to fully capture these complexities, presenting gaps in the literature. This chapter explores these gaps and examines the proposed approaches for addressing them, setting the stage for a deeper investigation of the nature of co-action between learners and their programming environment.

Benefits of Learning CT

Various studies suggest that CT skills offer significant advantages in problem-solving (Buteau et al., 2017; diSessa, 2001; Liao & Bright, 1991; Saad & Zainudin, 2022). Buteau et al. (2017) emphasize that CT tools foster innovative approaches to mathematics problem-solving, enhancing engagement and cognitive abilities. diSessa (2001) adds that CT improves the ability to communicate computationally. Liao and Bright's (1991) meta-analysis of computer programming's effects on cognition found that learning to code enhances reasoning, logical thinking, planning, and general problem-solving skills, regardless of the specific programming

language. Saad and Zainudin (2022) argue that CT provides significant educational benefits by improving problem-solving abilities and promoting structured, algorithmic thinking, and that it is increasingly recognized in curricula worldwide for supporting essential 21st-century skills. However, despite these benefits, a universally accepted definition of CT remains elusive (Shute et al., 2017; Tsarava et al., 2022). While some researchers, such as Guzdial (2011) and Saad and Zainudin (2022), advocate for broad conceptualizations and frameworks to guide CT pedagogy, others argue for a clear and robust definition to inform curriculum development, classroom implementation and assessment (Selby & Woollard, 2013; Shute et al., 2017; Tsarava et al., 2022; Zhang & Nouri, 2019). The following section will explore how CT is defined in the literature.

Definitions and Skills of Computational Thinking (CT)

Ideas relating to CT can be traced as far back as the mid-20th century to “algorithmic thinking,” which denotes problem-solving by developing and applying a clear and precise sequence of step-by-step instructions (Denning, 2009). However, CT as a concept was first highlighted by Seymour Papert (1980) in his book *Mindstorms: Children, Computers, and Powerful Ideas*, in which programming is highlighted as a powerful tool to transmit CT skills (Zhang & Nouri, 2019). Computational thinking, popularized by Wing (2006) in her seminal article, now has a broader outlook, with components of CT emerging across many disciplines (Shute et al., 2017; Zhang & Nouri, 2019). Despite this broadening scope, certain common themes persist in the literature.

Selby and Woollard (2013) note that there appears to be an agreement that CT definitions should capture the essence of thought process and emphasize the concepts of abstraction and decomposition. Zhang and Nouri (2019) assert that “many definitions of CT in the 21st century

emphasize the concepts that are commonly engaged in programming or computer science” (p. 3) and categorize these definitions according to three themes: (a) those that emphasize the thought process for problem-solving leading to computational solutions, (b) those that focus on operational definitions in accordance with problem-solving processes presented by the Computer Science Teachers Association (CSTA), and (c) educational definitions that focus on CT practices necessary for schooling. Zhang and Nouri identify problem-solving as a common thread within all categories and define CT “as a thought process, through skills that are fundamental in programming (CT skills), to solve problems regardless of discipline” (p. 3). The following section of this chapter organizes the findings under the three categorizations of CT definitions proposed by Zhang and Nouri: CT as a Thought Process for Problem-Solving, CT as a Problem-Solving Process, and Educational Definitions.

CT as a Thought Process for Problem-Solving

Computational thinking (CT) is increasingly recognized as a thought process fundamentally driven by problem-solving. Janette Wing (2006, 2010) highlights that CT equips individuals with cognitive tools to reframe complex problems into computational tasks that can be managed by information-processing systems, whether human, machine, or a combination of both. This view is supported by Shute et al. (2017), who define CT as the conceptual foundation for solving problems efficiently and algorithmically, with or without computers, and for applying these solutions in various contexts. The Royal Society (2012) further emphasizes that CT enables individuals to identify computational aspects in their environment and apply computer science principles to understand and reason about both natural and artificial systems.

The role of computers in CT is seen as pivotal for transforming problems into information processes and solutions into algorithms (Berland & Wilensky, 2015; Papert, 1980;

Selby & Woollard, 2013). Programming, often associated with CT, is believed to enhance cognitive skills such as creative thinking, mathematical reasoning, and problem-solving (Scherer et al., 2019; Tsarava et al., 2022). Grover and Pea (2013) argue that programming is a key computational competency essential for developing CT, while others, like Yadav et al. (2017), suggest that CT can be nurtured through non-computer-based activities, such as incorporating algorithmic thinking into other subjects.

Consensus exists on several core skills associated with CT, including abstraction, decomposition, algorithmic thinking, evaluation, and generalization (Selby & Woollard, 2013). Additionally, CT overlaps with various forms of thinking, including logical, algorithmic, engineering, and mathematical thinking (Grover & Pea, 2013). Hsu et al. (2018) illustrate that mathematical problem-solving skills are integral to computational thinking, demonstrated through the use of computers to solve problems and create digital content.

While CT shares features with other cognitive skills, Denning and Freeman (2009) assert that its distinct focus on information processes driven by problem-solving sets it apart. Although research indicates that CT is a thought process that strengthens cognitive abilities, Tsarava et al. (2022) emphasize the need for broader investigation into both cognitive and non-cognitive factors affecting CT development. Tsarava et al. posit that expanding research in this area is crucial for enhancing the effectiveness of CT curricula and assessment tools.

CT as a Problem-Solving Process

Practices in CT vary with the research focus and operationalization process. Zhang and Nouri (2019) posit that the range of operational definitions based on problem-solving processes is somehow hinged on CT skills provided by the Computer Science Teachers Association (CSTA): “formulating, organizing, analyzing, automating, presenting, implementing, and

transferring” (p. 3). One example of these operational definitions is put forward by a group of educators interested in CT, in response to a charge by the U.S.-based National Science Foundation (NSF) to make CT concepts more accessible to all grade levels (PK–12) (Barr et al., 2011). By their definition, computational thinking as a problem-solving process involves the following:

1. Formulating problems so that computational tools can be used to attempt solutions
2. Organizing and analyzing data logically
3. Abstracting data using methods such as models and simulations
4. Thinking algorithmically to automate solutions
5. Considering efficiency and effectiveness in identifying, analyzing, and implementing steps and resources to solve problems.
6. Employing CT skills as a general problem-solving process to a broad cross-section of problems.

These educators argue that the process mentioned above may develop CT skills across curricula through all content areas. Additionally, they explain that various attitudinal factors such as confidence and persistence are crucial for scaffolding and enhancing the CT problem-solving process (Barr et al., 2011).

Table 1 illustrates how the skills involved in an operational definition—that of the educators (Barr et al., 2011)—correspond to those put forward by the Computer Science Teacher Association (CSTA). As can be observed from Table 1, although there is not necessarily a one-to-one correspondence between operational skills involved in the problem-solving process by Barr et al. Still, the general operation of “formulating, organizing, analyzing, automating, presenting, implementing, and transferring” (Barr et al., 2011, p. 3) can be seen.

Table 1

Comparing Barr et al. (2011) CT With Skills from Computer Science Teacher Association

CSTA	Group of educators (Barr et al., 2011)
Formulating	Formulating problems in such a way that computational tools can be used in solution attempts
Organizing	Organizing and analyzing data logically
Analyzing	Abstracting data using methods such as models and simulations
Automating	Thinking algorithmically to automate solutions
Presenting	Considering efficiency and effectiveness in identifying, analyzing, and implementing steps and resources to solve problems.
Implementing	
Transferring	Employing CT skills as a general problem-solving process to a wide cross-section of problems

Educational Definitions

CT educational definitions emerge from the core affordances of computer science based on a curriculum's need to outline CT teaching and learning skills, concepts, and practices involved in grade K–12 schooling. Educational studies in CT express the need for PK–12 educators to have a clear and practical definition of CT to guide their work to assist students in acquiring requisite skills (Barr et al., 2011; Selby & Woollard, 2013; Weintrop et al., 2016). An example of an educational definition is by Weintrop et al. (2016).

Weintrop et al. (2016) outline four main taxonomies that are specific to the context of mathematics and science: “data practices, modeling and simulation practices, computational problem-solving practices, and systems thinking practices” (p. 2). Data practices refer to the extent to which an individual can collect, create, and manipulate visual data. Modeling and

simulation refer to the degree to which individuals can use a model to understand a concept, find and test a solution, and make assessments to design and construct a computational model.

Computational problem-solving practices refer to the extent to which an individual can prepare a solution to be admitted by a computational model, program the solution using an established programming language, choose appropriate computational tools, evaluate different solutions to problems, develop modular computational solutions, create computational abstractions, debug, and troubleshoot when carrying out solutions. Systems thinking refers to an individual ability to solve complex problems by thinking about them as a whole and in parts—thinking in levels to define, communicate, and manage complexity. Weintrop et al. (2016) derived the taxonomy from literature, interviews with practicing experts in the fields, and instructional materials. Another example of educational definition is by Brennan and Resnick's (2012) three dimensions of CT practices. This definition highlights three sets of skills, computational concepts, computational practices, and computational perspectives, that students utilize or develop through engagement with programming tasks. These are explained further in the theoretical framework of this research.

Furthermore, Zhang and Nouri (2019) explain that components of CT have emerged across disciplines (Shute et al., 2017) and in curriculum documents around the world under various names such as “ICT capability” and “digital competence” (Zhang & Nouri, 2019, p. 1). Coding, programming, CT, and computing are generally common to these documents; however, there is a conflation in meaning among these terms such that they are often used synonymously (Zhang & Nouri, 2019). Moreover, coding, programming, and CT are all under the umbrella of computing. Computing involves understanding the principles which govern computation and information, using that knowledge to create and manipulate digital systems, and expressing and communicating ideas through programming (Department of Education UK, 2015). In addition,

Zhang and Nouri (2019) explain that although in everyday use, programming and coding are often used to mean the same thing, originally, coding was considered as a task that is part of the more complex processing of programming, which also involves problem decomposition, design, and maintenance. Wing (2006) cautioned that while CT can be defined by thought processes engendered by computer programming, it should not be erroneously conceptualized as simply programming a computer since multiple layers of abstraction are required to think computationally. Still, Berland and Wilensky (2015) clarify that CT is a mental process that may be functionally conceptualized as thinking “like a computer programmer or computer artist” (p. 630). Furthermore, Wing (2006) states the following:

Computational thinking is thinking recursively. It is parallel processing. It is interpreting code as data and data as code. It is type checking as the generalization of dimensional analysis. It is recognizing both the virtues and the dangers of aliasing, or giving someone or something more than one name. It is recognizing both the cost and power of indirect addressing and procedure call. It is judging a program not just for correctness and efficiency but for aesthetics, and a system’s design for simplicity and elegance. (p. 33)

The Evolving Nature of CT Definition

The literature suggests that computational thinking is a refined term that emerged from a deep history of observing cognitive processes motivated by problem-solving and is most often expressed using technological tools. Denning (2009) indicates that its origin stemmed from algorithmic thinking. Still, algorithmic thinking itself is rooted in mathematics and computer science, which are interrelated and cover a wide range of cognitive processes linked to other disciplines (Knuth, 1985). Technological advancement across disciplines and extensive ongoing research have popularized CT, bringing new interpretations of skills, practices, and perspectives.

Furthermore, even though computational thinking skills and practices are linked to non-computer related problem-solving, a vast expanse of literature suggests that computational thinking is incubated in a programming environment, giving learners the leverage to use their personal approach in developing public and sharable objects, artifacts, or instrument (Berland & Wilensky, 2015; Brennan & Resnick, 2012; Papert, 1980; Wing, 2006; Zhang & Nouri, 2019). Given that personal appropriation (cultural and socially situated), through occasioning (Kieren et al., 1997), propels evolving CT perspectives (Brennan & Resnick, 2012), this suggest that CT is emergent, complex, and dynamic. Moreover, the literature's inability to capture a clear definition or stable skillsets of CT (Selby & Woollard, 2013, Shute et al., 2017, Zhang & Nouri, 2019) indicates these characteristics.

While many researchers believe that a stable definition is necessary to conceptualize CT, similar to Guzdial (2011), this research embraces the idea that a general perception of CT is sufficient to operationalize CT in research given its emergent, complex, and dynamic nature. Also, as this research seeks to understand the co-action between learners and their environment, evaluated by the world of significance they bring forth, receiving a precise definition would be counterproductive. This would be adopting a prescriptive lens, which limits possibilities rather than a proscriptive lens, which opens up possibilities (Varela et al., 2016). From prescriptive logic, what is not allowed is forbidden, whereas from proscriptive logic, what is not forbidden is allowed (Varela et al., 2016). A stable definition through a prescriptive lens would put the observer in danger of inadequately explicating learners' experience as they couple with a programming for mathematics environment and would counteract the theoretical presumption of this research. On the other hand, embracing a proscriptive perspective, the observer has the leverage to explore the emerging CT skills and practices of learners in relation to their emergent understanding from coupling with their technological milieu.

Against this backdrop, this research defines computational thinking as an evolving process whereby individuals develop skills mediated by their social and cultural histories as they interact with their environment, using programming tools through occasioning as a versatile tool to think with. Given the programming context, to begin to conceptualize CT, the research embraces Brennan and Resnick's (2012) three dimensions of computational thinking as occasioning tools: computational concepts, computational practices, and computational perspectives.

Mathematical Thinking

Over the last few decades, progressive emphasis has been placed on mathematical thinking processes, practices, and ways of doing rather than on content (Goos & Kaya 2020; Mason et al., 2010). In particular, research in the 1980s documented a wave of interest in mathematical processes (Burton, 1984; Mason et al., 1982). Mason et al. (1982, 2010) propose four central processes of mathematical thinking: specializing, conjecturing, generalizing, and convincing. These processes have been further explored in the literature, as detailed in the conceptual framework section, using the case discussed by Burton (1984).

The interest in delineating the mathematical processes, which is seen as the means of acquiring and applying mathematical knowledge, has carried through to the 21st century as evident in curriculum documents such as *Principles and Standards for School Mathematics* developed by the National Council of Teachers of Mathematics (2000) in the USA, and Ontario's curriculum (Ontario Ministry of Education, 2020). These documents highlight mathematical processes as a means of applying mathematical content and skills across grade levels. In the U.S. document, process standards include problem-solving, reasoning and proof, communication, connections, and representations. In the Ontario document, mathematical

processes include problem solving, reasoning and proving, reflecting, connecting, communicating, representing and selecting tools and strategies. In both documents, the processes are interconnected and conjecturing, and justification is critical to reasoning and proving.

Furthermore, the fundamental nature of mathematical processes is captured in the five strands of mathematical proficiency presented in *Adding it up: Helping Children Learn Mathematics* (Findell et al., 2001). These strands represent key capacities that all students should develop to achieve success in mathematics:

- *Conceptual understanding* refers to grasping mathematical concepts, operations, and the relationships among them.
- *Procedural fluency* involves executing mathematical procedures with flexibility, accuracy, efficiency, and appropriateness.
- *Strategic competence* is the ability to formulate, represent, and solve mathematical problems effectively.
- *Adaptive reasoning* denotes the capacity for logical thinking, reflection, explanation, and justification.
- *Productive disposition* reflects a consistent tendency to see mathematics as logical, valuable, and worthwhile, along with confidence in one's own ability and persistence.

As evident in the standard for mathematical proficiency, the essence of mathematical thinking processes, as discussed earlier, is most prominently observed in the strands of strategic competence and adaptive reasoning.

Moreover, Drijvers et al. (2019) identify three major approaches to mathematical thinking, problem-solving, modeling, and abstraction, which collectively capture the essence of mathematical processes. Problem-solving, emphasized in both the U.S. and Ontario curriculum

documents, involves devising strategies to tackle non-routine problems. Modeling refers to conceptualizing or posing a problem based on a real-world context, translating it into a mathematical formulation, and performing computations to reach a solution. In essence, modeling entails “connecting mathematics and the world around us” (Drijvers et al., 2019, p. 439). Abstraction, the third approach, involves engaging with mathematical objects and relationships by isolating specific attributes for focused consideration. Drijvers et al. argue that these three elements—problem-solving, modeling, and abstraction—are central to mathematical thinking and should be integrated into curriculum frameworks. They further propose a theory-based model to support the analysis of mathematical assessment.

While integrating mathematical processes is recognized as essential for fostering students’ full mathematical proficiency, significant gaps persist. These include curriculum overload that prioritizes breadth over depth (English & Gainsburg, 2015; OECD, 2020), limited teacher preparation and professional development (Herbert, 2021; OECD, 2020), student disengagement and low confidence in problem-solving (McCarthy-Curvin et al., 2020), assessment systems that emphasize rote procedures over reasoning (English & Gainsburg, 2015; Herbert, 2021), lack of instructional time, insufficient access to high-quality tasks and manipulatives (Herbert, 2021), and inequitable availability of technological tools and support (Ling & Mahmud, 2023), all of which hinder the meaningful implementation of mathematical processes in classroom instruction.

Computer Technology in Mathematics Classroom

Research shows that student failure in mathematics is often attributed to outdated teaching methodologies instead of a lack of talent or academic difficulties (Boaler, 2016; Garzon & Bautista, 2018). However, further research suggests that computer technology, rightly

appropriated, can effectively raise students' mathematics achievement (Aydin, 2005; Garzon & Bautista, 2018; Li & Ma, 2010; Misfeldt & Ejsing-Duun, 2015). In a systematic literature review by Li and Ma (2010) on the impact of computer technology on mathematics education, a meta-analysis involving 36,793 K–12 learners reveals that computer technology positively impacts mathematics achievement. This may be because the flexibility of the computer affords multiple ways of thinking and knowing (Turkle & Papert, 1990), and there is a symbiotic relationship between mathematics and computer disciplines (Misfeldt & Ejsing-Duun, 2015; Tooke, 2001). In fact, computer science originated as a part of mathematics before gaining independence as a sole discipline (Aydin, 2005; Tooke, 2001):

There is a trivially obvious connection between mathematics and the computer. It is, in fact, a symbiotic relationship. Without mathematics, the computer would not even exist. However, the existence and development of the computer have enhanced mathematics, allowing us to go beyond the mathematics of imagination and paper only... Mathematics gave birth to computer science, but together they have both developed significantly. All of this has certainly had an impact on many areas of mathematics education, including the mathematics curriculum, mathematics instruction, and mathematics learning. (Tooke, 2001, p. 2)

There are various ways that the computer is integrated into mathematics classrooms: from tutorial software that teaches mathematics by drill and practice to communication media such as the internet and videoconferencing that facilitate easy and effective sharing of information to programming languages such as C++ and Python (Li & Ma, 2010). The manifestations of computer integration in mathematics classrooms are tied to a complex network of paradigmatic changes in cognitive science, teaching and learning, and philosophical worldviews. Some

schools of thought that have impacted mathematics education and education, in general, include behaviourism which focuses on shaping and measuring observable behaviour; cognitivism which views mental processes as information-processing (input, storage, processing, and output) and application of information; constructivism which focuses on the active creation and co-creation (sociocultural) of knowledge and, humanism which focuses on self-actualization. In the next section, I explored how different perspectives of cognition impacted the evolution of computer technology in mathematics classrooms.

Early Years of Computer Integration in Classrooms

The early years (1960s) of computer integration in the classroom within an evidence-based positivist paradigm coincided with the early (mid-1950s) development of cognitive science centered in cognitivism and behaviourism learning theories (at its tail end). At the onset of cognitivism, the dominant metaphor for the brain was that of a sequential processing computer driven by stimulus (Thagard, 2005). Brain operations (i.e., intelligent behaviours) were considered as computations of symbolic representations imputed from a pre-given world; thus, human experience was absent from the calculus of the mind. This idea flourished for an extended period with little controversy since the era of cognitivism (void of human experience) smoothly transitioned into behaviourism, which completely ignores the mind (seeing it as a black box) (Varela et al., 2016). Although the advancement of computer technology was beginning to penetrate the zeitgeist of behaviourism (by facilitating investigational hypotheses or justifiable theories about the mind), ideals in behaviourism that associate observable behaviours to stimuli ideals still had a stronghold on educational practices and operation. Consequently, the initial years of computer integration in classrooms were contextually situated within the thinking of the time: cognitive behaviourism.

Behaviouristic Perspective of Computer Integration in the Classroom

Behaviourism has its root in psychology (governs affairs concerning the mind) before being taken by cognitive science. Behaviourism was coined by John B. Watson in his 1913 proposal, counteracting the then hegemonic introspectionist approach, which situated conscious experience within the mind (Wang, 2013; Yushau et al., 2004). Experimental psychologists were wary of the incommensurable nature of introspectionist methods, which opened questions about its legitimacy as a scientific procedure (Varela et al., 2016). Watson successfully proposed observable human behaviour (a measurable attribute) as the groundwork for rigorous scientific inquiry to empirically validate human psychology (Wang, 2013; Yushau et al., 2004). Through a behaviourist lens, inputs (stimuli) and corresponding outputs (behaviour) of an organism could be objectively observed to determine the lawful relationships between variables over time; however, the biological makeup of an organism was methodologically incomprehensible to behavioural science. This mindset was readily accepted under the positivist epoch of *mind-free* objectivism; however, this nuptial simultaneously divorced the mind from human experience (Varela et al., 2016).

Pavlov's (1927) theory of classical conditioning, which result from his experimental studies with dogs, anchored Watson's (1913) behaviouristic thinking. Classical conditioning posits that an involuntary response to a neutral stimulus can be learned when a neutral stimulus is paired with a potent stimulant. Further, B. F. Skinner (1957) extended Edward Thorndike's (1927) work on the Law of Effect, introducing operant conditioning within behaviourism learning theory. Operant conditioning learning theory is based on reinforcing or discouraging voluntary actions by paring them with rewards or consequences, respectively. In general, behaviourism puts forward that all behaviours can be conditioned through interaction with the

environment. That is, human beings develop new actions as they repeatedly respond to stimuli from their environment, not considering the internal mind (Wang, 2013; Yushau et al., 2004).

Behaviourist thinking prevailed until the mid-1950s and lingered even longer, impacting teaching and learning. Instructional designs based on a behaviourism framework mainly focus on accomplishing a specific set of objectives, which is repeated sequentially and hierarchically. For example, mathematics is taught from a behaviourist perspective when teachers aim to improve students' ability to answer questions by exemplars to replicate multiple responses, without consideration of innate or inherent factors such as social and cultural experiences of a student. However, Towers and Davis (2002) suggest that behaviourist teaching directs attention toward activities such as articulating clear, observable outcomes and designing structured tasks and reward systems to increase the likelihood of achieving those outcomes. The mindset is that learning is presupposed by interaction with the environment.

A behaviourist mindset is reflected in the amplifier perspective of computer usage. From this perspective, the computer enhances students' test-taking abilities (Pea, 1987; Yushau et al., 2004), aligning with what Boaler (2016) describes as a performance-based approach to mathematics. The amplifier perspective views the computer as a tool that serves as a stimulus in the mathematics classroom, extending the degree to which students can replicate instruction to achieve "growth and mastery of an individual's knowledge of an existing portion of mathematics" (Yushau et al., 2004, p. 167). This aligns with the behaviourist belief that learning occurs through reinforcement and repetition. In this view, the computer functions as a stimulus that amplifies a student's ability to practice and reinforce learned material. Rather than fostering the creation of new knowledge, amplifier-based software primarily focuses on improving speed, reinforcing content-specific concepts, and developing skills through drill and practice (Pea, 1987; Yushau et al., 2004). A key advantage of this type of software is its ability to provide

individualized instruction by adapting computerized questioning to match a student's demonstrated mastery (Yushau et al., 2004). While behaviourism acknowledges the role of the environment in learning, Towers and Davis (2002) argue that its emphasis on external stimuli and response reinforcement relies on “the uncritical assumption that learning is a matter of constructing an inner model of an outer reality” (p. 315).

Alternative to Behaviourism

By the 1970s, as developing computer technology opened new alternatives about how the mind functions, granting society an unparalleled mirror of itself, knowledge about cognition and learning began to expand. This opens an era of cognitive science, which is inextricably linked to this mirror (Varela et al., 2016). For example, computer technology facilitates artificial neural networks that model how the brain works in a simplified way. Neural networks are the visible front to the connectionism or emergence movement in cognitive science.

Connectionism postulates that the brain consists of neural components that give rise to global properties, which, when appropriately connected, provide insight into cognitive dynamism. The configuration of neural models enables the exploration and corroboration of theories long overshadowed by cognitivism. For instant, the connectionist model of the brain confirms Donald Hebb's proposition of 1949 about neuronal implications of learning. Hebb suggested that connected neurons that are active together are strengthened and keep getting stronger when the same stimulus is recurrent; ultimately, pathways are created in the brain, which allows habitual actions to be intuitive (Varela et al., 2016).

Undoubtedly, the visualization of theories replicates in questions that disturb the dominant edifice of cognitivism: Is the mind a sequentially stimuli-driven processor? Is the mind a symbolic representation manipulated by computational structures? (Varela et al., 2016).

With evolving theories embracing connectionism, new approaches to teaching and learning surfaced in the classroom, opposing the behaviourism standpoint (Thagard, 2005), such that the amplifier view of computers slowly began to loosen its stronghold (Kurland & Kurland, 1987). Opponents of the behaviouristic approach argue that learners are not just passive receivers of carefully designed instructions but active figures in knowledge construction. With the emerging paradigmatic shift, new possibilities for computer usage began to take shape (Kurland & Kurland, 1987).

Constructivist Viewpoint of Computer Integration in the Classroom

A popular critique of the behaviourist's view of using computers in mathematics education comes from Seymour Papert (1980). Decades ago, he opposed using the computer as an amplifier, arguing that learners should be empowered to use the computers to cultivate knowledge to inspire creativity and innovation (Papert, 1972). He describes using the computer differently from how the computer is traditionally used to "amplify" mathematics in the classroom, in a way similar to the organizer's perspective. From the organizer's standpoint, a computer is a tool for reshaping mathematics activities (Pea, 1987; Yushau et al., 2004). The focus of mathematics is not primarily to fulfill a set of objectives but to develop mathematical process competencies such as conjecturing, reasoning and proving, abstraction, generalization, and communicating. The organizer approaches the computer as an instrument to liberate and empower students to embark on the same creative thinking journey as mathematicians; students are often seen as the creator of knowledge (Papert, 1972; Yushau et al., 2004). Subsequently, computer-related tasks are designed to extract creativity through modeling, visualization, and exploration. The premise here is that mathematics knowledge is constantly evolving, and existing principles are simply the groundwork or the starting point for investigation, as expressed below.

The following is an expression of Papert's (1972) way of utilizing computers in the classroom.

...in a way that is very different from the usual suggestions of using them either as "teaching machines" or as "super-slide-rules." In our ideal of a school mathematical laboratory the computer is used as a means to control physical processes in order to achieve definite goals—for example as part of an auto-pilot system to fly model airplanes, or as the "nervous system" of a model animal with balancing reflexes, walking ability, simple visual ability and so on. To achieve these goals, mathematical principles are needed; conversely, in this context, mathematical principles become sources of power, thereby acquiring meaning for large categories of students who fail to see any point or pleasure in bookish mathematics and who, under prevailing school conditions, simply drop out by labelling themselves "not mathematically minded." (Papert, 1972, p. 251)

While Papert interpreted the integration of the computer in the classroom from a constructionism framework, his organizer's view stemmed from a constructivist philosophy of learning. Papert's constructivism will be looked at in the next section, but I will now look at constructivism.

There is no stable definition for constructivism as a result of how often the theory has been cited in educational literature (Kretchmar, 2019; Towers & Davis, 2002); however, most authors seem to maintain that constructivism involves situations in which learners construct their own meaning and knowledge as they experience the world (Confrey, 1999; Kretchmar, 2019; Proulx, 2006; Towers & Davis, 2002). The constructivism ideology of experience is evident in Yushau et al.'s (2004) organizer description. The authors purport that "the students doing mathematics is the hallmark of this conceptualization, and it is the doing (experimenting,

abstracting, generalizing, and specializing) that constitutes mathematics, not the transmission of a well-formed communication” (Yushau et al., 2004, p. 168). In contrasting their idea of doing mathematics to the transmission of information inherent in the amplifier's view of using the computer for mathematics learning, Yushau et al. (2004) suggest a reformation in doing mathematics that allows students to make their own meaning. This is the underlying principle of constructivism.

The developmental process of a learner's experience based on constructivism may be interpreted from a cognitive perspective by Jean Piaget or social perspective by Lev Vygotsky or both, depending on the theoretical assumption of the inquirer (Kretchmar, 2019). Jean Piaget’s cognitive constructivism posits that knowledge is constructed through adaptation, where learners assimilate or accommodate new information to resolve cognitive disequilibrium (Bruner, 1997; Confrey, 1999). Rooted in biology, Piaget emphasizes logical operations and individual interaction with the environment (Kretchmar, 2019; Proulx, 2006). In contrast, Vygotsky’s social constructivism highlights social interaction, language, and culture in cognitive development (Bruner, 1997; Vygotsky, 1978a). Learning occurs within the zone of proximal development (ZPD), where a more knowledgeable other (MKO) scaffolds the learner’s progress (Kretchmar, 2019). While Piaget focuses on individual exploration and Vygotsky on social mediation, both emphasize developmental change and the interplay between individuals and their environment (Cole & Wertsch, 1996; Lourenço, 2012).

Papert Constructionism: A Merger of Piaget’s and Vygotsky’s Constructivism

Papert’s (1993) constructionism combines both Jean Piaget’s emphasis on cognitive construction of knowledge and Vygotsky’s sociocultural construction of knowledge and is consonant with their mutual acknowledgment of the environment as a critical interaction point

for learning (Ackermann, 2001; Fosnot, 2005; Papert & Harel, 1991). Papert worked with Piaget and was fascinated by his cognitive constructivism ideologies, with the idea of a child refining their thinking by adapting and making accommodations as they encounter new knowledge or experiences. He states:

This powerful image of child as epistemologist caught my imagination while I was working with Piaget. ... I came away impressed by his way of looking at children as the active builders of their own intellectual structures. (Papert, 1993, p. 21)

Therefore, Papert, advocating for transformation in mathematics education, extended this idea, proposing constructionism as an epistemological construct for doing mathematics, whereby learners' cognitive processes can be refined as they negotiate the computer as an educational problem-solving tool within a programming culture (Kynigos, 2015; Papert, 1972). Papert argues that computer programming environments enable students to engage actively in doing mathematics rather than simply learning about it (Papert, 1972); he further suggests that programming serves mathematics learning much like immersion in France supports learning French (Papert, 1980). This act of doing mathematics, he proposed, essentially means breaking down the barricades that have been built around mathematics over centuries to secure it as a paradigm of truth. He asserts that doing mathematics is preceded by an understanding that mathematics "truth" is fallible, refutable through conjecturing, deductive reasoning, generalization, and proof (Kynigos, 2015). Papert maintains that where a dialectical tradition of teaching mathematics is limited in conveying understanding, learners can engage in a heuristic process to understand mathematics within a programming environment. Further, Yadav et al. (2017) explain that this heuristic process involves a computational cognitive exchange such that a learner probes for problem-solving strategies, which may or may not lead to a solution. Given a

stimulating context for problem-solving, Papert advocates that computer assisted learning (CAL) be geared towards using the computer as an object-to-think-with, an instrument to help individual learners develop their computational thinking skills and intellectual abilities by programming the computer, rather than the computer instructing them (Papert, 1972, 1980).

Papert's expression of doing mathematics typifies the organizer's mindset of applying computers in the mathematics classroom, which also echoes Paulo Freire's (1996) critique of traditional educational frameworks of their time. Freire's critical pedagogy has been an influential framework of the 21st century. Freire entered the academic arena by critically analyzing his pedagogical practices as an adult educator, adopting a social constructivism perspective as he advocates for student-centered education. His critical pedagogy examines the dictatorial social structure that characterizes the political and historical climate of the 1960s to 1970s, which negatively impacts education. Freire describes this relationship within the traditional educational framework as the "banking model," where teachers possess knowledge, and students are static recipients of that knowledge. He criticizes this type of education system, stating that education is used as a tool to maintain dominance over the oppressed (Smith, 2019). He advocates for a liberating education where both the oppressed and the oppressor play a critical role in effecting change. Instead of remaining passive, the oppressed must participate in the struggle to free themselves from their oppressor. The oppressor must engage in critical self-reflection to understand their role in perpetuating historically inherited power imbalances (Mayo, 1999).

Freire's (1996) revolutionary stance, further supported by Gutstein (2012), reinforces the idea of the computer as an object-to-think-with. When a student has that power to instruct the computer through, as Papert proposed, it becomes an object of freedom. The computer as an

object-to-think-with places the learners in an empowered position to be free from their oppressor, which may also be the computer. In the preface of *Minds in Play*, authored by Kafai (2012), Papert captures the pedagogical underpinning of an object-to-think-with in the following statement:

Every educator must have felt some envy watching children playing video games: If only that energy could be mobilized in the service of learning something that the educator values. But the envy can take very different forms. Instructionists show their orientation by concretizing the wish as a desire for games that will teach math or spelling or geography or whatever. The Constructionist mind is revealed when the wish leads to imagining children making the games instead of just playing them. Rather than wanting games to instruct children they yearn to see children construct games. (p. ii)

Further, Turkle and Papert (1990) maintain that computers facilitate this constructionist mind by affording learners to use concrete thinking to develop abstract computational processes. Turkle and Papert explain that computer programming is adaptable to multiple ways of thinking, such that individuals can use their unique concrete and personal approach. Thus, whereas Piaget extended his ideas by outlining the order in which children develop, Papert focused more on how personal appropriation of the affordances of materials within a programming culture influences that (constructionism) (Kynigos, 2015; Papert, 1993). Moreover, as an object-to think-with, the computer transcends the transfer of knowledge to facilitate conceptual understanding. It equips students with transferable thinking skills that position them to think like mathematicians, acquiring the deeper meaning of mathematics as they manipulate technological tools that emerge as objects of their thinking (Papert, 1972). Concurring, Wing (2006) further explains that computers not only encourage users to instruct the computer to solve problems but also give the

user insight and cognitive advantage to solve problems in novel situations, developing computational thinking skills.

Papert's emphasis on the cultural context of learning and the affordances of languages (programming) and tools within an environment (Kynigos, 2015) resembles Vygotsky's social constructivism. Papert proposes that a child's cognitive ability is optimized and scaffolded by the sociocultural domain of computer programming (Papert, 1980). He believes that:

meanings are naturally generated in our social, intellectual and physical environment and that digital technology makes it possible for us to enrich this environment so that learners would enjoy more opportunities for the formation of meanings. (Kynigos, 2015, p. 419)

Consequently, constructionism focuses on the active construction of understanding by creating things (artifacts) that are tangible and shareable by utilizing the affordances of materials within the intellectual learning culture of programming to enhance meaning-making (Papert, 1980).

Papert believes that as learners use programming for the construction of artifacts, they become more culturally sensitive to the environment, allowing them to fluently use programming as an object-to-think-with. In signifying the cultural potency of a programming environment to solve problems, constructionism supports Vygotsky's perspective regarding the importance of the sociocultural environment (with the affordances of tools and signs) for cognitive development (Cole & Wertsch, 1996).

While Papert's proposition for the use of the computer was unpopular in the 1970s and 80s, he was not alone. During the reformatory period of education in the late 20th century, when the zeitgeist of behaviourism lingered (Kynigos, 2015) and many questioned the role of the inescapable presence of the computer in classrooms, Pea and Kurland (1984) also indicated that there are cognitive benefits to using it as a programming tool. They stated:

through learning to program, children are learning much more than programming, far more than programming “facts”... children will acquire powerfully general higher cognitive skills such as planning abilities, problem-solving heuristics, and reflectiveness on the revisionary character of the problem-solving process itself. (p. 138)

Critique of Constructivism. Constructivism is critiqued by Varela et al.’s (2016) enactivism, which posits that cognition arises through an organism’s embodied interaction with its environment. Rather than processing information from a pre-given reality, knowledge is actively constructed through perception and action. This perspective emphasizes that mind and world are co-emergent, meaning that understanding is shaped by lived experiences and sensorimotor engagement rather than solely by representation or computation. Varela et al. (2016) argue that while both Piaget’s cognitive constructivism and Vygotsky’s social constructivism recognize cognizing agents as interconnected with and influenced by their environments, they diverge in their assumptions about a “pregiven” world. Piaget posits that a child’s sensorimotor intelligence is essential for transmitting conceptual understanding, asserting that comprehension of the external world is closely tied to action. In other words, ideas, laws, logic, and concepts can only be grasped through interaction with the environment. This view aligns with Varela et al. (2016), who contend that cognitive structures emerge from repeated sensorimotor activities that guide perceptual actions. Similarly, Vygotsky (1978a, 1978b) emphasizes that learning is rooted in interactions within specific environments shaped by shared cultural and social interpretations. However, both Piaget and Vygotsky operate within a pre-given world assumption, in which representational mental structures guide the cognizing agent in achieving higher-order thinking. Piaget suggests that cognitive structures are either assimilated or accommodated into an existing external reality through logical reasoning (Varela et al., 2016),

whereas Vygotsky views a pregiven system of cultural symbols as being transferred from the external world to the internal mind (Nemirovsky et al., 2013).

Varela et al. (2016) argue that recognizing the absence of pregiven realities is crucial for understanding a cognizing agent's experience as shaped by perceptually guided action, as assuming a static world obscures the dynamic interplay between the cognizing agent and their environment, limiting our understanding of how knowledge is constructed through active engagement; this enactive approach, when applied to a mathematics programming environment, highlights how a cognizing agent's cognitive structure—emerging through interactions and responses to perturbations—reflects their thinking and actions, allowing observers to move beyond preconceived notions and focus on learning as an enacted process.

Furthermore, both Piaget's and Vygotsky's constructivist theories are primarily one-directional: they explain how a learner's cognitive structure is influenced by the environment but do not adequately address how learners' actions can shape their environments. Additionally, traditional constructivist thought often neglects to analyze cognition by exploring the coordination of actions arising from structural coupling (Towers & Martin, 2015). Hegedus and Moreno-Armella (2010) argue that the interaction between a cognizing agent and a dynamic software environment, such as programming, is a "bi-directional process that has the potential to ground and develop certain mathematical concepts" (p. 26). Therefore, they advocate for examining the co-action between the user and the environment as essential for maximizing learning in such dynamic contexts. Thus, this study's interest lies in exploring co-action between learners and their programming for mathematics learning to better understand cognition and learning in this context.

Moreover, Vergnaud (2013) critiques the ambiguity in both cognitive and social constructivism when applied to pedagogy. Cognitive constructivism, for instance, often leads to

a reductionist perspective by focusing too narrowly on the development of schemes, which centers attention on the epistemic agent, without attending fully to the psychological agent as they interact with their environment. This focus limits the exploration of how learners dynamically respond to their surroundings.

Similarly, while Vygotsky's social constructivism highlights the role of tools in mediating learning, particularly in mathematics, and emphasizes how learners adapt to environmental pressures (Bruner, 1997; Nardo, 2021), it does not clearly explain the transition from learning to development. Vergnaud addresses this gap by introducing the concept of operative knowledge, stressing that activity in context, such as gesture, selective attention, reasoning, and uncertainty management, is the primary source of development. Therefore, he proposes that active engagement in specific contexts is essential for understanding how learners develop competence and acquire new knowledge.

As outlined in the next sections, this bi-directional process is attended to by enactivism through structural coupling: world and mind emerge together in enaction (Varela et al., 2016). In like manner, Papert's constructionism acknowledges this interaction within a mathematics programming environment. He explains that learners participate in active dialogue with their environment, manipulating computation tools to solve problems. He used the metaphor of the computer as a "mathematical speaking entity" (Papert, 1980, p. 20) to describe the interchange between learners and their environment within a programming context. Thus, it appears that Papert's (1980) constructionism is an expression of enactivism. In addition to that acknowledgment, he posits that mathematics understanding is achieved through one's actions (perception and actions) in a programming environment. The action is enabled and constrained by perception from the environment itself. This perspective parallels Varela et al.'s (2016) position that perception is embedded in perceptually guided action and that cognitive structures

emerge from repeated sensorimotor patterns that determine perceptually guided action. Furthermore, Papert (1972) does not seem to subscribe to a pre-given world, as he believes that mathematics “truth” is constructed in the act of doing (in a programming environment). He notes that cognitive development is facilitated because knowledge and insight gained within a programming environment are promptly utilized for further learning as they are manifested within the context of exploration. Each band of cognitive construct initiates a means by which the learner can test their understanding, constituting mental growth (Papert, 1980).

The approach taken to integrate computer technology into mathematics teaching and learning has been of interest for several years (Grover & Pea, 2013; Hsu et al., 2018; Papert, 1980; Pea & Kurland, 1984), but now that computer technology is ubiquitously and progressively integrated into all our societal endeavours, and CT being included in mathematics curricula, there has been a proliferation of research geared toward appropriately integrating computer technology in K–12 mathematics and other subject areas to enhance CT skills (Grover & Pea, 2013; Li & Ma, 2010; Yadav et al., 2017). Current studies on integrating CT and technology in mathematics education suggest a positive impact on student understanding of content knowledge and processes (Benton et al., 2017; Forsström & Kaufmann, 2018; Gadanadis, Cendros, et al., 2017; Li & Ma, 2010; Yadav et al., 2017). However, Li and Ma (2010) observe that most studies assess the impact of application methods that do not include programming to teach mathematics, with Yadav et al. (2017) pointing out that there is a difference between using computer technology in the classroom and embedding technological tools like programming to develop CT. Furthermore, while there are many recent inquiries relating to using computing to mediate learning across different subject areas, this focus of research remains largely uninvestigated (Gadanadis, Cendros, et al., 2017; Grover & Pea, 2013).

In a discussion about incorporating CT concepts and practices into K–12 curricula, Yadav et al. (2017) highlight several educational institutions in the United States like the National Science Foundation (NSF) and reform movements such as the Next Generation Science Standards and the Common Core that are steering projects to integrate CT across disciplines to drive future economy growth. While the methods of acquiring CT skills vary with the types of projects, an overarching goal in these reform initiatives is creating and modifying knowledge. Additionally, many of these efforts acknowledge programming as a necessary skill to give learners agency to be innovative thinkers and creators, having the ability to solve problems through an iterative process (Papert, 1980; Yadav et al., 2017).

Programming in Mathematics Education

Given the potential of programming to give learners agency and the promotion of CT as a vital skill for 21st-century education (Weintrop et al., 2016; Wing, 2006), coding has been progressively included in K–12 curricula across the globe using programming tools such as Scratch, Hopscotch, and programming manipulatives (Benton et al., 2017; Yadav et al., 2017). Programming is not only regarded as a computational thinking platform but a tool to develop technology artifacts that are essential for our modern and changing society (Benton et al., 2017; Weintrop et al., 2016; Yadav et al., 2017). But still, there are many debates about whether to develop CT skills by including programming and other CT activities into mainstream education as a standalone subject or by embedding concepts and ideas into other subject areas (Benton et al., 2017; Grover & Pea, 2013; Yadav et al., 2017).

Nevertheless, to achieve the mandate of computational competency for school-age children, many countries have included coding or programming into their mathematics curricula (Misfeldt & Ejsing-Duun, 2015). This inclusion is mainly because of similarities between

mathematical and computational thinking skills (Misfeldt & Ejsing-Duun, 2015; Rich et al., 2019) and because computing is reshaping what it means to do mathematics (Misfeldt & Ejsing-Duun, 2015). In Ontario, Canada, coding is included in the algebra strand of the 2020 mathematics elementary curriculum (Grades 1–8) to develop technological fluency for problem-solving. For similar reasons, Finland, Sweden, Norway (Forsström, & Kaufmann, 2018), France, Estonia, and England (Misfeldt & Ejsing-Duun, 2015) have integrated programming into their mathematics education. In these reformed curricula, programming activities are not limited to learning generic skills to utilize everyday technologies but are specific to using computational tools to develop and implement algorithms to solve problems (Benton, 2017; Misfeldt & Ejsing-Duun, 2015).

With the developing trend of including programming into mathematics curricula comes various studies dedicated to giving pertinent information about pedagogical best practices and overall benefits. Gadanidis, Cendros, et al.(2017) posit that programming pedagogy scaffolds student mathematical and conceptual understanding in three main ways: allowing a tangible medium for abstraction, providing opportunities for dynamic modeling through automation, and offering a low floor, high ceiling differentiated environment, giving students agency for mathematical exploration. Low floor, high ceiling context provides beginners an accessible entryway to engage in activities, also providing opportunities for appropriate challenge and extensive learning (Boaler, 2016; Grover & Pea, 2013). Furthermore, Misfeldt and Ejsing-Duun (2015) observed that projects and frameworks in mathematics education that utilize programming to accomplish mathematics learning goals could be categorized under three headings: viewing students as creators, scaffolding abstract thinking, and strengthening algorithmic thinking.

The assumption that students are creators using programming for mathematics learning originates with Seymour Papert. Papert (1972, 1980) committed most of his early years to LOGO, a computer coding and mathematics environment (microworld) for children (primary and middle school). This initiative gave attention to his idea of computer programming being a powerful avenue for cognitive development in the 1980s (Misfeldt & Ejsing-Duun, 2015; Stager, 2016). Putting forward a constructionist framework, Papert believed learners should use computers to stimulate mental activities for knowledge construction as an object-to-think-with through programming to support abstract thinking. His idea is rooted in constructivism, which, like constructionism, supports learning by doing, using concrete ideas to support abstract ideas. Abstracting becomes tangible when problem solutions are represented using algorithms (writing instructions in computer-literate form). Algorithmic thinking is a crucial aspect of both computational thinking and mathematical thinking, as clear and precise instructions are the driver of doing (Misfeldt & Ejsing-Duun, 2015).

Although the introduction of programming for mathematics learning initiated a buzz, promising a radical reformation in mathematics education, Papert's (1972, 1980) ideas were not developed into mainstream classrooms for several reasons, including technological deficiency in schools, teacher competencies, and pedagogical difficulties conceptualizing programming activities with mathematics curriculum objectives (Misfeldt & Ejsing-Duun, 2015). However, as many mathematics educators, researchers, and other stakeholders champion change in mathematics education to reflect the work of mathematicians, it becomes necessary to reframe pedagogical tools for mathematics learning (Boaler, 2016). The tools used for mathematics learning, to a large extent, determine the nature of the mathematics that is enacted by learners (Ainley et al., 2006; Boaler, 2016). It is this motivation for innovative tools to teach meaningful and authentic mathematics (Misfeldt & Ejsing-Duun, 2015), along with the heightened

awareness of the importance of CT skills to future endeavors (Wing, 2006), that has shed new light on Papert's constructionist approach as a research and teaching framework for mathematics learning. Kynigos (2015) notes that "Constructionism has established itself as an epistemological paradigm, a learning theory and a design framework, harnessing digital technologies as expressive media for students' generation of mathematical meanings individually and collaboratively" (p. 417). Given this projection, with more user-friendly programming languages (Lye & Koh, 2014), research in mathematics education and many other areas such as science, engineering, and robotics (Berland & Wilensky, 2015; Catlin & Blamires, 2018; D'Amico & Guastella, 2018; Dishon & Kafai, 2020; Holbert & Wilensky, 2019; Papavlasopoulou et al., 2019; Sung et al., 2017) have been employing a constructionist framework in capacities outlined by Kynigos (2015). Many have reported that a constructionist approach facilitates the exploration of ideas, motivates autonomy, and positively impacts knowledge construction (Girvan & Savage, 2019; Joshi et al., 2019). However, despite acknowledging that programming is an essential skill for CT, there is little consensus on integrating it in specific subject areas (Benton et al., 2017; Forsström & Kaufmann, 2018; Grover & Pea, 2013).

Recent Integration of Computational Thinking in the K–12 Mathematics Classroom

Lee et al. (2020) note that recent efforts to integrate CT in science education fall along a continuum: on one end, there are software coding activities, which provide little or no support for subject area development; and on the other end, there is the integration of CT to scaffold textbook content knowledge, coupled with the integration of contemporary uses of computation within the STEM field. In mathematics education, there is the question of where to place CT integration along a similar continuum. It appears that many educators interpret learning goals relating to the integration of CT in the K–12 mathematics curriculum as standalone objectives

rather than a means to scaffold and improve understanding of existing mathematics (Gadanidis, Cendros, et al., 2017). Hickmott et al. (2018) conducted a systematic analysis of CT literature, spanning 2006–2016, intending to link mathematics education to computational thinking and found that the majority of work that involves mathematics and CT is focused on teaching standalone programming skills rather than integration of CT within specific areas of mathematics such as probability, measurement, or functions.

The tendency to isolate programming to teach CT may be because it is often unclear what CT skills entail and what programming activities are considered useful to teach them (Barr et al., 2011; Weintrop et al., 2016; Yadav et al., 2017; Zhang & Nouri, 2019). However, Benton et al. (2017) point out that the relationship between programming and CT is not lucid. They point out that now that technological resources are commonplace in schools and programming languages have advanced to reduce the functional tediousness of syntax and error messages, we can clarify the extent to which programming contributes to CT and what other knowledge might be obtained from exploring their relations. Along that trajectory, Lye and Koh (2014) analyzed 27 intervention studies within 5 years on the development of CT through programming in K–12 and higher education and reported that more similar studies should be done in classroom-type settings to assess CT practices and perspectives. They also proposed that an appropriate context for these intervention studies would be a constructionism-based environment where authentic tasks are geared towards problem-solving and information processing, and students can reflect upon their learning. However, studies indicate that within K–12 classrooms, proficiency in using programming tools and software for CT learning may be lacking (Benton et al., 2017; Yadav et al., 2017). Furthermore, different learning challenges are associated with assessing CT skills, which may be a conundrum for CT education. For example, CT perspectives (and, to a lesser

extent, CT practices) are infrequently accessed because of associated difficulties (Lye & Koh, 2014; Zhang & Nouri, 2019). CT concepts are conveniently assessed using Brennan and Resnick's (2012) suggested assessment methods (e.g., project analysis, artifacts-based interviews, and designing scenarios).

The lack of convenient ways to evaluate CT practices and perspectives may result in avoidance in assessing them, which can undoubtedly influence the extent to which they are developed (Zhang & Nouri, 2019). However, Lye and Koh (2014) provide some suggestions to assess both CT practices and perspectives. They contend that the less cognitive demanding option of visual programming languages, such as Scratch, facilitates the development and assessment of computational practices because students' programming can be viewed in animated form. They also suggest that, to examine these two dimensions better, students could verbalize their thoughts orally while programming, and their coding script could be taken and analyzed. Further, Zhang and Nouri (2019) urge researchers and educators to reflect upon assessment methods for CT to provide balanced opportunities for all aspects of CT to be advanced.

A challenge for researchers with assessment strategies is to explore programming environments and tools that are well-suited for developing all facets of CT skills (Hickmott et al., 2018; Zhang & Nouri, 2019). Consequently, studies in mathematics education (Benton et al., 2017; Buteau et al., 2017; Weintrop & Wilensky, 2015; Zhang & Nouri, 2019) now explore the implications of using programming as a platform for CT and mathematics learning. Still, in revisiting programming for mathematics learning, the results from past research regarding the impact programming plays in mathematics learning are inconclusive due to diverse outcomes (Benton et al., 2017; Misfeldt & Ejsing-Duun, 2015). For instance, early research tends to focus

on out-of-school or selective settings, highlighting the need for more attention to curriculum and teacher considerations in typical classroom contexts (Benton et al., 2017). Concerns like those expressed by Benton et al. (2017) may be validated by other studies, suggesting that careful planning is necessary to incorporate programming and CT into classroom settings to provide the right balance between student engagement and targeted learning.

Clements and Sarama (1997) observe that specially designed programming tasks that target specific learning outcomes are critical to reducing students' chances of bypassing key mathematics ideas within a classroom context. Zhang and Nouri (2019) suggest that it is necessary to consider the nature of programming tasks or artifacts in relation to the skills and subject being targeted in curriculum planning. However, designing stimulating programming tasks that are effective poses a "planning paradox" (Misfeldt & Ejsing-Duun, 2015). The planning paradox proposed by Ainley et al. (2006) addresses a situation that can be a dilemma for planners: tasks that are arranged with careful attention to learning objectives is likely to be unfulfilling for students and mathematically barren, while tasks that are centered around student engagement are likely to generate more activity but are less targeted and challenging to assess. To overcome the planning paradox, Ainley et al. recommend situating tasks within an authentic context to inspire a sense of purpose and considering the affordances of the tools within the environment. While Ainley et al.'s suggestions may offer some guidance for task designers in highly stimulating teaching and learning contexts like a programming environment, the problem of the "play paradox" observed by Noss and Hoyles (1992) is still a possibility. Play-like freedom can foster increased learning activities; however, the intended use of tools introduced into a learning environment by educators may be overlooked or repurposed for alternative objectives, complicating the assessment of learning outcomes. The planning paradox proved challenging during the early integration of programming in mathematics education, where

students often overlooked targeted mathematical concepts intended to build their mathematical capacity, thereby rendering the microworlds non-mathematical in practice (Healy & Kynigos, 2010).

In design-based research (DBR) that utilizes a constructionist approach to integrating CT through coding in K–12 education, Papavlasopoulou et al. (2019) suggest nine best practices to consider when designing coding activities: (a) how to facilitate social interaction and collaboration among learners; (b) how to implement an age-appropriate approach that will engage different groups of learners according to their characteristics; (c) how long an activity will engage learners, personally, intellectually, and emotionally; (d) how to contextualize an activity so that it is meaningful and provide relevant content; (e) What physical and digital artifacts can be included to help learners be insightful while also feeling relaxed; (f) how to supports learners' attitude and awareness; (g) how to maintain the appropriate mental challenge to prevent cognitive overload; (h) how to capture learners' interest while sustaining learning; and (i) how to maintain authentic teaching relationships with learners.

Despite observations and suggestions from researchers on how to support the teaching and learning of CT in mainstream curricula, there is clearly a need for additional research to devise ways to foster computational thinking skills within mathematics practice (Hickmott et al., 2018; Sung et al., 2017; Yadav et al., 2017) that supports meaningful learning of mathematics and CT. In their decade scope of literature analysis, Hickmott et al. (2018) unearth the following additional gaps: limited research involving educational experts; lack of concrete recommendations to K–12 educators that explicitly unite mathematics and computational thinking pedagogies; and the lack of studies linking mathematics and CT, while reporting students' mathematical achievement.

While there are gaps in research concerning the impact of programming on mathematics learning and initial studies may be inconclusive (Benton et al., 2017; Hickmott et al., 2018; Yadav et al., 2017), exploration indicates the potential benefits of using programming as a mathematics and CT learning tool. Benefits include the potential for mathematics exploration and the development of CT-related practices such as decomposition and problem-solving (Benton et al., 2017; Papert, 1972, 1980; Turkle & Papert, 1990). For instance, Clements and Sarama (1997) emphasize that with Logo, children are empowered to ascertain a deep understanding of abstract mathematics by representing and manipulating ideas concretely, thereby facilitating meaning-making. Logo initiates the use of block-based programming for novice learners, enabling the assembly of jigsaw-like programming blocks. “The types of activities supported by block-based tools draw from the constructionist tradition that emphasizes learner-directed construction and exploration and the importance of learners creating public, shareable artifacts, often in the form of artwork, games, and interactive stories” (Weintrop & Wilensky, 2015, p. 200).

Given the promise of programming to enhance mathematics learning, CT being at the forefront of 21st-century skills, and the development of more user-friendly block-based programming (such as Scratch and Alice), which enable increasing variety of activities, there are many emerging investigations exploring block-based programming for mathematics education both at the K–12 and preservice level (Benton et al., 2017; Buteau et al., 2017; Weintrop & Wilensky, 2015; Zhang & Nouri, 2019). Studies indicate that while block-based programming tools may be utilized to develop specific programming skills, they might not facilitate challenging skills enough (Weintrop & Wilensky, 2015; Zhang & Nouri, 2019). On the other hand, empirical data suggests that text-based programming might be explored to help scaffold

skills that are more challenging to developed. In conducting a systematic literature review, Zhang and Nouri (2019) determine that in addition to all the CT skills in Brennan and Resnick’s (2012) framework, the following CT skills can be developed through block-based programming such as Scratch: “input/output, reading, interpreting, and communicating code, using multimodal media, predictive thinking and human-computer interaction” (p. 18). Weintrop and Wilensky (2015) present findings from high school students suggesting that while block-based programming may be easy to use (with browsing versatility, drag-and drop-interactive component, visual clues, and natural use of language to describe blocks), text-based programming might be more powerful, and authentically represents what it means to program. Due to these findings and identifiable differences between block-based programming and conventional text-based programming, researchers in this area suggest that it is worth considering the suitability of programming tools (including robots) for targeted programming activities (Weintrop & Wilensky, 2015; Zhang & Nouri, 2019).

While there is much research targeting effective ways to support computational thinking, there remains much work to be done to understand how teachers may help scaffold CT learning in specific subject areas (Rich et al., 2019).

Preservice Teachers Scaffolding CT Integration

Expanding preservice teachers’ understanding of CT skills may help bridge the gap of infusing CT in subject areas within K–12 learning (Rich et al., 2019; Yadav et al., 2017) without compromising content knowledge of the discipline. Research has long indicated that teachers’ content and pedagogical content knowledge (Rich et al., 2019; Shulman, 1986; Toom, 2017) impact their classroom practice. Thus, Yadav et al. (2017) argue that teacher education programs should aim to equip preservice teachers with the requisite knowledge and skills to think

computationally before addressing methods of teaching their future students. Yadav et al. recommend that an opportune place to start is embedding these concepts within subject areas (alongside pedagogical practices) of specialization of future teachers. They argue that teacher education programs should prepare teachers who are skilled to integrate CT into their subject area and teaching practice so that they can facilitate the development and use of computational thinking strategies. They added that computational thinking knowledge should “allow teachers to explore core computational thinking ideas, why those ideas are central, and how computational thinking constructs are similar to or differ from other parallel concepts (such as mathematical thinking)” (p. 61). However, the knowledge of how to facilitate preservice teachers' understanding of CT constructs specific to their subject areas is lacking. One reason is that most of the recent efforts to improve teachers' capacity are geared towards professional development sessions for in-service teachers (Yadav et al., 2017). Nevertheless, much might be garnered about developing all teachers' CT pedagogical knowledge from studies targeting the scaffolding of in-service teachers' competencies.

Rich et al. (2019) believe that understanding how teachers think about computational thinking is a first step in orchestrating valuable educational experiences to aid teachers in supporting CT in K–12 mathematics and science classrooms. Rich et al.'s study was in part motivated by Yadav et al.'s (2018) advice to researchers to go beyond back-boxed methods, such as surveys, to explore the nuances involved in teachers' knowledge of supporting CT in their classrooms. Therefore, in a qualitative inquiry, Rich et al. (2019) interviewed 12 elementary school teachers to provide insight for building rich educational experiences at the preservice level. They identify applicable knowledge resources—pieces of their knowledge related to CT – indicating in-service teachers' concerns about applying developmentally appropriate ways to

integrate CT in their mathematics and science classrooms. Furthermore, Rich et al. postulate that these findings could be leveraged into preservice teacher preparation programs to ensure that they are adequately prepared to teach CT skills.

Additionally, studies aiming to develop preservice teachers' CT proficiencies are mostly outside programming environments (Rich et al., 2019; Yadav et al., 2017). However, many researchers indicate that a programming environment might be the hub for CT development (Benton et al., 2017). Thus, Zhang and Nouri (2019) suggest exploring the question of “what can be learned” through a programming environment as a starting point to make pedagogical decisions of “what to teach” for CT learning, and by extension, “how to teach.”

The Programming Learning Environment

In general, the learning environment is an essential consideration in investigating the effect of computer technology in mathematics classrooms (Benton et al., 2017; Li & Ma, 2010; Papert, 1972, 1980). The learning environment encompasses many facets apart from using technology itself (Clements & Sarama, 1997; Li & Ma, 2010; Papavlasopoulou et al., 2019; Zhang & Nouri, 2019). These include pedagogical approaches such as from a behaviourist or constructivist perspective; teaching strategies such as cooperative learning, whole-class instruction, or problem-based learning (Li & Ma, 2010); the nature of the tasks (Clements & Sarama, 1997; Zhang & Nouri, 2019); artifacts such as computational concepts and practices of the mathematics programming environment; and the characteristics of learners including their achievement, and affective factors which may be impacted by underlying factors such as gender, socioeconomic status, and education levels (Li & Ma, 2010).

Furthermore, assessing the relationship between mathematics and programming requires looking at programming as a unique culture as it opens up a low-floor computational

environment that favours, in Turkle and Papert's (1990) words, "epistemological pluralism, the validity of multiple ways of knowing and thinking" (p. 1). Papert and Turkle maintain that while the computer is promoted as the epitome of abstract and formalism, it gives learners an agency of personal appropriation. It was this low floor, high ceiling potential of the programming environment that captivated Papert's attention in promoting LOGO for mathematics learning decades ago (Grover & Pea, 2013; Papert, 1972, 1980). Grover and Pea (2013) assert that, beyond offering a low-threshold, high-ceiling, and equitable context for learning, well-designed programming environments, whether text-based, web-based, or block-based, are computationally rich, sustainable, and conducive to the transfer of learning. Computationally rich environments engage users in complex CT concepts such as task decomposition, data abstraction and generalisation, iterative and recursive thinking, logical thinking, algorithmic thinking and debugging (Grover & Pea, 2013).

Furthermore, a programming culture allows researchers to get close to the experience of learners. In the case of mathematics, a programming environment provides a powerful medium for "doing mathematics"; thereby providing observers with the opportunity to analyse learners' personal appropriation; their authentic relationship with the mathematics (Papert, 1972, 1980; Turkle & Papert, 1990). Researchers can gain insight into how learners emerge from rudimentary knowledge to abstract embodiment of knowing (Turkle & Papert, 1990). Scibner (1986) asserts that within a practical setting, the way students solve problems is situational, arising from the circumstances that constrain the environment at any given moment. Thus, many different learners may arrive at a different solution for the same problem.

A dynamic technological learning environment for mathematics, like programming, offers rich interaction between the user and their environment, which is facilitated by embodied

activities such as construction, navigation and manipulation of objects. The nature of this interaction is bi-directional, characterized by reciprocal communication, an ongoing process in which learners engage with their environment, and the environment responds (Hegedus & Moreno-Armella, 2010). Many researchers indicate this type of bi-directional interaction facilitates deep understanding of mathematical concepts (Hegedus & Moreno-Armella, 2010; Papert, 1972, 1980). In addition, from an activity approach, the rich dynamics of the context in which learners act are inclusive of their inventiveness and creativity (Rabardel & Béguin, 2005; Scibner, 1986), acknowledged by the user's employment of resources within the environment, stimulating them for action (Rabardel & Béguin, 2005). Resources or artifacts may be physical or mental symbols that pivotally shape problem-solving (Rabardel & Béguin, 2005). The rich dynamic imposed by the learning context of programming is an essential consideration for research in resolving questions relating to the impact of programming on mathematics learning (Benton et al., 2017; Hegedus & Moreno-Armella, 2010; 2017; Shvarts et al., 2021). Hegedus and Moreno-Armella suggest that in this educational technology paradigm, it is worth analyzing the dynamics of use by considering the nature of co-action, which speaks to the interaction between the user and the operating environment.

Chapter Summary

This chapter establishes the significant role of CT in problem-solving and its integration into mathematics curricula worldwide, despite the lack of a universally accepted definition. CT is generally conceptualized as a thought process, a problem-solving approach, and a set of educational practices essential for schooling. The literature supports the idea that CT emerges within programming environments, where learners develop public and shareable artifacts.

Similarly, MT evolves through engagement in mathematical activities, fostering essential process skills such as making connections, communication, and problem-solving.

The review highlights that while constructivism does not fully address the bidirectional nature of learners interacting with dynamic programming environments for mathematical learning, constructionism (Papert, 1972, 1980) describes such environments as “mathematical speaking entities,” emphasizing their interactive nature. Additionally, research suggests that programming provides a powerful lens for observing learners’ personal appropriation of mathematical concepts as they engage with such environments. However, more studies are needed in classroom contexts where programming is used for CT learning.

To address this gap, this research explores the nature of co-action within a bidirectional framework like enactivism. This perspective may enhance our understanding of how CT and MT concepts are constructed through engagement with dynamic digital environments. The next section discusses the conceptual framework guiding this study.

CHAPTER THREE: THEORETICAL AND CONCEPTUAL FRAMEWORK

This section explores enactivism as the theoretical foundation of this research and situates it within a broader conceptual framework that integrates insights from various scholars on cognition, computational thinking, and mathematical reasoning. Enactivism posits that cognition is not a passive representation of a pre-given world but an active, embodied process that emerges through an agent's sensorimotor engagement with the environment (Varela et al., 2016). This study extends enactivist principles to a programming environment, where agents co-emerge through their interaction with their environment.

The framework is further enriched by Hegedus and Moreno-Armella's (2010) notion of "hot-spots," which conceptualizes tools as mediators between users and their environment through interaction. Kieren et al.'s (1997) concept of "occasioning" complements this perspective by emphasizing how learners actively select and act upon elements in their environment, altering both their understanding and the environment itself. Brennan and Resnick's (2012) dimensions of computational thinking intersect with Burton's (1984) and Mason et al.'s (1982, 2010) perspectives on mathematical thinking, which highlight problem-solving as a process of iteration, reflection, and expression.

Stigmergy provides the theoretical basis for identifying components of the emerging system. It explains how agents interact with and modify their programming environment through a decentralized, self-organizing process, where traces left in the environment guide further perception and action.

By synthesizing these perspectives, this framework captures the co-action of agents and their environment, emphasizing the emergent and dynamic nature of learning within programming activity.

Enactivism

Enactivism, or the enactive approach to cognition, inspired by Merleau-Ponty, challenges the representationalist view, which frames cognition as the recovery or projection of a pre-given world. Merleau-Ponty was a prominent philosopher in the post-war existentialist and phenomenological movement. His work critiques the dualistic views of the mind-body relationship held by several philosophers prior to the 20th century (Carman, 2020).

Dualism, originating with Descartes (1596–1650) and later developed by philosophers such as Immanuel Kant (1724–1804), approaches cognition from an epistemological standpoint that treats the mind as superior to and separate from the body. According to this perspective, reality lacks a biological foundation and cannot be studied scientifically; instead, it is interpreted solely through mental processes (Carman, 2020). In mathematics education, this mind-body separation influenced the way conceptual understanding was framed for centuries, with mathematical objects and problem-solving algorithms being treated as mental images divorced from embodied experience (Shvarts et al., 2021).

Influenced by a lineage of phenomenological philosophers, beginning with Husserl (1859–1937) and continuing through Heidegger (1889–1978) and Sartre (1905–1980), Merleau-Ponty rejected dualistic thinking. His work was primarily shaped by the ideas of Edmund Husserl and Martin Heidegger. Husserl, regarded as the founder of the phenomenological movement, introduced transcendental phenomenology, which is based on the premise that preconceived ideas must be set aside (bracketed) in order to perceive phenomena through an unfiltered lens. This approach allows for an accurate and justifiable view of the world to emerge naturally, revealing the essential structures of consciousness and enabling the exploration of rational interconnections (Carman, 2020; Wheeler, 2011).

Husserl, a student of Franz Brentano, developed his phenomenological approach by building on Brentano's concept of intentionality, the idea that all mental states are directed toward an object, whether or not that object exists in the empirical world. Brentano emphasized that the mind inherently represents the world, making intentionality a defining feature of consciousness. Husserl extended this idea by proposing that, through the method of epoché or bracketing, suspending judgment about the natural world, one can access the underlying intentional structures of experience. By setting aside the "natural attitude," or the uncritical acceptance of the world as it appears, Husserl believed it was possible to uncover the essential structures of consciousness. This process, he argued, opens a realm of knowledge that is foundational and precedes empirical science (Carman, 2020; Varela et al., 2016; Wheeler, 2011).

However, Husserl's initiative faced criticism. Varela et al. (2016) argue that Husserl encountered challenges because he did not recognize the inherent circularity of human existence—a core concept in enactivism—which involves a constant interplay between our physical, outer structure and our inner, phenomenological experience. Varela et al. (2016) explain that we move back and forth between these structures to make sense of our experiences, highlighting an inseparable connection between body and world. Reflection, as proposed by Husserl, invariably returns us to our embodied experience, which is deeply embedded in our social and cultural contexts (Varela et al., 2016).

In contrast, Husserl's view—based on the notion that cognition is independent of the world—was also critiqued by Heidegger and Merleau-Ponty. Heidegger approached philosophy by examining the phenomenon of "being," particularly the unique way humans exist within the world, challenging Husserl's idea that lived experience can be separated from one's social and cultural context. Merleau-Ponty, while initially influenced by Husserl's emphasis on essences

and Heidegger's focus on "being" as an activity embedded in the world, ultimately diverged from both. He argued that neither philosophy struck the right balance between empiricism and intellectualism (Carman, 2020; Wheeler, 2011). Merleau-Ponty (1962) captured this middle ground in *Phenomenology of Perception*:

In the first case consciousness is too poor, in the second too rich for any phenomenon to appeal compellingly to it. Empiricism cannot see that we need to know what we are looking for, otherwise we would not be looking for it, and intellectualism fails to see that we need to be ignorant of what we are looking for, or equally again we should not be searching. (p. 28)

Merleau-Ponty argued that the outer world—which includes the physical and natural environment, science, and our social background—cannot be separated from our inner world of lived experience and perception. He believed that both science and phenomenology help us understand our existence by first engaging with the world directly and then reflecting on those experiences to give them meaning (Varela et al., 2016, p. 19).

Similarly, the enactive approach seeks to balance these two perspectives: realism, which sees knowledge as coming only from the outer world, and idealism, which views knowledge as shaped entirely by the inner world of the mind. Instead of treating them as separate, the enactive approach, like Merleau-Ponty's view, emphasizes that understanding emerges from the continuous interaction between body, mind, and environment—bridging the outer and inner worlds.

Cognitivism provides a framework for understanding cognition as a process driven by mental representations and computational procedures. In general, cognitivism postulates cognitive processes are governed by the subconscious through a representational-computational

system that cannot be brought to consciousness. There are various theoretical applications in accordance with different types of symbolic structures (e.g., rules, concepts, images, analogies, combined structures, or from a connectionist perspective) that govern how the mind works; but, in general, the consensus is that cognitive computational procedures operate on mental representations (Thagard, 2005). Merleau-Ponty and the enactive approach perspective challenge the idea that knowledge is solely based on symbolic representations, instead highlighting the role of embodied experience in shaping how we perceive and make sense of our environment.

Both Merleau-Ponty (1962) and Varela et al. (2016) propose that the body is the primary medium for perception, positioning it as central to bridging the dual extremes of realism and idealism. For both thinkers, the mind and body are inseparable, with the body actively engaging with the world, challenging the division between subject and object. Merleau-Ponty (1962) explains that our body exists in the world like a heart within an organism: it continuously enlivens the visible world, infuses it with vitality, sustains it from within, and together they form an interconnected system.

In parallel, Varela et al. (2016) approach enactivism by recognizing this embodied state, viewing the body as both a physical, cognitive mechanism (biological) and a lived, experiential structure (phenomenological), thus integrating the outer and the inner. For Varela et al., ideas and concepts emerge in close relation to external interactions. In mathematics education, this embodied approach reconceptualizes mathematical concepts as distributed sensorimotor activities, arising from actions with environmental artifacts (Shvarts et al., 2021).

Varela et al.'s (2016) enactivist perspective counteracts today's representationalist (and even connectionist) views in cognitive science in a similar way that Merleau-Ponty's (1962) thinking contradicts the whole philosophical tradition of his time. Today's cognitive science is

heavily influenced by cognitivism. Cognitivism ignores the fundamental circularity of a midway between self and world, which is acknowledged by both Varela et al. (2016) and Merleau-Ponty (1962). While Merleau-Ponty identified the circularity, he did not explore it. Merleau-Ponty focused, instead, on how the act of knowing occurs through perception. For him, action is achieved by the body's consistent effort to get an optimal grip of the perceived in the circular state of embodiment. He believed that "the perceived is actually a reality, an epiphenomenon that is located beyond questions of truth and falsehood" (Lippi, 2016, p. 94). Moreover, Varela et al. (2016) indicate that, like Heidegger, Merleau-Ponty's (1962) attention to the pragmatic dimension of experience was purely theoretical.

On the other hand, enactivism was born out of a dedication to investigating this midway by applying science and experience (Varela et al., 2016). In acknowledging the midway between outer and inner—the body as both a physical structure and lived experiential structure—Varela et al. (2016) looked to the Buddhist tradition of mindfulness/awareness to understand human experience. This research will not explore mindfulness/awareness but is grounded in the relationship of structural coupling established by enactivism in explaining cognition through embodied action (Varela et al., 2016).

Embodied Action

According to enactivism, cognition does not exist independently of the body or reality beyond its own history; it is neither a mere reflection nor a projection of the external or internal worlds. Rather, cognition is embodied action, representing a dynamic and dialectical relationship between the body and its environment, both social and physical. Varela et al. (2016) clarify two key points on "embodied" cognition: (a) the entire body, with its sensorimotor capacities, is integral to cognitive experience, and (b) these sensorimotor capacities are embedded in a wider

biological, psychological, and cultural context. Varela et al. further emphasize that “action” in “embodied action” conditionally combines sensory and motor processes, which co-evolve in individuals.

In this framework of embodied action, enaction is understood through two essential components that illuminate how a perceiver guides their actions in a given situation:

1. Perception as Embedded in Perceptually Guided Action: Perception is inherently active; without action, perception cannot occur (Reid & Mgombelo, 2015).
2. Cognitive Structure as Emerging from Sensorimotor Patterns: Cognitive structures develop through repeated sensorimotor interactions, shaping perception and guiding action (Varela et al., 2016). Reid and Mgombelo (2015) capture this as, “Without a feedback loop of action and perception, perception does not occur” (p. 172).

These concepts will be further elaborated upon through the enactivist notion of structural coupling.

Structural Coupling

Structural coupling is a central concept in enactivism that provides a key theoretical foundation for this research. Maturana and Varela (1987) introduced structural coupling to describe the ontological dynamics through which a living system, or unity, interacts with its environment to produce intelligent behaviour. Specifically, cognitive structure emerges as a living being’s sensorimotor system interacts with its environment through continuous mutual influence.

In this framework, a system’s structure refers to its components and their configurations, while its organization encompasses the relationships essential to classify the system within a specific category (Maturana & Varela, 1987). A living system is identified by its autopoietic

organization, meaning it has the self-producing, self-organizing capacity that characterizes autonomy. Self-organization implies that interactions are shaped not by external influences alone but by the system's historical and organizational characteristics. Through this autonomy, a living unity (cognizing agent) engages with its environment, enacting a world through a history of structural coupling.

Structural coupling suggests that a cognizing agent and its environment are interconnected, undergoing co-ontogenetic structural changes through mutual influence. Both agent and environment evolve as a single system over time, with the agent using its emergent perspective to continually redefine the environment, which in turn impacts the agent's system. In this process, the agent's actions are perceptually guided by its self-organizing sensorimotor capabilities, allowing it to engage with the environment through recurrent feedback loops of action and perception. The system retains a history of encounters, drawing on this background in its ongoing interactions. The history of a living unity is thus shaped both by its internal dynamics and its environmental interactions (Varela et al., 2016).

Interestingly, while a living system's structure may continuously change through structural coupling, its identity is maintained through its autopoietic organization. If this organization is not preserved, the system disintegrates and becomes something different. However, if only the structure changes while the organization remains intact, the system retains its identity; it simply transitions into a different state (Maturana, 2002). For living systems, self-preservation is sustained through an autonomous, closed sensorimotor network, where local operations generate global internal behaviours (Varela et al., 2016). This self-organizing capacity enables the system to continuously produce the components essential to its identity as a living unity. Varela (1992) describes this network as simultaneously realizing the system as a tangible

spatiotemporal unity, thereby enabling the organization of its components. Reid and Mgombelo (2015) explain this dynamic by noting that “the system consists of components that interact, and those processes of interaction are recursive; they reconfigure the system in ways that allow those processes to continue” (p. 173).

Paradoxically, though having self-organizing potential, a living system retains its identity through interaction with the environment. The system must specify itself as separate from its environment; yet it must remain linked to it because it is by coupling that the system acquires its emergent state of the cognitive self. More precisely, by their sensorimotor capacity, living beings distinguish themselves as a unity, but at the same time is one system with their environment because it is the very environment that enables perceptually guided action that allows them to retain their cognizing identity (Varela, 1992). Reid and Mgombelo (2015) note that, owing to its autopoietic nature, this interestingly paradoxical relationship often causes some commentators in mathematics education to critique enactivism. However, the article “Survey of Key Concepts in Enactivist Theory and Methodology” (Reid & Mgombelo, 2015) clarifies the misconception that enactivism subordinates social interaction. They argue that how a living unit interacts is clearly described, highlighting that Varela and Goguen (1978) emphasize that the organizational closure does not equate to interactional closure. Reid and Mgombelo (2015) point out that every living being must maintain unceasing interaction with its environment, which impinges perturbation; that is, interaction is a necessary condition for autopoietic characteristic.

Moreover, when a living unit couples with its sociocultural environment, the equilibrium is slightly in favour of the living (with its autopoietic identity) because it has an active role in the entanglement. As mentioned earlier, the living being defines itself as a unity, and at the same instance, it also defines what remains external to it, that is, its environment surrounding it. The

autopoietic unit forms a perspective from which the surrounding is one, a perspective that cannot be mistaken with the physical environment as it is observed by an external being (Varela, 1992). Furthermore, Varela et al. (2016) explain that mind and environment are simultaneously enacted at a basic level of categorization, where an object is perceived as offering a certain kind of interaction. At this level of categorization, a cognizing agent uses the object in the afforded way. Form and function coordinated for the same purpose, and the activities achieved by the cognising agent “with basic-level objects are part of the cultural, consensually validated forms of the life of the community in which the human and object are situated—they are basic-level activities” (p. 177).

At this point, I must reiterate that the structural coupling of an autonomous system with its environment is antagonistic to how an input/output formulaic system operates. In the latter situation, the world is pre-given. All variables, properties, and processes are specified or prescribed, and symbolically represented in the mind (Varela et al., 2016). The intentional representation states are processed subconsciously through computation and subsequently brought into consciousness. In contrast, under structural coupling, an autonomous unity defines its own world of significance. This significance emerges as a difference between the environment devoid of reference to the autonomous unity and the environment that the system constructs for itself (the system’s world). The system’s world, which is not pre-given, exists solely through the mutual enactment of mind and world (Varela, 1992). Intentionality arises in the act of enactment, during the “surplus of signification,” when the perceiver (the system) is cognizing in response to perturbations from the environment. When perturbed, the cognizing agent (aware of something) must assess the encounter—valuing it (liking, disliking, or ignoring) and choosing a response (attraction, rejection, or neutrality), which in turn generates intention.

As illustrated, the actions of the cognizing agent are intertwined with the environment through perception, but the world of significance is ultimately determined by the cognizing agent (Varela, 1992).

It is the continuous bringing forth of signification (imbued with intentionality) that drives action, for, in signification, there is a constant lacking in the perceiver, whose action is to satisfy that lacking. The cognizing agent must always give to the coupling as a functioning whole (Varela, 1992). For example, in a mathematical problem-solving context, the lacking would arise in doing mathematics, in the desire to find convenient, personal, and comfortable ways to solving a task (Shvarts et al., 2021). Further, Varela (1992) explicates this “surplus of signification” using biological terms:

There is no food significance in sucrose except when a bacterium swims up-gradient, and its metabolism uses the molecule in a way that allows its identity to continue. This surplus is obviously not indifferent to the regularities and textures (i.e., the “laws”) that operate in the environment, such as the ability of sucrose to create a gradient and traverse a cell membrane. On the contrary, the system's world is built on these regularities, which ensure that it can maintain its coupling at all times. (p. 7)

Translating this example to the context of doing mathematics in a programming environment, mathematical concepts do not inherently possess meaning; rather, they gain significance through interaction and application. Just as sucrose is not intrinsically “food” but becomes meaningful when a bacterium detects and metabolizes it for survival, mathematical symbols, functions, and algorithms only acquire relevance when a learner actively engages with them through coding. Learners (cognizing agents) create a world of significance through their actions, using their sensorimotor capacities to interact with the environment in response to perturbations from the programming context that ignite their curiosity. Reid and Mgombelo (2015) note, “things get

interesting when there are recurrent patterns of triggering and being triggered that result in structures that allow these patterns to persist” (p. 172). From these recurrent patterns, the cognizing self (i.e., the learner or programmer/mathematician) emerges, maintaining its autopoietic identity as a cognizing agent. Simultaneously, the learner’s actions contribute to a series of engagements that define their overall experience in programming and mathematics. The programming environment provides structured rules and constraints, such as syntax, data structures, and logical operations, similar to the physical laws governing a bacterium’s interaction with its surroundings. These actions are structurally determined; while the learner’s cognitive structure is perturbed by the environment, their responses are shaped by prior experiences and an evolving history of structural coupling. As Varela (1992) states, “what the autopoietic system does—due to its very mode of identity—is to constantly confront the encounters (perturbations, shocks, coupling) with its environment and treat them from a perspective which is not intrinsic to the encounters themselves” (p. 7). Meaning, therefore, is not passively received but actively constructed through iterative engagement with the programming environment, where mathematical reasoning emerges through continuous interaction and adaptation.

A profound example is presented in Reid and Mgombelo’s (2015) explanation of more complex living and cognizing systems:

The motion of a billiard ball struck by another billiard ball is sometimes seen as determined by the force and direction of the ball striking it, but it is actually determined by the structure of the ball being struck. If the ball had the structure of a tennis ball, it would move very differently. The ball striking the other ball provides energy, but the structure of the ball being struck determines what happens to that energy. (p. 173)

The learner engaging in mathematics within a programming environment is not indifferent to the semantics that govern the programming language. The learner's world of significance is shaped by the syntax that imparts meaning to the programming language. If the learner is unaware of the language's syntax, effective coupling may not be possible, but the nature of the interaction is determined by the learner's structure. Drawing from Gibson's (1986) ecological perspective, Shvarts et al. (2021) explain that within a complex system, such as learners interacting with programming, the environment offers possibilities for action (affordances) while simultaneously imposing constraints within the system. Moreover, the affordances are complemented by the capacity of understanding due to the sensorimotor system, which Shvarts et al. (2021) express as "body potentialities—needed to recognize and act with affordances" (p. 452). The affordances of the programming environment and the learner's capacity to understand both mathematics and programming frame coupling. The structural properties of both the body potentialities and environmental affordances consequentially constitute actions, which are steered by a web of intentionality (Shvarts et al., 2021). Learners, therefore, enact a world of mathematics significance by acting on perturbations from the programming environment, different from the codes as seen by an observer. In that regard, understanding a learner's experience in a programming context, therefore, equates to understanding a perceiver's sensorimotor actions in relation to perturbations from the environment that result in the learner's programming activity. Thus, an observer can gain insight into a learner's conceptual understanding of mathematics and programming by exploring how they structurally couple with their environment.

In summary, structural coupling takes into account the current state of the living system, the environment, and the actions that a cognizing agent undertakes in response to perturbations.

There is never a predefined world; rather, the system continuously enacts a world through its history of structural coupling. From this perspective, knowledge is not derived from recovering or representing a world through lived experiences but is instead created by those lived experiences within an enacted world. Maturana (2002) suggests that we can only truly understand a system within its environment when we abandon the notion of a pre-given world and consider how the system utilizes its structure to couple with its environment; as he states:

In 1965, when I was studying color vision in pigeons, I realized that I could no longer pretend that one saw colors as features of an external world. I had to abandon the question, ‘How do I see that color?’ and instead ask, ‘What happens in me when I say that I see such a color?’ To make this change meant abandoning the notion that there was an external independent world to be known by the observer. Instead, I had to accept that knowing has to do with the congruent interactions between entities, each of which is a structure-determined system—that is, a system in which all that happens with it and to it is determined at every instant by the way it is made (its structure) at that instant. (pp. 5–6)

Co-Action Within a Mathematics-Programming Context

Many researchers indicate that interesting dynamics emerge between learners and their environment when they engage in mathematics within a software culture, leading to deep mathematical learning (Alqahtani & Powell, 2016; Hegedus & Moreno-Armella, 2010; Papert, 1972; Shvarts et al., 2021). At the same time, they observe that the nature and potential of this interaction have not been fully explored (Alqahtani & Powell, 2016; Hegedus & Moreno-Armella, 2010; Shvarts et al., 2021). The instrumental approach, a widely used framework for examining the potential of ICT in mathematics education (Misfeldt & Ejsing-Duun, 2015;

Shvarts et al., 2021), does not sufficiently address the role of the environment in the dialectical relationship between cognizing agents and their surroundings (Hegedus & Moreno-Armella, 2010; Nemirovsky et al., 2013; Shvarts et al., 2021). Originating from cognitive ergonomics, the instrumental approach posits that through a heuristic process, a learner utilizes the affordances of an artifact, transforming it into an instrument via a process known as instrumental genesis (IG). This approach differentiates two interrelated sub-processes: instrumentation, which focuses on how tools shape the learner, and instrumentalization, which examines how the learner shapes the tool (Verillon & Rabardel, 1995).

The instrumental approach has been applied in mathematics education, with researchers, educators, and designers emphasizing its epistemic and pragmatic value in fostering conceptual understanding. On the epistemic level, the approach is used to explore knowledge that emerges from the dialectical process of IG. On the pragmatic level, stakeholders emphasize knowledge that can be utilized in instrumented activities (Shvarts et al., 2021). While acknowledging the instrumental approach's role in facilitating technology integration in mathematics, Shvarts et al. (2021) critique it for treating cognition and intentionality from a dualistic, representationalist perspective, relying on schemes and viewing mathematics as primarily a mental activity. Specifically, Nemirovsky et al. (2013) question the feasibility of a dualistic view between instrument-mediated action and schemes, which mathematics education scholars regard as central to mathematical conceptual understanding. They argue that a critical aspect of mathematics learning involves the fluent use of tools, integrating perceptual and motor aspects (perceptuomotor integration) of the learner's activity with the tool, much like how the act of playing the piano intertwines the movement of the musician's fingers, the sound produced, and other bodily engagements.

Shvarts et al. (2021) also critique the notion of schemes regulating behaviour through weak anticipation, arguing that when the body and environment interact through recurrent engagement, behaviour is regulated by strong anticipation. They suggest that understanding the embodied nature of the relationship between learning and technology can deepen insights into the body's role in "regulating instrumented actions and acknowledging new developments in educational technology and cognitive science" (p. 449). This proposition is supported by Verela et al.'s (2016) view that "cognitive structures emerge from recurrent patterns that enable action to be perceptually guided" (p. 173).

In assessing the dialectical relationship of the instrumental approach, Alqahtani and Powell (2016), along with Hegedus and Moreno-Armella (2010), posit that significant insights into integrating technology into mathematics education can be gained by exploring the co-action of learners within a dynamic software environment. Hegedus and Mreno-Armella (2010) state:

We propose that as users (or students), we redefine the exploration space through new forms of action and interaction; thus, we need to enhance the framework of instrumental genesis to accommodate the changing nature of technology, particularly the evolving actions of users, as a cultural dimension. Users actively modify the status of the object of inquiry through the examination and executability of the environment, as facilitated by their interactions. (p. 26)

Similarly, Alqahtani and Powell (2010) recommend that by extending the instrumental approach to examine how users respond to perturbations in a dynamic software environment, researchers may gain further insights into how technology shapes mathematics learning, as simply teaching mathematics with technology does not guarantee improved achievement. Moreover, in exploring the collective nature of emergent mathematical understanding, Towers

and Martin (2015) investigate the concept of coaction to examine interactions among learners during collaborative mathematics problem-solving. In this context, Martin and Towers (2009) define coaction as the coordinated activity of learners

through which mathematical ideas and actions, initially stemming from an individual learner, are taken up, built upon, developed, reworked, and elaborated by others, thus emerging as shared understandings for and across the group, rather than remaining isolated within any one individual. (p. 4)

Despite the semantic differences, both “co-action” (Hegedus & Moreno-Armella, 2010) and “coactions” (Martin & Towers, 2009) convey the nature of interaction whereby actions are reciprocated in response to perturbations. The singular form of Hegedus and Moreno-Armella’s (2010) co-action refers to the general nature of the interaction between a cognizing agent and the environment, whereas Towers and Martin’s (2009, 2015) coactions narrate the evolution of interaction over time. This study adopts Hegedus and Moreno-Armella’s (2010) concept of co-action to refer to the general nature of agent-environment interactions; however, unlike Hegedus and Moreno-Armella (2010), this research considers the relationship in co-action to be non-symmetrical, favouring the cognizing agent. While Hegedus and Moreno-Armella acknowledge the active role of cognizing agents in continually redefining their local context, they maintain that the relationship is symmetrical. This research embraces Hegedus and Moreno-Armella’s definition of co-action as indicative of the interaction’s nature, while rejecting the notion of symmetry. Users may redefine their environment, but their cognitive structure determines how they act and is modulated by that environment (Varela et al., 2016); thus, the relationship is not symmetrical. The advantage in equilibrium lies with the living system, which maintains an active role in this reciprocal coupling.

While Hegedus and Moreno-Armella (2010) and Varela et al. (2016) differ on the symmetrical nature of co-action, both recognize it as a necessary precursor for understanding cognitive processes. Pickering (2013), like Hegedus and Moreno-Armella (2010), regards co-action—referred to by him as the dance of agency—as symmetrical; however, his description of it as “a performative back and forth between the human and non-human” informs this research (p. 78). Pickering (2013) explains that the interaction “between people and things” is crucial for understanding scientific processes, as “we engage with the environment performatively” (p. 78). He asserts that human action or agency in achieving specific goals is conditionally influenced by the environment's ongoing response to that agency. He suggests that while the overarching goal of engaging in activity may remain constant, the process of co-action—the dance—and the resulting outcomes are emergent. In the context of this research, while a learner’s objective in designing a mathematical learning or exploratory object for a particular purpose initiates interaction with the environment, the mathematical/programming process embodies the dance of agency: the co-action between the programming environment and the learner’s agency. Pickering emphasizes that since agency “is emergent in practice in a brutal sense, we need to think instead about forward-looking evolutionary processes, dances of agency that explore a world of endless emergence and becoming” (p. 78). This study responds to such recommendations.

Co-Action With Tools

Several authors acknowledge the importance of tools or artifacts in mediating activity within a learning environment (Nemirovsky et al., 2013; Shvarts et al., 2021; Towers & Martin, 2015; Vergnaud, 2009; Vygotsky, 1978a, 1978b). In their discussion, Hegedus and Moreno-Armella (2010) explore how tools within dynamic geometry environments (DGEs), specifically hot-spots, impact the interaction between users and their learning environment. Aiming to

enhance the framework of instrumental genesis, they propose that changes within a DGE are inseparable from users' actions with hot-spots, thereby conceptualizing tools as connectors between users and their environment through interaction. They explain that hot-spots are utilized to create objects, which become integrated into the infrastructure of the environment, establishing a set of rules that constrain the nature of co-action between users while providing feedback. Agency, they assert, emerges from a collaboration between the user and the environment, with both acting and reacting upon one another.

Furthermore, Hegedus and Moreno-Armella (2010) propose that hot-spots “offer an ‘invisible hand’ that projects the intentionality of the designer of the environment into the actions of the user” (p. 27). Their characterization of tools as infrastructural is particularly relevant when examining a cognizing agent's actions with tools in a programming environment in this study. I conceptualize the infrastructure of a programming environment, comprised of tools like Integrated Development Environments (IDEs), compilers, libraries, and interpreters, as consisting of the components and rules developed by designers to support code development, exploration, and collaboration. Thus, infrastructure represents the software of a specific programming environment, created through co-action with designer-agents who generate elements prompting other agents interacting with the software to either accept, reject, or remain neutral toward these prompts.

However, although Hegedus and Moreno-Armella (2010) describe actions with hot-spots as embodied action, they appear to separate the hot-spot as a tool from the mental activity associated with its use, akin to the instrumental approach that distinguishes between artifacts and mental schemes (Rabardel & Beguin, 2005). They explain that the embodied actions of dragging hot-spots to construct geometric objects in a DGE facilitate a form of “semiotic mediation”

between the object and the user, who is attempting to understand or manipulate the diagram (p. 29). Semiotic mediation, a concept proposed by Vygotsky (Wells, 2007), differentiates between tools that mediate physical activity and signs or symbols that mediate psychological activities. In contrast, the enactive approach views the physical form and function of tools as integrated aspects of the same process, with the cognizing agent responding to their coordination (Varela et al., 2016). This perspective emphasizes that tools and cognitive processes are deeply interconnected rather than being distinct or opposing attributes. Therefore, while the infrastructural nature of tools in Hegedus and Moreno-Armella's (2010) article is instructive for considering elements of the programming environment that might prompt learners into action, it is necessary to view tools through an enactivist framework.

Shvarts et al.'s (2021) description of tools as part of a functional system provides additional insight that aligns more closely with enactivism. They suggest that tools are integrated into the entanglement of agents coupling with the environment through a non-centralized multilevel functional system, such that the cognizing self (with a tool) becomes a "body-artifact system," extending the agent's bodily potentialities. They explain that these body-potentialities, when coupled with the tool, transform the nature of the co-action with the environment during interaction. They argue that the brain does not centrally regulate tool use; instead, the interaction between the user and the environment adapts and regulates the use of tools. This is evidenced by individuals quickly learning to accurately estimate spaces and navigate effectively with a wheelchair after a brief trial, suggesting that understanding and use are developed through practical interaction with the environment rather than through pre-existing mental schemes. While this study does not fully endorse the extended mind perspective that tools are incorporated into the body, tools are viewed as part of a functional system in which the cognizing agent plays

a role. The cognizing agent, as an autopoietic unity, specifies itself and what remains external to it, including tools (Varela, 1992). Nevertheless, as a self-organizing system, the living must be linked to tools because it is the affordances of the environment that enable coupling.

While Shvarts et al.'s (2021) body-artifact system and Hegedus and Moreno-Armella's (2010) hot-spots are helpful in explicating tools, this research distinguishes tools as "occasioning" tools from an enactivist mindset. Kieren et al. (1997) argue that selecting elements from the environment, referred to as occasioning, is essential for "bringing forth" or enacting a world; they explain:

Occasioning occurs as the person selects elements from the environment and acts on them, thus changing the environment and in fact bringing forth a world; but also, the history of the person's mathematical activity is changed, and thus his or her understanding (structure–schemes) is altered. (p. 628)

Building on this foundation, this research conceptualizes occasioning tools as those specified by the cognizing agent in response to perturbation during mathematical or programming activities, thereby enacting a world of significance through structural coupling. These tools may originate from the infrastructure of the programming software or emerge as agents engage with the programming environment for mathematics learning. In other words, occasioning tools arise from coupling and are components of an evolving complex system (Shvarts et al., 2021; Varela et al., 2016) in which both agents and the environment actively modify structure. Recognizing that the dynamic nature of a coupling system precludes a precise definition of occasioning tools, this study adopts constructs from the literature that align with its context to begin conceptualizing these tools. The researcher acknowledges that tools may emerge in novel ways as participants engage in co-action, necessitating redefinition. Generally, tools are viewed as

computational or mathematical concepts, techniques, practices, or attitudes that a cognizing agent employs in response to perturbations from the programming environment.

For computational occasioning tools, Brennan and Resnick's (2012) three dimensions of computational thinking, computational concepts, computational practices, and computational perspectives, are utilized to begin conceptualizing how these tools are specified by the agent in response to environmental prompts. Brennan and Resnick identify three key skills that students develop through programming tasks:

1. *Computational Concepts*: Fundamental ideas such as sequences, loops, parallelism, events, conditionals, operators, and data, which serve as building blocks for designing programs.
2. *Computational Practices*: Habits of thought in programming, including being incremental and iterative, testing and debugging, reusing and remixing, and abstracting and modularizing.
3. *Computational Perspectives*: The evolving attitudes and viewpoints individuals develop about computing, its applications, and its impact on the world.

In this study, these dimensions of computational thinking are regarded as initial occasioning tools that continuously emerge as the agent interacts with and responds to the programming environment.

Regarding mathematical thinking, Burton (1984), alongside Mason et al. (1982, 2010), presents an understanding of mathematical thinking that aligns favourably with the enactivist framework of this study and is therefore adopted to begin conceptualizing mathematical occasioning tools. Burton (1984) proposes that mathematical thinking involves the operations, processes, and dynamics that arise during engagement in mathematical activities with a goal-

oriented purpose to enhance understanding and extend control over one's environment. Operations can range from well-recognized mathematical procedures like addition and subtraction to less conspicuous cognitive activities such as enumerating, repetition, or iteration, identifying relationships between elements (e.g., ordering, correspondence, equivalence, inverse, converse), and transforming elements from one state to another through combination or substitution.

Furthermore, Burton (1984) suggests four central processes in mathematical activity, as proposed by Mason et al. (1982, 2010): specializing, conjecturing, generalizing, and convincing. Specializing entails finding specific examples that support a problem, while conjecturing about relationships naturally follows when numerous examples are examined. Generalizing is the process through which learners seek to derive order and meaning from overwhelming data. The inductive processes of specializing, conjecturing, and generalizing can also be viewed deductively, where a cognizing agent begins with a generalization, explores subsequent conjectures prompted by it, and tests them using specific examples. Convincing, in both cases, involves a process of persuading oneself and others through doubting, probing arguments, questioning assumptions, and negotiating meanings toward the best possible proof in the given circumstances (Burton, 1984).

Mathematical dynamics refers to the iterative process of mathematical thinking, highlighting how surprise, tension, and curiosity drive cognitive activities. This dynamic process involves the development or modification of mathematical systems or concepts through cycles of manipulation and pattern recognition. It encompasses the ongoing interaction between variables in mathematical models and the identification of trends, where initial vague understandings

evolve into more refined insights through continuous exploration and adjustment. As Burton states:

A perceived gap between what is expected from the manipulation and what actually happens provokes tension that provides a force to keep the process going until some sense of pattern or achievement, wonder, pleasure, or further surprise or curiosity that drives the process on. Although the sense of what is happening remains vague, further manipulation is spurred on, developing a flow towards the purpose of some form of controlled mathematical activity. (1984, p. 52)

This approach recognizes that mathematical dynamics, similar to the functional systems proposed by Shvarts et al. (2021), involve the interplay of various components—concepts, techniques, perspectives, and processes—facilitating a deeper understanding of mathematical phenomena through cognitive engagement and exploration. Thus, the key elements of mathematical dynamics and computational thinking are identified as tools arising from the cognitive activities of the learner as they engage with the programming environment.

In summary, this study adopts an enactivist framework to characterize tools as occasioning from the interaction between the cognizing agent and the programming environment. This perspective integrates insights from the works of Shvarts et al. (2021) on functional systems, Hegedus and Moreno-Armella (2010) on hot-spots, Kieren et al. (1997) on occasioning, Brennan and Resnick (2012) on computational thinking, and Burton (1984) and Mason et al. (1982, 2010) on mathematical thinking, all of which provide a nuanced understanding of the interplay between tools, agents, and the environment. Table 2 summarizes the initial conceptualization of computational and mathematical occasion tools in this study.

Table 2*Summary of Computational and Mathematical Occasioning Tools*

Occasioning tool	Description	Coding application example
Computational		
Computational concepts	Understanding both the technical and conceptual elements associated with the core components of coding, including their functions and practical uses.	Sequencing, event and triggers, parallelism (executing processes simultaneously), using conditionals, operators, and variables.
Computational practices	The techniques and strategies utilized in the process of coding.	Incremental and iterative coding (building, testing, and modifying code step by step), debugging, remixing or reusing code (own or others'), modularization (organizing code into modular blocks that contribute to a larger procedure).
Computational perspectives	The attitudes and dispositions exhibited during the coding process.	Sharing code with others, collaborating to solve problems, viewing coding as a personal creative expression, recognizing technology as a problem-solving tool.
Mathematical		
Mathematical Operations	This encompasses a range of mathematical actions, from commonly recognized procedures such as addition and subtraction to more subtle cognitive activities.	Enumerating, repetition, or iteration; identifying relationships between elements (e.g., ordering, correspondence, equivalence, inverse, converse); transforming elements through

Occasioning tool	Description	Coding application example
Mathematical Processes	The methods through which learners acquire and apply mathematical knowledge and skills.	combination or substitution. Specializing, conjecturing, generalizing, convincing, problem-solving, reasoning and proving, reflecting, connecting, communicating, and selecting appropriate tools and strategies.

In the next section, the research questions are further explicated.

Theoretical Basis for Identifying Components of the Emerging System of Co-Action

This study investigated the co-action between mathematics learners and their programming environment, focusing on how learners perceive, interact with, and respond to perturbations within this context. Specifically, it examines how learners engage with computational and mathematical tools (occasioning tools) and how their interactions shape both their cognitive development and the learning environment.

To thoroughly examine the nature of co-action between mathematics learners and their programming environment, it was essential to identify and describe focal elements during interactions: perturbations, occasioning tools, and physical structures resulting from structural coupling with tools. This required identifying the present states of these elements, despite their dynamic nature, over time. Having already established a basis for beginning to understand occasioning tools, it was crucial to select an analytical method aligned with the enactivist theoretical framework to pinpoint pertinent elements in the data.

To achieve this, cognitive stigmergy was combined with an analytical tool—Towers and

Martin's (2015) colour-coded analysis—to examine the data. The colour-coded approach helped identify distinct sections within the data indicating focal element, including environmental prompts or perturbations, participant thinking, participant actions with tools in response to perturbations, and emerging physical structures. Cognitive stigmergy was further used to analyze the nature of these interactions. The following section explores cognitive stigmergy as a theoretical perspective.

Stigmergy

Stigmergy was included to analyze the nature of each focal element of co-action. The inclusion of stigmergy was inspired by Mgombelo's (2017) paper, "Collective Learning: Rethinking the Environment, Artifact and Classroom Interactions." In this article, Mgombelo discusses the significant role artifacts, such as whiteboards, play in coordinating actions and behaviours in mathematics classrooms. The paper builds on decades-old studies that investigate mathematics classrooms as complex systems where agents instinctively and collaboratively interact and adapt to each other, organizing and sustaining learning (e.g., Blikstein et al., 2006; Davis & Simmt, 2003; Staples, 2008). From this theoretical lineage, Mgombelo posits that cognitive stigmergy can help us better understand how agents in a complex system interact by indirectly communicating through their environment, which is articulated by artifacts.

The word stigmergy originates from the Greek words *stigma* (sign) and *ergon* (action) and was coined by French zoologist Grasse (Parunak, 2006). Stigmergy means that due to agents' actions within their environment, they leave signs or traces that determine the subsequent actions of other agents or the agents themselves who sense the signs (Heylighen, 2015, 2016; Parunak, 2006). Grasse's term "stigmergy" was coined after he observed how social insects communicate indirectly using artifacts such as chemical traces and materials within their environment (Heylighen, 2015, 2016; Parunak, 2006). Parunak (2006) notes that, despite

Grasse's initial focus on insects, stigmergic systems are widespread in human interactions, making it more difficult to identify situations where stigmergy does not apply. Heylighen (2015) extends this idea, arguing that stigmergy is not only present in human interactions but also in complex systems more broadly, particularly those involving cognition, cooperation, or organization as products of evolution. Heylighen explains that when properly defined, stigmergy emerges as a nearly universal mechanism capable of offering valuable insights into a range of conceptual challenges in non-trivial ways. Heylighen further asserts that in a stigmergic system, actions can influence future actions, even without a specific agent directing them, as in the case of chemical reactions. Therefore, Heylighen (2015) refines previous definitions in literature, stating that "stigmergy is an indirect, mediated mechanism of coordination between actions, in which the trace of an action left on a medium stimulates the performance of a subsequent action" (p. 5). For the purposes of this study, I will focus on systems that involve agents.

Since the 1990s, stigmergy has been diffused to other fields and studies to gain insight into complex systems (Heylighen, 2015, 2016), encompassing a wide range of multi-agent coordination mechanisms that depend on information interchange through a common space (Parunak, 2006). Stigmergy, a spontaneous form of self-organization, has significantly influenced fields such as communication networks, robotics, artificial intelligence, manufacturing, marketing, and collaborative computer-aided work among human agents. Its impact extends to theories of cognition and epistemology, with ongoing advancements continuing to expand its relevance (Heylighen, 2015, 2016). According to Heylighen (2015), stigmergy remains in the early stages of its development, with vast potential for further exploration. Its application to human interactions is still largely unexplored, yet it holds the possibility of transforming various scientific, technological, and social disciplines that study society, cognition, and behaviour.

Mgombelo (2017) urges mathematics educators and researchers to acknowledge the stigmergic nature of human agents in order to better understand how their collective coordination of actions—emergent behaviour—develops through interactions with artifacts in their environment. A defining characteristic of stigmergy is its ability to position agents within their surroundings, making it a flexible concept across disciplines. Drawing on existing literature, Mgombelo (2017) emphasizes that the environment is not merely a passive container but an active system that provides dynamic mechanisms and processes, fostering the emergence of both local and global coordinated behaviours. Emphasizing Maturana's (2002) work, Mgombelo's (2017) literature review shows that the environment in which humans are embedded is an integral expressive medium for them to communicate, using objects that emerge in interaction. From Maturana's (2002) work, Mgombelo (2017) analyzes that, for human systems, the environment offers signs or signals and artifacts that are valued and used in action within a shared conventional system of signs coordinated through languaging. Based on common themes in literature that support the idea that systems are embedded in their environment such that they shape and are shaped by their environment, Mgombelo submits that a mathematics classroom (with agents) is a stigmergic system and proposes Parunak's (2006) work as a framework of analysis for interacting agents in a classroom. For this study, I considered stigmergic systems to comprise a collection of agents and the environment in which they are embedded.

Mgombelo's (2017) perspective, combined with Parunak's (2006) stigmergic framework, aligns with enactivism, which posits that cognition is embodied action, involving the entire body with its sensorimotor capacities and embedded within its environment. Similarly, Parunak's framework of stigmergy, which describes an agent or a cluster of interacting agents embedded in an environment, allows Mgombelo (2017) to situate the complex system of learner interactions

within the broader classroom context, offering insights into how collective learning behaviours in a mathematics classroom are coordinated through artifacts.

In articulating the enactive approach, Varela et al. (2016), expand on emergence theory, embedding living systems within their environment to emphasize the environment's role as an active, natural, and necessary part of system interactions. From an emergence perspective, a complex system self-organizes so that internal dynamics within the system's structure give rise to emergent global properties, which appear to an observer as a purposeful, coordinated, and integrated whole without central supervision (Maturana, 2002; Mgombelo, 2017; Varela et al., 2016). Varela et al. (2016) suggest that a system's autopoietic capacity allows it to self-organize and achieve global coherence. However, unlike traditional emergence theory, enactivism posits that the agent and its external environment are interdependent. A living system and its environment are closely intertwined, as the system both shapes the environment and is continually shaped and transformed through its interactions with it (Varela et al., 2016).

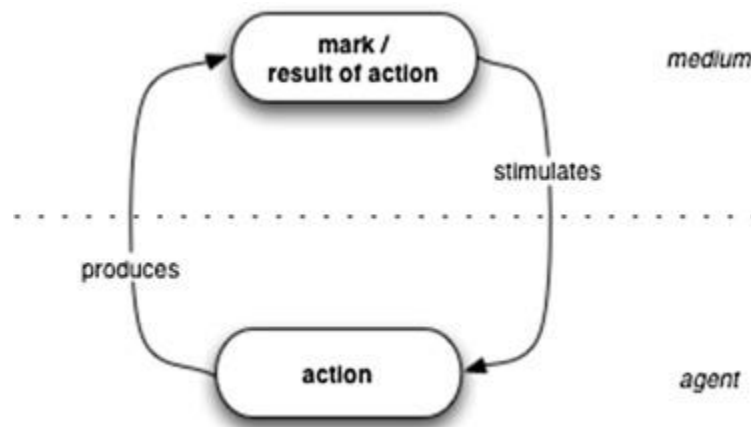
Furthermore, regarding the collective behaviour of agents in a system, Davis and Francis (2023) elaborate that enactivism positions the cognizing agent (knower) as an essential, active part of larger dynamic systems, wherein the agent continuously interacts and couples with other agents. As the agent influences the system, the system, in turn, simultaneously influences the agent. This mutual influence extends beyond the non-human environment, as the knower is also affected by the more-than-human environment (Davis & Francis, 2023), which includes broader influences such as technology, nature, and even social and cultural constructs.

Both stigmergy and enactivism recognize that the environment, together with other agents, actively guides a knower's actions in a dynamic system. Enactivism generally posits that action is perceptually guided through a continuous feedback loop of action and perception,

facilitated by the environment (Varela et al., 2016). Similarly, Parunak (2006) emphasizes that stigmergy involves the coordination of bounded agents embedded in a potentially unbounded environment, where they sense the state of the environment to guide their actions and modify it as a result of those actions. Heylighen (2016) supports this by illustrating the stigmergic feedback loop of interaction, as shown in Figure 1.

Figure 1

Heylighen's (2016) Stigmergic Feedback Loop



Heylighen's (2016) stigmergic diagram illustrates how actions leave marks or traces in the environment that trigger perception and subsequent actions. This concept parallels Varela et al.'s (2016) idea that action is perceptually guided through interaction with the environment. The notable similarities between enactivism and stigmergy offer a valuable perspective for identifying key elements of co-action using stigmergy. While stigmergy is increasingly applied across various domains, it still holds significant potential for further theoretical exploration and practical application (Heylighen, 2016). In this research, the concept of stigmergy is utilized to explore agents in programming environments.

Basic Architecture of Stigmergy. Parunak (2006) emphasizes that for distributed agents with diverse cognitive structures and limited computational resources, stigmergic interaction

serves as the primary mechanism for coordinating actions. He explains that human interaction, mediated by the environment, is inherently stigmergic and provides a useful model for designing computer-enabled systems that facilitate interactions through environmental cues. The effectiveness of applying stigmergic principles to complex systems depends on two key characteristics outlined by Parunak. First, regardless of the scale of the environment, agents interact only at a local level, preventing their limited processing resources from being overwhelmed. Second, through self-organizing dynamics, these local interactions collectively produce a coherent system-level outcome, ensuring effective control without centralized coordination.

As humans increasingly rely on computers to amplify stigmergic properties, technological systems shaped by stigmergy have become more prevalent and identifiable. In today's digital landscape, even a single agent engaging in a local interaction is more likely than ever to be connected to a broader human stigmergic system. Recognizing the widespread role of stigmergy in human interactions, Parunak (2006) conceptualizes human co-action with the environment under the framework of human-human stigmergy and proposes a fundamental architecture for stigmergic systems. However, Heylighen (2016) cautions that stigmergy is not only relevant for understanding interactions among multiple agents but is also crucial for analyzing how a single agent co-acts with their environment. He argues that stigmergy's explanatory power lies in its ability to account for environmental factors that shape and coordinate individual and collective activity.

Heylighen (2016) also addresses our tendency to misunderstand the causes of coordinated behaviour, explaining:

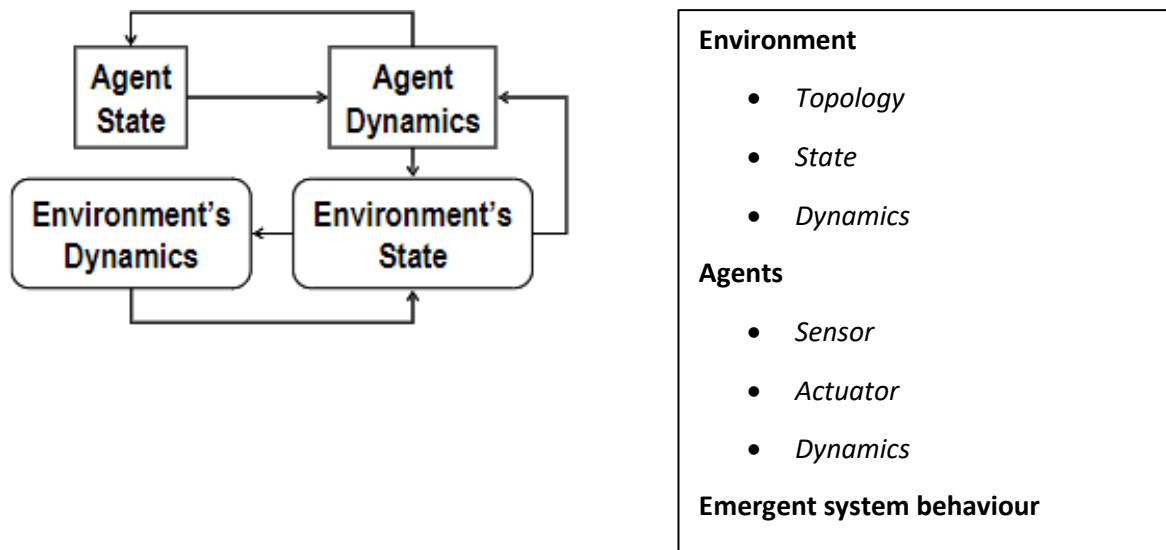
If we observe agents acting in a coordinated way, we tend to look for a direct link

between one agent's actions and another's, assuming immediate communication or a shared instinct, plan, or leader guiding their behaviour. Rarely do we consider that one agent's behaviour might influence another's only through an indirect trace left in a passive environment. (Heylighen, 2016, p. 6).

Stigmergy offers a framework for analyzing coordinated action without resorting to these intuitive explanations. Given this study's focus on exploring co-action between learners and their environment, the researcher found Parunak's (2006) basic architecture of stigmergy (see Figure 2) to be a practical tool for analyzing the focal elements of co-action. The components of Parunak's architecture are explained below.

Figure 2

Parunak's (2006) Basic Architecture of Stigmergy With Template of Analysis for Stigmergic System



Parunak's (2006) basic architecture outlines three main components: agents, environment, and emergent system behaviour. Each agent has three attributes (sensors, actuators, and dynamics) that specify its internal state. Sensors allow an agent access to some state

variables of the environment; actuators allow modification of some of the environment's state variables; and a program (agent's dynamics) that maps current internal state and sensory readings to state changes and commands given to sensors and actuators. An agent's state usually is not directly visible to other agents.

Comparable to an agent, the environment has three attributes (state, dynamics and topology). Among them, similarities may be found between the environment's attributes and an agent's attributes; however, there are two major differences. The first major distinction concerns the attribute state. Certain aspects of the environment's state are generally visible and accessible to agents with appropriate sensors. Another distinction concerns the structural difference between agent and environment. Topology refers to the way elements in an environment are arranged and how they relate to one another. Unlike agents, which are distinct and self-contained entities, the environment is usually structured according to a specific topology that defines its organization and connectivity (Parunak, 2006).

For example, a Cartesian space, such as the Earth's surface, is an environment where locations are defined by coordinates, allowing agents (such as people or robots) to navigate based on spatial relationships. A telecommunications network represents another type of topology, where different nodes (such as computers or phones) are connected in a structured manner to facilitate communication. Similarly, social organizations have a hierarchical or networked structure, where relationships between individuals or groups define the flow of information and decision-making. Another example of topology is a list of unrelated categories, where elements are arranged based on an imposed structure, such as a classification system in a library or an academic subject chart. This kind of topology helps organize information in a way that makes it easier for agents to locate and interact with relevant elements within the system.

Ultimately, topology provides a framework for understanding how agents interact with their environment by defining the spatial, relational, or categorical structure in which they operate. Agents' interactions with their environment dynamics govern how the system changes overtime. That is, the environment dynamics govern how the system's state evolves over time in response to agent interactions (Parunak, 2006).

Emergent system behaviour refers to complex, coordinated patterns that arise from the indirect interactions of individual agents through the environment. The decentralized process, in which agents modify their surroundings and leave traces, influences future actions, fostering adaptive and robust behaviours. This can be seen in examples like ant pheromone trails, termite nest building, and swarm robotics, where collective intelligence emerges without central control.

Types of Stigmergy. To thoroughly examine the nature of perturbations and occasioning tools encountered or chosen by a cognizing agent, it was essential to investigate the different varieties of stigmergy. Parunak (2006) distinguishes four types of stigmergy based on two orthogonal categories. The first category differentiates between marker-based stigmergy, where agents respond to markers (signs deposited by agents in the environment), and sematectonic (Sema) stigmergy, where agents respond to sematectonic signs (the current state of the environment). The second categorization addresses whether these signals are quantitative or qualitative: quantitative stigmergy occurs when the strength or amount of a stimulus influences how often or intensely an action is performed. Qualitative stigmergy occurs when different types of stimuli lead to different types of actions (Heylighen, 2015).

For example, social insects may base their actions quantitatively on a single scalar pheromone field or qualitatively on a combination of discrete pheromones. Similarly, ants may quantitatively cluster corpses in an ant cemetery based solely on density, while wasps may

qualitatively select a nest template that best matches the local shape from multiple options. The first pair of examples, which involve quantitative and qualitative markers, represents instances of marker-based stigmergy, whereas the second pair, which involves quantitative and qualitative sematectonic responses, illustrates sematectonic stigmergy (Parunak, 2006).

Heylighen (2015) expands these categorizations by introducing two additional distinctions: transient versus persistent traces and broadcast versus narrowcast communication. Transient traces are short-lived and disappear quickly unless reinforced, whereas persistent traces remain over time without requiring immediate reinforcement. In human behaviour, a transient trace might be a conversation or verbal instruction, which fades unless written down or repeated. For instance, if a teacher gives oral directions to students but does not provide written instructions, those directions may be forgotten if not reinforced. In contrast, a persistent trace could be a written sign, such as a posted classroom rule or a graffiti tag on a wall, which continues to influence behaviour long after its initial creation.

Broadcast communication refers to messages that can be received by everyone in the system or environment, similar to a loudspeaker announcement, whereas narrowcast communication is directed toward specific individuals or small groups, much like a private conversation or a text message.

While Heylighen's (2015) categories offer additional insight into co-actional systems, Parunak's (2006) binary distinctions serve as the primary framework used to identify and describe perturbations and occasioning tools.

Agent(s) Interacting in a Programming Environment. Now that the framework of stigmergy has been examined, explaining key components of the system, this framework can be applied to agents immersed in a programming environment to conceptualize the key components

of the co-actional system: environment, agents, and emergent system behaviour. Figure 3 outlines a stigmergic template for agent interactions within a programming environment, with further explanation provided below.

Figure 3

Stigmergic Template for Agent Interacting in a Programming Environment

Environment: Mathematics Programming Environment

- **Topology:** Linear or, if structured, hierarchical
- **State:** Degree of completeness of the program or programmed mathematical model
- **Dynamics:** Evolving relationship between inputs and outputs (semantics) as the program develops

Agents: Learners, Teaching Assistants, and Instructors (i.e., any actor operating within the programming environment)

- **Sensors**
 - **Sematectonic (Sema):** Perception of the current state of the program (e.g., incomplete code, syntax errors, unexpected output)
 - **Marker-based:** Engagement with explicit task instructions, problem statements, or coding scaffolds
- **Actuators**
 - **Sematectonic (Sema):** Use of occasioning tools (e.g., debugging tools, visualization aids, algorithmic strategies) that support mathematical or computational thinking
 - **Marker-based:** Placement or response to explicit indicators in the environment (e.g., code comments, instructional hints, syntax highlighting) to guide action
- **Dynamics:** Agent-driven adjustments to the program or mathematical model aimed at increasing efficiency, correctness, or alignment with task expectations

Emergent system behaviour: A completed program or programmed mathematical model that results from iterative agent-environment interactions and emergent learning processes

Firstly, as established, the environment itself is characterized by its topology, state, and dynamics. The topology refers to the structural layout of the environment, which can be linear or hierarchical. In this study, MICA students worked with syntax-based programming languages and integrated development environments (IDEs), specifically using Visual Basic.NET (VB.NET) in Visual Studio and Python in the Spyder IDE. In VB.NET programming, students design graphical user interfaces (GUIs) through mouse interactions and use VB.NET syntax to implement functionality. Similarly, Python programming involves creating GUIs and writing

structured syntax-based code. Both languages follow a linear, sequential execution and procedural structure, moving through step-by-step processes. They also incorporate a hierarchical structure to organize code through modules, packages, classes, and namespaces. The state of the environment is defined by the completeness of the programs or mathematical models under development. The dynamics refer to the changing relationship between input and output, which evolves as a result of the agent's interactions within the environment.

Secondly, as outlined in the previous section, the agent's description includes state, sensors, and actuators. The agent's state represents its dynamic structure, shaped by its autopoietic capacity and a history of structural coupling (Maturana, 2002). Maturana (2002) argues that a system's structure changes through structural coupling, which involves ongoing interactions with the environment. Parunak (2006) notes that an agent's state is often not directly visible. Sensors allow agents to perceive the condition that governs the current state of the programming environment (the completeness of the programs or mathematical models or other environmental prompts) that inform or influence their actions. Heylighen (2015) describes action as a causal process, meaning that every action has a cause, or triggering condition, and an effect, or change in the environment. Actions modify the state of the environment, creating new conditions that may trigger further actions. However, a condition does not always guarantee an action; it only increases the likelihood of one occurring. For example, a green light increases the likelihood that someone will cross the street, but it does not guarantee it (Heylighen, 2015). Actuators are programming or mathematical tools (as perceived by the agent) that enable agents to interact with and modify the environment state.

Both sensors and actuators can be sematectonic (sema) or marker based. In this study, sema sensors correspond to the current state of the program as perceived by the agent. These include recognizing syntax errors, incomplete code, unexpected outputs, or structural

inconsistencies in the program. The agent detects these emergent conditions through direct interaction with the programming environment, which influences its subsequent actions. Marker-based sensors correspond to the agent's affiliation with the mathematics programming task or instructions. These are explicit guides, such as problem statements, step-by-step instructions, or predefined coding structures that direct the agent's approach to constructing the program. Unlike sema sensors, which arise from the evolving state of the program, marker-based sensors provide structured external input that influences the agent's decisions.

Sema actuators correspond to the agent's use of tools or actions that indirectly modify the programming environment in ways that support mathematical or computational thinking. In addition to Brennan and Resnick's (2012) three dimensions of computational thinking, these include debugging tools, function libraries, algorithmic strategies, and visualization aids that assist the agent in refining and optimizing the program. Through engagement with these tools, the agent alters the program's structure or logic, not necessarily to communicate, but to improve performance, efficiency, or clarity.

Marker-based actuators correspond to explicit indicators deliberately placed in the mathematics programming environment to guide the agent's or others' actions. These may include comments written by the agent, syntax highlighting, compiler warnings, or error messages. Such markers shape the agent's subsequent modifications by prompting specific adjustments aligned with programming conventions, mathematical accuracy, or task expectations.

Through these interactions, the agent dynamically adjusts the program or programmed mathematics to develop an efficient and functional solution, continuously responding to both sema- and marker-based influences in the programming environment.

Thirdly, emergent system behaviour in a programming environment refers to the development of a completed program or mathematical model that evolves through iterative agent-environment interactions and emergent learning processes. As agents engage with the environment (e.g., modifying code, interpreting feedback, and responding to both marker-based and sematectonic cues) the program is continuously refined. Rather than being fully predetermined, the final product emerges from cycles of debugging, structuring, and optimizing, shaped by both individual decisions and the cumulative effects of interaction with the environment.

It is essential to clarify that sensors and actuators should not be mistaken for the physical environment as viewed externally. From an enactivist perspective, the autopoietic agent perceives its surroundings at a fundamental level, categorizing objects based on the interactions they afford. As Varela et al. (2016) argue, the cognizing agent, when perturbed, delineates a world of significance through interactions with tools. An agent's dynamics play a crucial role, as they govern commands given to sensors and actuators and ultimately result in state changes. For this study, dynamics refer to the agent's adaptive behaviour, modifying programs or mathematical models to improve efficiency. Implicit in an agent's dynamics is intentionality, which arises as the cognizing agent is perturbed by the environment. Upon encountering a perturbation, the agent must value the experience and respond, which gives rise to intention. Thus, intentionality is dynamic, continually emerging through enaction, and is essential for action.

The cognizing agent's action is inherently linked to the environment by perception, but its world of significance is determined only through active engagement with tools in response to environmental perturbations (Varela, 1992). Parunak's (2006) distinction between marker-based

and sematectonic interactions provides a means to investigate the nature of perturbations and the tools involved in agent action. Table 3 and Figure 4 below summarize this study's analytical framework, drawing on stigmergy literature (Heylighen, 2016; Parunak, 2006) and enactivism. Figure 4 illustrates the dynamic architecture of co-action between agents (learners) and the environment in a programming context. It shows how an agent's state, shaped by prior experiences (structural coupling), interacts with environmental feedback. These interactions trigger actions that alter the state and dynamics of the environment, producing new conditions for perception and further engagement.

Figure 4

Basic Architecture of Stigmergy for Agent Interacting in a Programming Environment

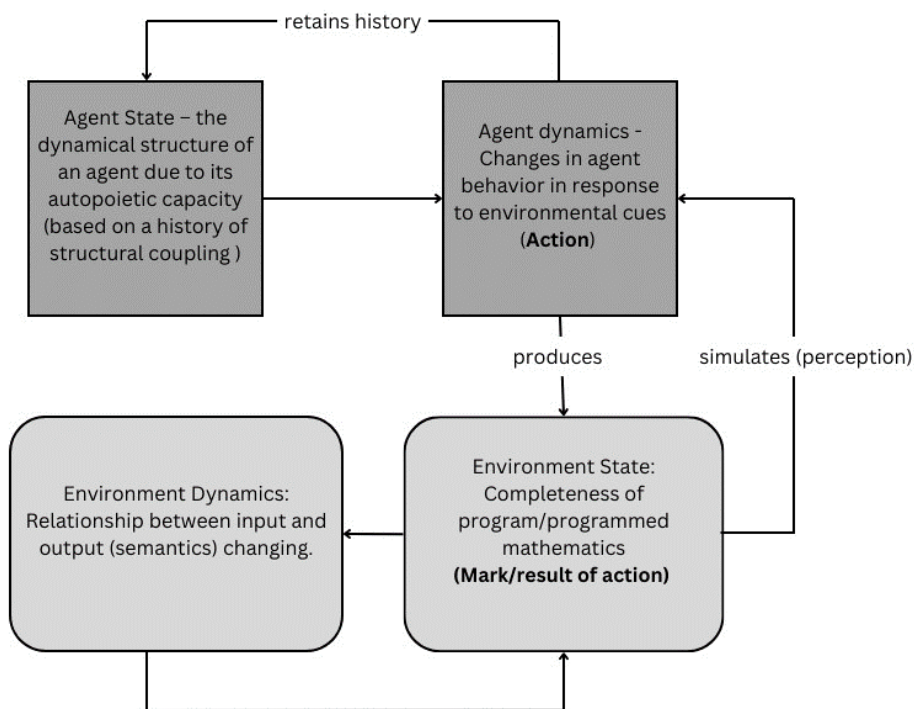


Table 3 (below) outlines an analysis template for examining how learners interact with the programming environment through co-action. It breaks down the system into key parts, including agent state, environmental prompts, tool use, and emergent outcomes, and provides

guiding questions to help interpret how learners respond to perturbations and progressively construct knowledge through programming for mathematics. Each component in Table 3 maps onto a corresponding element in Figure 4. For example, the Programming Environment in Table 3 corresponds to Environment State and Environment Dynamics in Figure 4, while environmental dynamics and emergent learning in Table 3 align with the evolving system feedback loop shown visually in Figure 4. Together, they support a coherent enactivist interpretation of how learning unfolds in a mathematics programming environment

Table 3

Analysis Template for Co-Action for Agents Interacting With Environment

System part	Description guide	Corresponding element in Figure 4
Programming Environment	<ul style="list-style-type: none"> - <i>Topology</i>: Is the environment organized in a linear or hierarchical structure? - <i>State</i>: What is the completeness of the program or programmed mathematics? - <i>Dynamics</i>: How does the relationship between input and output evolve? 	- Environment State and Environment Dynamics
Agent's State (History of Structural Coupling)	<ul style="list-style-type: none"> - <i>Cognitive Structure</i>: What is the agent's cognitive background in relation to interaction with the programming environment? - <i>Prior Experience</i>: What is the learner's prior experience with similar tasks? - <i>Affective State</i>: How does the learner feel about mathematics/programming? 	- Agent State (based on structural coupling)

System part	Description guide	Corresponding element in Figure 4
<p>Prompting/Perturbation from the Environment</p>	<ul style="list-style-type: none"> - <i>Sensors</i>: What elements within the system detect and perceive changes in the environment? <ul style="list-style-type: none"> ▪ Sema: What is the current state of the program? ▪ Sema: What is the nature of the agent affiliation with the task/environment? ▪ Marker: What objects are added to guide learners (e.g., comments)? 	<ul style="list-style-type: none"> - Perception (simulated through sensors and markers)
<p>Agent's Thinking in Relation to Perturbations (Choice of Occasioning Tools)</p>	<ul style="list-style-type: none"> - <i>Actuators</i>: What elements within the system carry out actions or modifications based on environmental cues? <ul style="list-style-type: none"> ▪ Sema: What tools are chosen to initiate action? ▪ Marker: What comments or prompts guide the agent's actions? 	<ul style="list-style-type: none"> - Actuators (tools and strategies used in action)
<p>Actions Taken by Agents</p>	<ul style="list-style-type: none"> - <i>Dynamics</i>: How does the agent adjust behaviour in response to environmental cues? <ul style="list-style-type: none"> ▪ How does the agent modify the program or mathematics to improve efficiency? ▪ What types of inquiries are initiated based on these cues? 	<ul style="list-style-type: none"> - Agent Dynamics (Action informed by perception and history)
<p>Completed Program/Programmed Mathematics with Emergent Learning</p>	<ul style="list-style-type: none"> ▪ What patterns or outcomes emerge from the agent-environment interactions? ▪ What final structure or solution is developed as a result of these interactions? 	<ul style="list-style-type: none"> - Environment State and Dynamics (marks of action and change)

Chapter Summary

This chapter explored the application of enactivism within programming environments, highlighting how cognition emerges from active interaction between agents and their environments. It positioned enactivism within a broader conceptual framework, drawing on Hegedus and Moreno-Armella's (2010) "hot-spots" concept, which identified key moments where tools facilitated meaningful interactions. The chapter also incorporated Kieren et al.'s (1997) idea of "occasioning," which illustrated how agents' selective engagement with elements of their environment reshaped their understanding. Additionally, Brennan and Resnick's (2012) dimensions of computational thinking were integrated with Burton's (1984) and Mason et al.'s (1982, 2010) perspectives on mathematical thinking, emphasizing problem-solving as iterative and reflective. Finally, the chapter used the concept of stigmergy to conceptualize how agents self-organized and co-emerged with their programming environment

CHAPTER FOUR: METHODOLOGY

The proposed methodology for this research was guided by the study's purpose, the nature of the research questions, and the theoretical framework of enactivism. The purpose of the study was to explore the interactions between learners and their environment (co-actions), as they used programming for mathematics learning. It examined what guided their actions as they shaped and were simultaneously shaped by the environment while engaging with mathematics through programming.

Given the exploratory nature of the research questions, a qualitative approach was employed. A qualitative approach allows researchers to explore and describe the sociocultural dynamics of complex and highly contextualized learning environments (Miles & Huberman, 1994; Strauss & Corbin, 1998). It is particularly suited to understanding the situated nature of students' experiences within dynamic learning contexts, where the learner or cognizing agent is deeply intertwined with the environment and the phenomena being studied.

Furthermore, due to the phenomenological roots of enactivism, this research was informed by phenomenology as a research method. The following section provides a brief overview of phenomenology.

Brief Overview of Phenomenology

Phenomenology originated as a philosophical school of thought and later evolved into a research method (Groenewald, 2004). As a philosophical movement, it can be traced back to Immanuel Kant and Georg Hegel. However, the German mathematician and philosopher Edmund Husserl (1859 to 1938) is widely credited as the pioneer of phenomenology. Later, philosophers such as Jean-Paul Sartre and Maurice Merleau-Ponty expanded phenomenology's influence in diverse directions (Groenewald, 2004).

Developed in the aftermath of World Wars I and II, phenomenology emerged in response to the physical and ideological disruptions that destabilized social order across Europe. It rejects Cartesian dualism, which assumes that objects in the external world exist independently of the mind and that knowledge about them is purely objective. Instead, phenomenology explores the subjective nature of consciousness. Its goal is to “describe the universal structures of subjective orientation in the life-world” (Eberle, 2014, p. 5).

As a research method, phenomenology is used to explore and understand the essence of lived experiences from the perspectives of individuals engaged in a particular phenomenon. Rather than seeking to generalize, this approach aims to gain deep insights into how participants experience, interpret, and make meaning of specific situations or events. This focus aligns with the purpose of this research. Therefore, guided by phenomenology, the study explores the lived experiences of students in mathematics programming courses. The emphasis is on their firsthand experiences, particularly how they interact with programming environment, navigating their learning in real time. As a result, a phenomenology-informed approach is well-suited to this inquiry because it captures the subjective meanings of participants' experiences and uncovers patterns that might remain hidden through other research methods (Groenewald, 2004).

Furthermore, phenomenology also aligns with the theoretical framework of enactivism, which emphasizes the dynamic interdependence between learners and their environment. Both approaches reject rigid, objectivist views of knowledge and focus on understanding how individuals actively shape their surroundings and are simultaneously shaped by them. Through phenomenological-informed inquiry, this study aims to uncover the deeper, situated meanings of preservice teachers' interactions with both the learning environment and other participants as they engage in mathematics through programming. However, unlike Husserl's philosophy,

which assumes that personal experience can be fully bracketed, in this study I adopted a more reflective stance toward bracketing—one that acknowledges my positionality and treats bracketing as an act of reflexive awareness rather than suspension of prior experience.

Researcher Positionality

As a woman, a researcher, a mathematics educator, and a Caribbean immigrant who has taught in Jamaica as well as in the culturally diverse contexts of the Cayman Islands and Canada, my lived experiences and professional roles shape the way I engage with this research. From an enactivist perspective, I recognize that I cannot stand outside the research as a neutral observer; my interpretations are entangled with my cultural history and professional practice. My two decades as an educator influence how I understand mathematics, pedagogy, the theoretical frameworks I draw upon, and the ways I relate to participants' data.

Rather than a limitation, I see my positionality as an advantage, it allows me to bring my empathy and contextual sensitivity to participants' experiences, to sense cultural, affective and pedagogical elements in their learning, and to situate findings in broader conversations about mathematics education in diverse environments. To remain reflexive, I practiced bracketing—not in Husserl's sense of fully suspending all preconceptions, which is impossible, but as a reflective process of making my assumptions explicit and holding them in the background. This allowed participants' enacted meanings to guide the analysis while keeping me aware of how my own perspective shaped interpretation.

In addition, including this positionality statement provides transparency by showing readers how my perspective shaped the study, enhances the credibility of the research by demonstrating critical awareness of my influence, and strengthens alignment by connecting my viewpoints—enactivism, stigmergy, and phenomenology—with how I designed and interpreted

the study. It also enables readers to critically assess the findings, as they can see the interpretive lens through which knowledge was co-constructed. In this way, my positionality enriched the study, contributing to the co-construction of knowledge between participants, their programming environments, and myself as researcher.

Participants for the Study

According to Hycner (1985), the phenomenon under investigation shapes the overall methodology, including the number of participants and the criteria for their selection. Hycner explains that maintaining control and rigor in a study partly depends on selecting participants who are well-suited and capable of thoroughly describing the experience under investigation. Following this approach, the participants were students from a Canadian university enrolled in a course where mathematics is taught through a coding or programming platform. Purposeful sampling was used to select participants who could offer meaningful insights into their co-action with the learning environment.

All students enrolled in the MICA I, II, and III courses were invited to complete an online questionnaire and volunteer for participation in the study. This approach targeted participants at varying stages of their learning trajectory, allowing for a comprehensive understanding of co-actional experiences across different levels of engagement. A sample size of approximately eight participants was targeted, as it is generally accepted that a maximum of 10 participants is sufficient to gain an in-depth understanding of the selected phenomenon (Creswell & Poth, 2016; Dworkin, 2012). However, ultimately, six participants (two males and four females) completed the online questionnaire. All six participants were invited to participate in the study and signed the informed consent form. This sample size was considered adequate for several reasons:

1. The primary data collection method involved interviewing, and as Hycner (1985) notes, even one participant can yield substantial data.
2. The data set was homogeneous (Dworkin, 2012); all participants were students of a similar age group at the same university, enrolled in the MICA courses, with an expected total population of around 60 students.

Additionally, the six participants were selected based on criteria reflecting the intended diversity of the research group, including gender, program of study, and year of study. Specifically, the participants included a 2nd-year student double majoring in mathematics and physics in his first MICA course, a 3rd-year mathematics co-op student in her second MICA course, a 4th-year combined major in physics and mathematics in his second MICA course, a 4th-year concurrent intermediate/senior education student with teachables¹ in mathematics and biology in her third MICA course, a 3rd-year concurrent education mathematics major in her third MICA course, and a 4th-year concurrent education student with teachables in mathematics and French in her third MICA course. The purposeful sampling method used in this study, tailored to the specific needs of the research, is a common approach in qualitative studies for obtaining rich information from a small number of participants (Palinkas et al., 2015).

Data Collection Methods

Conducting a phenomenological study involved capturing the richness of the phenomenon at hand and the setting (Kensit, 2000). In this case, the setting was the emerging mathematics and computational thinking arising from engaging with mathematics in a programming context. Kensit (2000) further suggests that researchers should allow data to

¹ In the context of Canadian universities, a “teachable” refers to a subject area that a teacher candidate is qualified to teach, typically in secondary school. When enrolling in a Bachelor of Education (B.Ed.) program—especially at the intermediate/senior level (Grades 7–12)—students are usually required to choose one or two “teachable subjects.”

emerge naturally. In alignment with this perspective, three data collection methods were used to maintain data quality: semi-structured, stimulated recall interviews with EOs, field notes, and student reflections.

Audio-recorded, semi-structured, stimulated recall interviews served as a primary means of data collection. Semi-structured interviews provided the advantage of using pre-determined guided questions to focus the interview process while also allowing the interviewer to explore participants' responses, seek clarifications, and probe for additional information (Kallio et al., 2016). In addition, as part of the course requirements for MICA, students were asked to submit reflections for each EO they completed, which were analyzed alongside their interview data. Participants' EOs and associated reflections were also used as stimuli in these interviews. Stimulated recall interviews originated from introspective research procedures and are commonly used in educational research to access participants' reflections on cognitive processes (Lyle, 2003). In many cases, participants recall their thinking about a learning experience by viewing a pre-recorded video of the event. However, other approaches have also been used to stimulate memory. For example, Kus and Cakiroglu (2022) employed non-video stimulated recall interviews to explore 7th-grade learners' visuospatial interactions in an art studio environment, using their artworks in response to geometrically rich tasks as prompts to recall their creative processes. Learners reflected on their creative experiences while constructing their drawings and explained what prompted key actions. Similarly, Moreland and Cowie (2016) and Fox-Turnbull (2009) used non-video stimulated interviews with autophotographs (photographs taken by students to document their creative processes) as prompts in semi-structured interviews. Fox-Turnbull (2009) employed autophotographs in a Ph.D. study to prompt discussions about students' learning, providing insight into their thinking and decision-making processes as they

developed technological objects to meet specific needs. Following these examples, participants' EOs and associated reflections were collected and reviewed beforehand so that the interviewer could use the associated assignment pieces alongside the interviews to prompt participants' recollection of the computational and mathematical practices they applied in developing their EOs. All interviews in this study were audio-recorded for transcription.

Finally, field notes were used to record information during the study, including the researcher's observations, reflections, and things seen or heard during the interview process. Groenewald (2004) highlights the importance of field notes in keeping the researcher focused, as it is easy to become absorbed in data collection and neglect reflection. The use of these three data collection methods provided triangulation, enhancing the validity of the findings (Creswell & Poth, 2016). All data were stored with unique codes to ensure confidentiality, and the research data were secured on a password-protected computer and files.

Data Analysis

Data from the interview were transcribed and then analyzed using thematic analysis (TA), following Braun and Clarke's (2006; 2020) six-phase framework. The process involved a colour-coded method inspired by Towers and Martin (2015) to begin to capture the co-action and context. However, both methods were preceded by transcription of the data.

Transcribing the Interview

Manual transcription of the data was conducted orthographically, capturing various features such as inflections, breaks, pauses, tones, and other relevant elements from both the interviewer and the participants (Braun & Clarke, 2013). This process involved repeatedly playing back the recordings while following along with the transcript, which greatly facilitated a deep immersion into the data. In the transcription process, Hycner's (1985) recommendation was

followed by leaving a large margin on one side of each transcription to record units of general meaning. For this transcription, two columns were created: one for documenting observations and insights during the early stages of analysis, and another for coding. As part of the member checking process (discussed further in a later section), each participant reviewed their transcript within two weeks of the interview and confirmed its accuracy.

Thematic Analysis (TA)

TA is commonly used across qualitative research to understand subjective experiences. TA is accessible and flexible, enabling researchers to locate, analyze, and report patterns of meaning (themes) within data (Braun & Clarke, 2012; Guest et al., 2011). It allows the researchers to identify commonalities among interview participants in relation to the research question, thereby making sense of shared experiences (Braun & Clarke, 2012), which in this case is doing mathematics in a programming context.

The six phases of TA outlined by Braun and Clarke (2006, 2020) and applied by Byrne (2022) were generally followed in this study and unfolded similarly to their description. The process did not proceed in a strictly linear manner; instead, it progressed recursively and iteratively. Earlier phases were revisited as needed to refine themes and deepen the understanding of the data (Byrne, 2022). This recursive approach was useful in thoroughly capturing the complexity of the participants' experiences. The following paragraphs outline the thematic analysis process applied.

Familiarization With the Data. The interview data were organized in table format following Braun and Clarke's (2012) recommendation. A three-column table was used to distinguish two levels of coding for each participant: Level 1 coding (colour-coding) and Level 2

(bottom-up, inductive and top-down deductive approaches). For example, Figure 5 below presents a segment of the table used in analyzing Luca's data.

The first column contained the transcribed interview, the second column was used for Level 1 coding, and the third column for Level 2 coding. Further, in the second column, notes were recorded while repeatedly listening to the interview recordings and reading the data to become familiar with each set. This process involved reading critically and posing questions to interpret the data in relation to the research questions. As Braun and Clarke (2012) note, "these initial observations suggest the data will provide fertile grounds for analysis" (p. 61). During this phase, Towers and Martin's (2015) colour-coded technique (Level 1 coding) was also applied to support the analysis.

Figure 5

Example of Table Used in the Process of Thematic Analysis

<p>1. What is the nature of co-action between mathematics learners and their environment as they engage in doing mathematics through programming? 1.1 In what ways do the perturbations prompt computational thinking for mathematics learning? 1.2 How do learners respond to the perturbations within a mathematics programming environment? 1.3 How do learners shape the programming environment for mathematics learning?</p>		
<p>parts of the overall conversation that reflect environmental prompting. (Speaking with the learner) participants' thinking parts of the participants' speech that reflect action.</p>		
<p><i>I'm sorry about that. So first of all, you basically you plan it out, you plan that out on paper?</i></p> <p>Yep.</p> <p><i>Yeah. And that's sound like some math was involved. So, you did the math on paper first too?</i></p> <p>Mm-hmm. Yeah, like how I kind of did it was like I'd have an example already written out, so I knew kind of the steps of how it should work like on paper, and then what, like the solution should be like what the message should encrypt to and what it should decrypt to and then I kind of tried to do it with that given example in the program.</p> <p><i>Mm-hmm.</i></p> <p>And then from there like you can kind of extend that to any other code you want to do.</p>	<p>Plan out the math on paper.</p> <p>Figure out steps to program from the example on paper (intentionality)</p> <p>Written steps placed in environment (marker)</p> <p>Intentionality</p>	<p>Planning on paper as a guide</p> <p>Develop steps as a guide</p> <p>Understanding the process/problem</p> <p>Use specific examples to outline steps</p> <p>Iterative approach</p> <p>Collaborating</p> <p>Mathematical processes</p> <p>Mathematical operation</p> <p>Mathematical dynamics</p>

The inclusion of colour-coded analysis, used by Towers and Martin (2015), was inspired by the enactive nature of co-action and Papert's (1980) view of the computer as a "mathematical speaking entity" (p. 20). Papert (1980) and Towers and Martin (2015) both observed that learners engage in constant dialogue with their environment to develop mathematical understanding. For Papert, with the computer as a "mathematical speaking entity," and for Towers and Martin (2015), with other learners when doing mathematics as a collective effort. Towers and Martin (2015) noted that they "could not tease apart the actions and dialogue to create separate, coherent, individual traces of understanding" but had to treat them as a "single coherent voice, emerging through a coactional process" (pp. 250–251). They used different colours to represent individual voices, helping to illustrate co-action among participants.

While participants shared their individual experiences on assignments, Towers and Martin's (2015) analysis method was adapted during the familiarization stage to explore the conversations that emerged through co-action. In each transcription, different colours were used to distinguish segments reflecting prompts to participants, participants' thought processes, and their speech related to actions. This technique helped identify factors influencing participants' actions and thinking, shedding light on the significance they ascribe to their experiences.

By corroborating notes from the colour-coded analysis with participants' EOs, reflections, and field notes, and guided by the conceptual framework, an annotated narrative was developed for each participant during the familiarization stage to gain deeper insight into their stigmergic experiences. Throughout this process, a conscious effort was made to bracket personal assumptions by setting aside preconceived ideas or biases in order to remain true to the participants' experiences, as recommended by Groenewald (2004). This bracketing ensured that interpretations emerged directly from the participants' accounts, rather than being influenced by

prior expectations or knowledge. This detailed exploration of each participant's account provided valuable insights into the data, which informed the Level 2 coding process of producing pithy descriptive or interpretive labels for pieces of information in the interview data relevant to the research question (Braun & Clarke, 2012). During the Level 2 coding of the semi-structured interviews, information from participants' EOs, reflections, and field notes was also corroborated with the interview to check for consistency.

Generating Initial (Level 2) Codes. Codes were created by first using a combination of bottom-up, inductive, and then by using top-down, deductive approaches (Braun & Clarke, 2012; Byrne, 2022) aligned with the research focus. A bottom-up approach was employed to identify any information relevant to the research question that emerged from the data. Conversely, a top-down approach was applied by using Brennan and Resnick's (2012) dimensions of computational practices and Burton's (1984) conception of mathematical thinking to frame occasioning tools. All codes were recorded in the third column of each participant's table (see example in Figure 4 above). Following this, Byrne's (2022) worked example was replicated. A Microsoft Excel spreadsheet was then used to document and manage the iterations of coding. An abbreviation for each participant was listed in the first column, and data items or excerpts were listed in the second column, with subsequent columns recording each iteration of codes. Changes between iterations were highlighted to track the evolution of the coding process. Initially, all first-iteration codes for each participant were transferred into a separate spreadsheet corresponding to each individual. The original transcripts were revisited throughout the process to refine existing codes and identify new ones as familiarity with the data deepened. During the iterative coding process, some initial coding was revised to enhance clarity and relevance (Braun & Clarke, 2012), and redundant codes unrelated to the research questions were eliminated

(Groenewald, 2004). In addition, the Level 2 coding process was informed by the Level 1 coding, in that each part of the colour-coded data was interrogated to understand the nature of the interaction. This interrogation was guided by the theoretical framework of stigmergy. For example, sections coded as “perturbation” were examined to determine the type of perturbation (e.g., sensor-based, including sema or marker) and its source (such as environmental output or agent traces). Segments coded as “action” were analyzed to identify the nature of the action and the tools (actuators) used to carry it out. Colour-coded segments reflecting “thinking” were used to interpret both sensing and actuation processes. However, this coding was not limited to rigid categories, as the entire conceptual framework informed the interpretive process holistically.

The coding process captured both semantic meanings, reflecting the explicit content of participants’ statements, and latent meanings, uncovering underlying assumptions and implied ideas. Throughout the process, again a conscious effort was made to bracket personal assumptions, and care was taken to ensure that codes were meaningful and connected to multiple data items, allowing for the development of patterns and themes (Groenewald, 2004). Also, as Hycner (1985) cautioned, the researcher made a conscious effort to remain open during coding so that meaning could emerge. Groenewald (2004) emphasized that openness is crucial, as the researcher must explicate meaning by judging the situation.

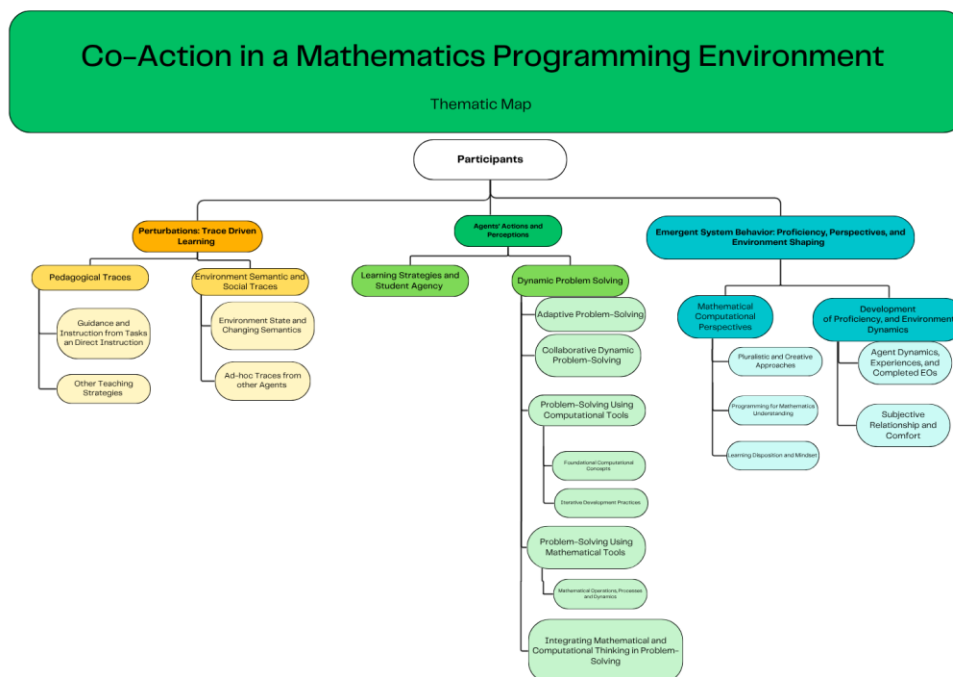
Generating Themes. With the codes generated, those with similar or unifying features were grouped into clusters to reflect meaningful patterns in the data (Braun & Clarke, 2012; Byrne, 2022). Clustering involved identifying themes or recurring ideas that spanned multiple participants or data points, facilitating a deeper understanding of the phenomenon under investigation. Coded data were organized into candidate themes and sub-themes, focusing on identifying patterns of meaning across the dataset rather than individual data points. Some codes

were combined, collapsed, or promoted into overarching themes based on shared meanings (Byrne, 2022).

Overlapping clusters revealed connections between ideas across themes, contributing to the development of a coherent narrative (Braun & Clarke, 2012). A thematic map (see Figure 6) was used to visualize these themes and sub-themes, ensuring that each theme meaningfully addressed the research questions (Byrne, 2022).

Figure 6

Analysis Template for Co-Action for Agents Interacting With Environment



Throughout the process, themes were refined, complex or unclear themes were revised, and irrelevant codes were discarded to maintain clarity and coherence, following the guidelines of Braun and Clarke (2006, 2012, 2020). The final version of thematic map in Figure 6 presents the six themes, organized under three overarching categories: Perturbations: Trace-Driven

Learning; Agents' Actions and Perceptions; and Emergent System Behaviour: Proficiency, Perspectives, and Environment Shaping.

To ensure the findings accurately reflected participants' experiences, again personal assumptions were bracketed (Groenewald, 2004). Additionally, following Hycner's (1985) guidance, the original interview recordings were revisited iteratively alongside the list of non-redundant meanings. This iterative approach prevented oversimplification and ensured no key insights were overlooked. As a result, the final themes maintained the integrity of the data and accurately represented participants' lived experiences.

Reviewing Potential Themes. In this phase, a recursive review of candidate themes was undertaken to ensure they provided meaningful interpretations aligned with the research questions (Braun & Clarke, 2006, 2012, 2020; Byrne, 2022). This process involved evaluating the coherence and relevance of themes, focusing on whether each theme captured significant patterns in the data (Byrne, 2022).

The review involved examining internal consistency by analyzing the relationships between data items and codes within each theme. It also assessed external distinctions between themes, ensuring they offered non-overlapping and insightful interpretations of the overall dataset. Themes were refined through a series of adjustments, such as merging, splitting, or removing certain themes (Byrne, 2022).

Defining and Naming Theme. In this phase, the thematic framework was refined by ensuring each theme was coherent, distinct, and relevant to the research questions. Themes were revised as needed, with names crafted to be concise, informative, and engaging (Braun & Clarke, 2006, 2012, 2020; Byrne, 2022). This phase involved selecting data extracts that vividly illustrate key points within each theme. Multiple extracts were used to represent the diversity of

perspectives and ensure the cohesion of the data. The extracts were not only reported but also deeply analyzed, linking them to their respective themes and the broader research context.

Producing the Report. Writing the report occurred fluidly throughout the analysis, as described by Byrne (2022), with the writing process embedded rather than left as a final step. Themes and codes were revisited recursively. The themes were then presented in a logical narrative, each coherent on its own while building on others to address the research question. To contextualize the data, a discussion of the results followed.

Limitations

This study has several limitations that must be acknowledged. First, the qualitative nature of the research may limit the generalizability of the findings, as the insights derived are context-specific and may not apply to other settings or populations (Creswell & Poth, 2016; Patton, 2014). Additionally, while the sample size is considered sufficient for thematic analysis, it may not capture the full range of perspectives within the broader community, potentially resulting in a narrower representation of views (Patton, 2014). However, the use of purposeful sampling aimed to enhance the relevance of the findings by including participants who could provide in-depth insights into the phenomenon under study.

The subjective nature of the study that cannot be entirely eliminated in qualitative research opens up another potential researcher bias during data interpretation (Creswell & Poth, 2016; Patton, 2014). Despite bracketing efforts, personal perspectives may inadvertently influence the analysis, and the themes identified.

Furthermore, the reliance on participant self-reports may introduce biases such as social desirability, where respondents may present themselves in a more favourable light. This can affect the authenticity of the data collected (Creswell & Poth, 2016).

Ethical Considerations

This study adhered to ethical guidelines to foster an environment of trust and respect, ensuring that participants' sentiments and intentions were treated with dignity (Creswell & Poth, 2016). Ethics clearance was obtained from the Research Ethics Board at Brock University (REB clearance, File number: 21-276), confirming that all research activities complied with institutional and ethical standards. To uphold these regulations and cultivate trust throughout the research process, the study implemented several strategies, including informed consent, the right to withdraw or refrain from answering questions without penalty, confidentiality, and member checking.

Recruitment and Informed Consent

During the recruitment, all participants received detailed information regarding the study's purpose, procedures, and their rights as participants, including the voluntary nature of their involvement and the option to withdraw at any time without penalty. Additionally, informed consent was obtained through a consent form, allowing participants to acknowledge their understanding of the study and agree to participate. Participants were informed of the potential psychological risks associated with recalling negative experiences related to the programming environment. To mitigate psychological risks participants were reminded before each interview that they could decline to answer any questions or terminate the interview at any time if they felt uncomfortable. Furthermore, participants were reassured that their involvement in the study would not influence their academic performance or grading in the MICA course. Finally, participants were provided with contact information for the Office of Research Ethics should they have any concerns regarding their rights or the conduct of the study.

Right to Withdraw or Not Answer a Question Without Penalty

Throughout the study, participants had the right to withdraw at any point without needing to provide a reason. If they chose to withdraw, they could simply email at a given email address. However, it was important to note that participants could not withdraw data during advanced stages of data analysis or after their data had been published in any format, such as in conferences, journals, or dissertations. Participants also retained the option to decline to answer specific questions during the interview without facing any penalties or affecting their participation status.

Confidentiality

Confidentiality is a priority in this research. Each participant was assigned a unique code to ensure that their identities remained anonymous throughout the study. Data collected during interviews, including audio recordings and transcripts, were stored securely on a password-protected computer. Only the principal investigator and faculty supervisor had access to this data. Participants were assured that pseudonyms and anonymized quotations would be used to safeguard their identities in the research report and resulting publications.

Member Check

The research design included a member-checking process where participants reviewed the transcripts of their interviews within 2 weeks of the recording. This process allowed participants to verify the accuracy of their contributions and to make any necessary clarifications, further enhancing the integrity and reliability of the data collected. Also, to foster transparency and provide feedback, a one-page summary of the research results was compiled and shared with all participants after the study's completion, allowing them to see the outcomes of their contributions.

Chapter Summary

This chapter details the study's methodology, designed to explore learners' interactions with their environment as they engaged in mathematics through programming. Guided by enactivism, a qualitative approach with phenomenological roots was used to capture the situated nature of participants' experiences.

Participants were purposefully sampled from Canadian university mathematics courses integrating programming. Data collection included semi-structured, stimulated recall interviews, field notes, and course-required reflections, enabling a comprehensive view of participants' computational and mathematical thinking. The combination of these methods ensured data triangulation, enhancing the validity of findings, while confidentiality and data security measures were strictly maintained. This methodology allowed for an in-depth analysis of co-actional learning, revealing how students shaped and were shaped by their learning environment.

CHAPTER FIVE: FINDINGS

This study aimed to explore the nature of interactions, or co-action, between learners and their environment as they use programming in mathematics. More precisely, the research examined how learners shape and are shaped by their environment through the practice of mathematics using coding. The inquiry was grounded in the embodied perspective of enactivism, particularly the notion of structural coupling, which explains the dynamic co-emergence of agents and their environment through reciprocal interaction. This conceptual framework also integrates ideas of occasioning (Kieren et al., 1997), where learners selectively engage with affordances in their environment; the mediating role of tools in shaping understanding (Hegedus & Moreno-Armella, 2010); and stigmergy, the decentralized coordination of action through environmental traces. To investigate these dynamics, the study was guided by one overarching question and three sub-questions:

1. What is the nature of co-action between mathematics learners and their environment as they engage in doing mathematics through programming?
 - 1.1 What perturbs learners in a mathematical programming environment?
 - 1.2 How do learners respond to the perturbations within a mathematics programming environment? (How are they shaped by the programming environment?)
 - 1.2.1 In what ways do perturbations in the programming environment prompt computational thinking?
 - 1.2.2 In what ways do perturbations in the programming environment prompt mathematical thinking?
 - 1.2.3 What perspectives do the learners form as they do mathematics within a programming environment?
 - 1.3 How do learners shape the programming environment for mathematics learning?

This analysis involved two distinct stages of thematic analysis: Level 1 coding and Level 2. The first section of this chapter presents a narrative description for each participant, annotated using the analytical framework and developed by synthesizing notes from the colour-coded analysis, participants' exploratory objects (EOs), reflections, and field notes. Guided by the conceptual framework, this phase was conducted prior to the Level 2. stage and aimed to gain a deeper understanding of each participant's stigmergic lived experience. The second section presents the results of the thematic analysis, examining shared experiences among participants engaged in mathematics through programming. This analysis followed Braun and Clarke's (2006, 2020) six-phase iterative and recursive approach, as outlined by Byrne (2022), enabling a nuanced exploration of participants' interactions with their environments. Each theme is accompanied by illustrative data extracts to provide a coherent account of how participants engaged with mathematical ideas through programming.

Participants

The six MICA students were Luca, a 2nd-year student double majoring in mathematics and physics enrolled in his first MICA course; Althea, a 3rd-year mathematics co-op student in her second MICA course; Ryan, a 4th-year student completing a combined major in physics and mathematics, also in his second MICA course; Nancy, a 4th-year concurrent intermediate/senior education student with teachables in mathematics and biology, in her third MICA course; Amanda, a 3rd-year concurrent education student majoring in mathematics, also in her third MICA course; and Gina, a 4th-year concurrent education student with teachables in mathematics and French, enrolled in her third MICA course (see Table 4).

Table 4*Participant Demographics*

Participant	Gender	Year of study	Program of study	MICA course level	Teachable/Majors
Luca	Male	2 nd	Double Major	MICA I	Mathematics and Physics
Althea	Female	3 rd	Co-op	MICA II	Mathematics
Ryan	Male	4 th	Combined Major	MICA II	Physics and Mathematics
Nancy	Female	4 th	Concurrent Education	MICA III	Mathematics and Biology
Amanda	Female	3 rd	Concurrent Education	MICA III	Mathematics
Gina	Female	4 th	Concurrent Education	MICA III	Mathematics and French

Participants' Narratives

Luca's Story. Luca's story was developed based on data relating to his final project for MICA 1. The final project was an open-ended assignment requiring the class to choose a mathematics topic of interest to investigate using programming [Environment: Mathematics Programming Environment]. They had the option of working with a partner, so Luca decided to team up with his colleagues for the class [Agents: Learners]. They reviewed the guidelines for possible final projects from a list provided by the course professor [Sensor: Marker-based (instructional scaffolds for project selection); Agent] and chose to investigate the Hill Cipher Encryption, because they did a similar type of cryptography and were interested in doing another one [Sensor: Marker-based (affiliation with prior learning experience)]. Actually, Luca said that he did RSA encryption before and "*found that to be a lot of fun*" [Emergent System Behaviour: Developing interest through prior experience with mathematical programming].

Thus, the purpose of his final project was to create a program in Visual Basic that would encrypt and decrypt a message using the Hill cipher algorithm [Environment Dynamics: Relationship between input and output]. The program allows a user to input an encryption key (four letters in the form of a 2 x 2 matrix) and enter a message in English to either be encrypted or decrypted. [Environment: Dynamics (mathematical input-output rules defined by encryption algorithm)]. A Hill cipher relies on basic linear algebra and modular arithmetic to encrypt messages [Actuator: Sema (mathematical tools)].

They approach the program by first working out an example of the mathematics on paper, “*like what the message should encrypt to and what it should decrypt to*” [Actuator: Sematectonic (pre-program modeling to conceptually clarify mathematical transformations; Actuator: Marker (Paper-based example used as an external guide for constructing and testing code.))]. Having previously understood the mathematical concepts through prior learning experiences, reviewed them in class [Sensor: Marker-based (review of formal mathematics instruction)], and engaged in “*a little bit of research and learning how to do it [the mathematics]*,” Luca had little difficulty working out the mathematics on paper. In ongoing dialogue with his partner, he then began developing steps for the program based on the paper example [Actuator: Sematectonic (translation of mathematical example into code structure)].

With the use of sub-procedures and functions as a requirement for the course [Sensor: Marker-based (recognition of course requirement to use sub-procedures)] and the learned knowledge that, “*it’s easier to structure it [the program] that way*,” the peers built their program using a lot of them [Actuator: Sematectonic (strategic use of sub-procedures to structure and simplify code)][Topology: Hierarchical]. They decided on which parts of the program needed sub-procedures by determining the sections that would need to be replicated throughout the

program several times [Sensor: Sematectonic (recognition of repeated code structure requiring modularization)]. He explained,

For example, this function encodes the message, so it takes an input message and input, and it returns the outputted encoded message as an output, both as strings. So, for example, this sub procedure here to convert the message letters into numbers. The sub procedure doesn't need to be encoded into the function. It can exist elsewhere because I also use it elsewhere in the code. When I decode the message. So, it doesn't really make a whole lot of sense to write out the code twice, rather than just having it in a sub procedure and then recalling it twice. [Actuator: Sematectonic (abstraction and reuse through procedural structure)]

Luca sequenced the program so that the sub-procedures were at the top so that they would load first when the program started to “*make the code more efficient - less chance of crashing,*” something he learned “*throughout the course of the course*” [Emergent System Behaviour: Realization of structural design improving efficiency]. He further recalled that he used other programming concepts based on the stage that he was at in developing the program. He stated, “*Like I get to here and I just kind of know, I’m gonna have to use a FOR loop. This is the only way it’s really gonna work*” [Sensor: Sematectonic (sensing programming state)] [Actuator: Sematectonic (use of FOR loop to structure logic)]. He learned how to use computational concepts like a loop, conditional, and array through labs and found that “*after taking the labs, you know when you’d have to apply these concepts*” [Sensor: Marker-based (prior lab experience informing intuition)]. He further explained that “*You develop like an intuition for how and when to use different objects*” [Emergent System Behaviour: Growth of programming fluency through applied practice].

In addition, Luca and his partner remixed codes from their last encryption program to satisfy the needs of the Hill cipher program [Sensor: Marker-based (recall of prior project code)] [Actuator: Sematectonic (adaptation of old code to new problem context)]. For example, Luca explained that the Hill cipher program needed to be encrypted mod 27 as opposed to mod 26 (for the other program) because mod 27 *“allows you to use a space when you encode something; you can encode like a whole sentence or a paragraph instead of just like having things all stuck together without spaces”* [Sensor: Sematectonic (adjustment based on contextual affordance of mod system)]. While the experience of doing an encryption program before provided him with general insight about developing the program [Emergent System Behaviour: Transfer of structural insight from previous project], he decided upon the exact mathematics... by doing some research (*“Google it, ... watch YouTube videos and read online forums”*) [Sensor: Marker-based (use of external digital resources)]. He added, *“once I had researched it, I knew this is how this is supposed to work, so these are the concepts I need to use”* [Actuator: Marker-based (selection of programming constructs from researched examples)].

He also stated that they played around and then added a few things to his program based on what they presumed a user of the program would need [Sensor: Sematectonic (perceived user needs through imagined interaction)] [Actuator: Marker-based (adding program elements as cues for user interaction or future program flow)]. In other sections, they *“tried multiple...ways”* while test-running the program to arrive at a way that *“would effectively work”* [Agent Dynamics: Iterative trial-and-error refinement to improve program functionality]. For example, after several attempts, they googled it and found the dictionary function to be effective in changing letters to numbers and found that they had to repeat the last letter of words with an odd number of letters to get the mathematics to work [Sensor: Marker-based (discovery through external resource search)] [Actuator: Sematectonic (adaptive implementation based on discovered behaviour)].

Luca pointed out that most of the changes that they made in their program were driven by the need to get the mathematics to work properly [Environment: Dynamics (adjusting code so input produces expected mathematical output)] [Agent Dynamics: Ongoing refinement of program structure to satisfy mathematical constraints]. In recalling the back-and-forth process that emerged during programming, Luca said:

You code up your program, you'll think it will work. Everything looks good, and then you run it and then it doesn't work, and then you're like, why doesn't this work? This looks like it should work. Everything seems fine. And then you spend like 2 hours trying to figure out what's wrong with it [Sensor: Sematectonic (state not as expected)]

[Agent Dynamics: Iterative debugging].

He further explained his deductive reasoning techniques as follows:

But I just kind of go through it and try to do like a little bit of deductive reasoning and being like, OK, well, if there's a problem with it decrypting, then there must be a problem with the matrices. Because then if it's not gonna generate, if it's not gonna decrypt it properly, then the inverse matrix there must be something going on with like the inverse matrix. So that's kind of what pointed me too here. And then, you know, you go through here and then you kind of like... I kind of try different things and be like, "OK, well, does this work for some values but not others?" [Actuator: Sematectonic (mathematical reasoning applied to debugging errors)]

While Luca found the process to be challenging, annoying and even aggravating at time, he found, "*this kind of thing [to be] fun*" because he "*generally really like to solve problems and stuff*" [Emergent System Behaviour: Positive identity development through engagement with challenge]. He believed that that type of dialogue with the computer to develop programs like the

Hill cipher helped him to learn more about both mathematics and programming because “*you’re forced to learn*” since “*you need the mathematics, ...and the programming for it to work...*”

[Emergent System Behaviour: Integration of programming and mathematical understanding through constraint-driven learning]. For example, without the process of developing his program, he would “*have never guessed that this type of encryption wouldn't have worked with a negative number*” or that “*there was such a thing in visual basic that would assign a letter to a number, and it's called a dictionary*” [Sensor: Sematectonic (unexpected insight triggered by program feedback)] [Actuator: Sematectonic (adaptive response to emergent program behaviour)].

In general, Luca found the course to be beneficial and implemented a lot of what he learned across other areas like in “*Excel and Microsoft Word, Microsoft PowerPoint, to like to execute things*” [Emergent System Behaviour: Transfer of programming thinking to other software contexts]. He also felt that while you could learn the mathematics behind encrypting and decrypting a message without ever having touched a computer, building and EO helps you to do mathematics faster and to understand how programming concepts and structures work, “*like when to use them and how they work*” [Emergent System Behaviour: Realization of programming as a tool for mathematical understanding]. Having entered the course with no experience in programming and not “*too much of a perspective on it*” he rated himself a seven for proficiency in using programming for mathematics learning, saying, “*I kind of like I knew it was out there and I knew people used it to do lots of stuff, but I didn't really know the depth or scale of it, and I didn't really know how useful it was before doing it myself*” [Emergent System Behaviour: Development of self-efficacy in programming through experience]. He believed that coding would help him in his future career and felt that the main thing that helped him to “*learn how to program was getting those hands-on application*” which goes further than “*a professor explaining to*

you how to code” [Emergent System Behaviour: Situated learning through applied programming experience].

Althea’s Story. Althea [Agent] is a 3rd-year Mathematics cooperative education major minoring in Economics. Her story was developed with respect to data about her final project for MICA II, in which she engaged with stochastic probability and deterministic difference equations to develop a program [Environment: Mathematics Programming Environment].

When Althea was doing her research for the assignment, she “*was really inspired by the things the professor (the instructor [Agent: Instructor]) did during the course ... calculating how populations grow*” using deterministic difference equations and stochastic probabilities [Sensor: Marker (intentional instructional markers that influenced agent perception)]. She googled the population growth of sand cranes, the topic that they did in class [Sensor: Marker (perceiving relevance through prior instructional cues or content)] and saw a paper written about it. Therefore, she wanted to “*do something real-world like in the real-world sense that really can show how this topic really works*” [Sensor: Marker (prior experience – connecting classroom activity to personal interest)]. She explained the following:

I said to myself, like, ” what is something that I'm curious about like that?” And I want to see how, you know, it affects how this topic can be used to interpreted, the population growth of that thing. And I was like, “OK greenhouse gases” because, you know, climate change is something that's really prominent and, like, is a very prevalent topic in the world right now. So, let me take this topic and see how I can tackle it [Sensor: Marker-based (identification of relevant, socially grounded modeling context)].

Thinking that stochastic probability gives a higher “*chance of calculating how populations grow when...[the]chances and like, the things that affect population growth*” is

known, Althea decided to investigate “*climate change*” using stochastic probability. More specifically, she “*decided to create a hypothetical circumstance in which...[she] use stochastic difference equations to demonstrate how two different rates of greenhouse gas emissions affect the atmosphere over a given period of time, when the rates change in specific conditions [in Canada]*” [Sensor: Marker-based (problem framing using classroom-taught concepts)] [Actuator: Marker-based (translation of conceptual framing into personalized simulation model)].

To go forward, she tried to find more data [Sensor: Sema perceiving the need for data to build program] to see how she could interpret the situation [Sensor: Sema (analyzing data to inform simulation setup)]. She said, “*I calculated on different probabilities and also looked at the data from the past 30 years before I settled on choosing the probability that I have that... looks more like what is happening right now*” [Actuator: Sematectonic (setting parameters using real-world data and engaging mathematical tools)]. So, with that probability she used “*the stochastic system equation to see how...*” she could “*project the population growth of greenhouse gases in the atmosphere in Canada*” [Actuator: Sematectonic (application of formal model for simulation); Environment Dynamics: Input (probability data → Output (projected emissions))].

Interestingly, she was “*completely stumped*” ... [Sensor: Sematectonic (perception of limits in implementing abstract models)] getting around the part of including the probability. Thus, she negotiated between two methods that she found similar, a stochastic difference equation and a deterministic difference equation. In the end, she settled with a stochastic difference because she “*couldn't bend my head around like the probability equation*” with the deterministic difference equation; however, she learned later that it could be done with the deterministic difference equation when she had dialogue with her professor on submitting her

program [Agent Dynamics: Navigating conceptual limits through comparison of methods]
[Actuator: Marker (selection of mathematically feasible implementation path)].

To start coding her program, she remixed standard codes that she worked on in class with her classmates and instructor [Actuator: Sematectonic (adaptive remixing of instructional code to meet evolving task needs)]. However, instead of doing one set of simulations, as she believed that she was expected to do, she programmed “*four sets of simulations...to better understand the variances and things like that because*” she was “*more interested in statistics side of things*” [Actuator: Sematectonic (expanding code functionality for statistical insight); Agent Dynamics (adjustment of program for personal statistical focus)].

Guided by the stochastic difference equation to remix her code, Althea noted that her model involved “*two random variables that are really important...: x_n which is the amount of greenhouse gas emissions after n months and R_n which is the random variable representing the monthly rate*” of greenhouse gas emissions. She further explained that she had to be mindful that there are two rates for R_n that she had to account for in her codes and their respective probability. That is, “*the probability that the monthly rate of emission is 0.05% is 75% [for non-summer months], while the probability that the monthly rate of emission is 0.3% is 25% [during the summer months]*” [Sensor: Marker (environmental variables guiding program design)]. She was a bit unsure how to remix her codes for the two rates and their probability [Sensor: Sematectonic (perceived gap in implementation strategy prompting instructor consultation)], so she asked her instructor for guidance. Her instructor was “*like ‘ohh just do a random number and then, you know, do an IF or ELSE loop into this. So, if any number before 75 comes out, you're going to like to return the value of R_n ... and then else just returned the other rates.’*” She found that that

made “*a lot of sense*” so she went ahead and did that [Actuator: Marker-based (guidance-structured implementation using conditional logic)].

Still following the stochastic difference equation, Althea coded n values, the number of months. She decided to code “*120 months so*” instead of 10 years (which she thought of at first) so, she could, “*in the latter portion ... calculate the expected value of greenhouse gas emissions after 10 years*” [Sensor: Sematectonic (recognizing need for time units to fit model goals)]. She then “*include a code for x of n itself*” using an “ *x _ cache because... [it was] a long loop of data*” and she had to store the values of x and “*go in and retrieve things like that*” [Actuator: Sematectonic (data caching to support extended loop execution)]. She continued coding using the initial value of GHG emission as 672 megatons of carbon dioxide for the first month according to what she calculated before actually started programming [Actuator: Marker (setting initial value based on real-world data)]. She said, “*I coded the initial value based on the value of greenhouse gas emissions in the year 2020 because that’s my base year for all my calculations I did in this project*” [Sensor: Sema (perceiving the need for appropriate real-world anchor data)]. She used an IF/Then statement to code the amount of GHG emissions in two situations: return the result of 672 when n is equal to 0 or otherwise, calculate using the stochastic difference equation. She did most of the programming by “*changing it [codes] up to meet... [her] expectations for this*” [Actuator: Sematectonic (adaptive structuring of functional code)]. She then wrote code to run each set of simulations. She said, “*I did 10, I did 20 and I did 50 just so we could get more variants in my calculations.*” In her report, she displayed a graph for each set of simulations. Due to the randomness of the situation her graphs kept changing when she reran them, but she said, “*but I chose these [for her report] at the end of the day because they were more like there were more forthcoming, they were more common in my results*” [Sensor:

Sematectonic (pattern recognition from repeated stochastic outputs)] [Actuator: Sematectonic (representing results for communication)].

Once Althea's base codes were done, she went "*in and changed a few things to get more results*" [Agent Dynamics (revising program to deepen exploration)]. Noteworthy, prior to starting her program, she "*kind of like set an idea of the things...[she] wanted this assignment to bring out*" which included setting questions [Sensor: Marker-based (anticipatory framing of learning goals)] [Actuator: Marker-based (setting framing of learning goals)]. As she was mostly interested in "*the statistics side of things,*" she focused on adjusting her program for her audience to see that "side of things" based on questions she had beforehand or as prompted by her program [Agent Dynamics: Personal curiosity driving iterative expansion of analysis]. She said:

So, for this, the 5th simulation, this is where I was like, OK, this is something really interesting because now you can see the dispersion of and the way the population grows over, you know, 120 months... And then I just went in and changed this depending on, you know - depending on my simulation, so run this again and then I have my 10 simulations now so we can really see the variance going on and the dispersion of the data in this situation. So, this was the very first part that I found interesting and then it made me think, OK, what else can I do to make this more interesting? [Emergent System Behaviour (emergent insight driving continued exploration)]

Althea went ahead and plotted the sample mean and the expected values [Actuator: Sema (integrating statistical visualization to understand mathematical behaviour)] because she wanted to explore them, given "*that the teacher explained a little in class about plotting the expected value with the sample mean and comparing them to really see if the data is like measuring up to what you would think it is*" [Sensor: Marker-based (prior instruction prompting extension of analysis)]. And once again, she felt like people would be interested in real results. She said, "So

then I plotted my sample mean for this data and I just also plotted the expected value because I was like, “OK, this is something that, you know, people want to see” like this is, I would say, real: like this have to be considered when you’re doing things like that [Sensor: Marker-based (perceived audience expectations for authentic output)][Actuator: Marker-based (adapting output presentation to align with perceived audience expectation)]. The expected value was calculated using formula they (as a class) “derived in the lectures [Sensor: Marker (recall of class-derived model for coding)], $E(x_n) = x_0 \cdot [p(1 + r_s) + (1 - p)(1 + r)]^n$ ”, which she coded into the program [Actuator: Sema (implementing expected value formula from class into simulation)]. After graphically comparing the sample mean against the expected value for each simulation, she concluded that the “*sample means for the 4 simulation cases are great estimators for the expected value of the total GHG emissions 10 years from now*” [Emergent System Behaviour (seeing patterns form after running many tests)].

Furthermore, with her program, Althea was curious about “*probability at which the total amount of emissions reaches 780 megatonnes within the first 10 years*” under the assumption that, “*780 megatonnes of carbon dioxide equivalent is harmful to the atmosphere*” [Sensor: Marker-based (social consequence guiding computational inquiry)]. Going by her “*calculations we would reach that within like 11 years and one month...*” but she wanted to know “*what is the probability that Canada gets there before the timeframe of 10 years.*” So, she modified her model to check that [Actuator: Sematectonic (modifying model to test hypothetical policy window)]. Althea attributed her decision to explore to her instructor [Sensor: Marker (instructor influence prompting further exploration)]. She had gotten to the end of her program, when she was prompted to do further exploration. She said:

“...at that point in my head I was done. I was like, ‘OK, this is the end of my assignment.’
But then I asked myself, ‘wait, this is something we have discussed, and I want to actually

see this happen'. What...like how long? So even if it takes about 11 years and one month, because these probabilities are random like nobody can ever like expect them coming, what are the chances that this will happen in less than the amount of time I predicted to happen? I would definitely say because of the discussion we had in class because I found the teacher to be interesting because he thinks a lot when it comes to these types of programs. There were so many things that I would say weren't in the course program or I would say in the course content but he wanted us to think so I was thinking and I was like....let me think from the point of view of Jesse, that was his name, 'how can I explore my project more' cause at this point I was like I think I'm done, but let me see if I can consider something he has asked before. 'Like how I can actually measure, you know, this kind of situation?'" [Agent Dynamics: Instructor-evoked mindset sustaining inquiry beyond original scope]

Althea also explained that she estimated “800 or 861 – one of those two numbers” for the minimum potentially harmful megatonnes of carbon dioxide in the atmosphere before she predicted 780. She said that “everything was good and fine” before she “decided to take on this task of finding the probabilities that this will reach this amount within 10 years.” She was “just getting zero as a probability” when she ran the numbers in her simulation. But she “wasn't really interested in having zero... even though zeros like it's a real situation.” She wanted something realistic to Canada; so, she “was like, “Canada has never experienced above 757, above 760, so 800 might be a far stretch. Let me come back down.” So, she adjusted her number to 780 [Agent Dynamics (refining parameters to improve realism based on historical constraints)].

To adjust her code, Althea interacted with the programming environment by “changing some things to comments..., moving some things around..., or completely scrap[ing] things out.” In addition, she had “two extra documents open” that she called “background work just to like to

see how things work.” She explained, “*I remove things here and move things there before I actually work*” [Actuator: Sematectonic (iterative restructuring for program debugging and refinement)]. Seemingly lost in her programming, unaware she demonstrated how she interacted with her programming environment. The following excerpt shows:

For example, if I had 10 simulations, then it would divide all the values for all the 10 simulations by 10 just so I would have one straight line which we then see in this plot here. [taking to herself quietly] “Let me run this and see if this will work. No, no. Let's go back to my documents OK. Scattered mean, sums, OK” [stop taking to herself quietly] ... So, I actually just, I also made this document just so I could copy my data directly from spider without it being interrupted... [started taking to herself quietly] - Let me paste this. This is more like it. [Playing around with program while talking] So. Let's go here. So, uh, wait, let me make one comment first so we can see the effect of one against the other before I begin to go in... [Environment: Dynamics (real-time interaction between simulation outputs and system adjustments)]

Althea explained that she gained a better understanding of how certain programming structures worked in Python through her method of interacting with the programming environment, which involved using background work to test and explore code. In one instant she “*understood how standard deviation works in regard to Python*” [Emergent System Behaviour (new understanding emerging from programming interaction)].

As it relates to programming, while Althea thought that she was “*not a programmer*” and “*not good with computers at all*,” she did not seem to encounter many challenges developing the codes to carry out what she wanted because most of the codes were similar to what she done in class [Sensor: Marker-based (confidence drawn from prior classroom programming patterns)].

Furthermore, she consulted her teaching assistant when necessary. She said, “*at some point, I even had to message my teaching assistant for the things we didn't particularly cover, but I just wanted to ask and find out how I can do this in the course*” [Agent: teaching assistant (supportive agent in programming environment)].

Giving an actual example where she asked for help, Althea said “*the last question wasn't something that we actually like, covered, covered but then I was really curious as to see how I could do this. So, I had to ask and say like, “oh, how can I compute this, you know, and show this in my work*” [Sensor: Sema (seeking targeted clarification to adjust program development)].

Furthermore, although Althea did not work closely with many of her peers, she picked up on programming techniques while programming collectively with her “*one friend in the class.*” For example, she got the idea of using background documents from that partnership [Agent: Peer (peer agent influencing strategy selection)]. She explained in the following excerpt:

I had a friend in the course who like I whenever we were doing studying together, things like that. I would see the way his mind is working and he's just like saying, like, oh, I'm going to see how this works. And I said I could learn a thing or two from him. And then I started doing that in my assignments. I would take all the bits and pieces that teacher had given us to work with...I would take that and I'm like, “I need to understand the framework before I go ahead” [Sensor: Marker-based (recognition of instructional fragments as scaffolds for understanding)] and then started doing the method my friend used... Like just take a little bits and pieces and actually understand how Python works, which is why I have the background works. And then once I understand the function of something, then I'll take the actual part of the code I need and also put it there and then see how it works and then bring it back change of values if I need to, you know, things

like that [Actuator: Sema (applying strategy to transfer and test segments of code)]
 [Actuator: Marker-based (modular code scaffolding from instructional examples)]. *So that was definitely how I played around with the probabilities to decide on 0.005 percent and zero point 3%* [Sensor: Sematectonic (exploratory feedback loops guiding parameter refinement); Agent Dynamics (personal experimentation based on shared strategy)].

Althea's mode of interaction suggested that the mathematics mostly guided her interaction in the programming environment. In agreement, Althea explained:

I don't think the purpose of my assignment really changed as I worked on it. I think I had - I feel like maybe it would have changed if I didn't have my mind set on getting a particular expectation. ... I want to know what happens after 10 years and I'm going to code according to that. And then I kind of just worked with everything to see how I could get my result. So, I definitely don't think Python was leading me. I think I was definitely leading Python. "I want this. Can you show me how I can get this and like what affects this?" Things like that [Agent Dynamics (program goal remains stable while tools are adapted to meet need)].

Althea further described the "long process" of interaction that saw her receiving "*so many error messages*", values that did not correspond to values that she researched – "*like actual data that Canada has actually collected... on the official Canada government website*" - and numerous adjustments of her codes and monthly rate of emission [[Sensor: Sematectonic (error-driven awareness of misalignment with empirical data); Actuator: Sematectonic (incremental calibration based on program output feedback)]. She was determined to get a good reflection of the values she was working with, so she continued in active dialogue with her computer [Agent Dynamics (persistent debugging and refinement)]. She explained, "*I said, 'there is no way Python is going to deceive me', I would say, 'into telling me... the expected value is like 1000 when Canada has*

never experienced such levels.” She further explained that “*Python was able to show me like ‘ohh these are the values you wanted to work with. I’m just telling you what the answers you’re going to get are. So now you modify and then come back’*” [Environment: Dynamics (input/output relationship prompting feedback loop)]. During the process, she learned that about perception and things like that. She expounded, “*OK, this is really teaching me that like I can’t pull values out of thin air. I actually have to see how this works because there is real data before this. So, anything I’m doing should align at least a little for it to be a little more realistic*” [Sensor: Marker (real-world constraints reinforcing programming choices)]. Moreover, Althea had an authentic real-world experience, even though she had to doing the “*math over and over again*” in python to get a realistic expected value for GHG emission. She said:

Honestly, like working with real life data and just trying to create something out of it is kind of a new concept to me... [Emergent System Behaviour: Real-world modeling experience contributing to authentic engagement], I was like, let me take this from a mindset of I’m assuming I actually work for the Canadian government, and I’m trying to propose ways or trying to, like, see how you know emissions will increase if, you know, this is what’s going on every month, things like that. So, I was like, I can’t have values that are like all over the place. So, it was difficult at first to like kind of rein in my values [Actuator: Marker (drawing from real-world experience to fine-tune model accuracy)].

Having been skeptical about programming for mathematics learning before the course, Althea ended with the perception that she would not have learned all she did if she had programmed by hand. In reference to manipulating her program to determine a realistic expected value for GHG emission, she said:

I feel like if I was doing this by hand, I definitely would not be able to see this far ahead... Mainly because I generally I feel like there’s only so much you can do with your

mind in a certain frame of time. But then if you're given a computer and things like that, it's much easier for you to get like for example a wide array of values or like more.... I would say like one simple computation like give you a wide array of values, whereas if I'm sitting down and just like writing out something I'm just going to be sitting there like, "OK" [Emergent System Behaviour (amplified insight through computational modeling)].

She rated herself at a 7 for proficiency in using programming for mathematics learning, given that she believed that she could have done more within the course, even though, judging from the feedback she received from her professor, she thinks she did well with the assignment. Thinking of herself as an introvert she believed interaction within these types of courses is advantageous for learning [Emergent System Behaviour (realization of learning progress through reflection after system interaction)].

Ryan's Story. Ryan's story was developed based on data collected about his final project for MICA II. Ryan was sitting with his classmate when the task of finding a suitable project to work on was discussed; so, he decided to team up with his classmate to accomplish the task [Agent: Learners (collaborative team)]. They agreed to investigate von Neumann's middle-square method, a program similar to the logistic program that they did [Sensor: Marker-based (link to prior classroom experience)]. They devised a plan of execution, which saw Ryan focussing primarily on the coding and his teammate primarily on the reporting [Agent Dynamics (division of roles)]. For von Neumann's middle-square method, they aimed to determine by programming if this method of finding prime numbers is uniformly pseudorandom, generally pseudorandom, or non-random [Sensor: Marker (initial guiding question)]. To accomplish this, they contrasted the middle-square method against a uniform random distribution from Python [Actuator: Sema (comparison using computational tools), Environment: Dynamics (input-output comparison to determine randomness)]. They decided beforehand that if the middle-square

method exhibits pseudorandomness comparable to uniformity, they will do “*further investigation to determine the nature of this pseudorandomness, in particular, its deformation from uniformity.*” On the other hand, should it exhibit nonrandomness, they would “*do a further investigation to determine the k-cycles where randomness blatantly fails, as well as outline their properties*” [Actuator: Marker (conditional plan to guide next steps)].

To get started designing the project, Ryan and his teammate, “*worked it [the mathematics] out for a little bit on paper*” [Actuator: Sematectonic (mathematical reasoning tools)] and figured out a formula that “*the entire project revolves around*” [Actuator: Marker-based (construction of a central model aligned with the task’s instructional framing)] and translate it to programming. Ryan explained:

In order to figure this out, you have to kind of rationalize like a more practical thing that you do on paper into programming [Actuator: Sematectonic [(translation of mathematical concept to code)]: The von Neumann Middle Square method - what it is? - a method of generating random numbers. You take a number that's even digits long. Say, I don't know, 1234 as a four-digit long number, and then you square it, right? And then you take its middle. That means you cut off the first two and last two digit and you're just left with another 4-digit long number. That's your new number. But that works nicely on paper, but not in programming, you see.... It doesn't make sense to ask what's the middle of a number unless you know how to find the middle using mathematical formalism [Sensor: Sematectonic (recognizing limitations in translating mathematics to programmatic logic)].

Ryan recalled how the problem perturbs their thinking to come up with a mathematical plan for the program:

We walked through the idea - What is the middle square method? So, you take a number,

and you want to square it, and you get this other number and then you want to take its middle. But the problem here is how do you get the middle? How do you select the middle using like a machine [Sensor: Sematectonic (programmatic constraint sensed by the agent)]. How do you formalize that process into something that a machine can do or understand? And so here it is. Basically, what we do is we find a discrete interval, a way of counting the digits and we cut off the first quarter and last quarter by dividing by some exponent of 10, and then taking the floor. And then after that, if you take the difference, you're left with just the middle part [Actuator: Sematectonic (algorithmic modeling strategy)], so that that's the kind of lengthy way of describing it, but that's the mathematical mechanism involved in doing this very simple thing of cutting off the edges of the number.

Thinking with the mathematical plan outlined during the design phase of his program, Ryan and his partner “*realized quickly that the von Neumann Middle Square method was non-random [and] ...immediately wanted to program... in such a way that it was going to tell*” then “*the length of cycles and find cycles for*” them [Sensor: Marker (insight prompting program development goal)]. This in turn prompts them to use modular arithmetic to cycle the values [Actuator: Sema (mathematical tool to structure program behaviour)]. Ryan determined the tools within Python that were necessary to develop their program [Sensor: Sema (identifying programming needs)]. For example, he used the INT function to round “*numbers to the nearest lower integer, so all the decimals get cut off*” [Actuator: Sematectonic (implementation of data handling mechanism)]. Also, as the plan for the program dictates, he employed both programming and mathematics concepts. For instance, he used “*a really big FOR loop,*” to execute the process for collecting the same type of data repeatedly and used modular arithmetic

for the index number of a time sequence [Actuator: Sema (algorithmic implementation for repetition and structure)].

Furthermore, Ryan explained that their program emerged into two main sections that he did not necessarily plan out beforehand: one section he called “*the control section*” and the other section he called “*the heart of the program... (doing all the heavy lifting – the work)*” [Emergent System Behaviour (structure emerging from iterative development)]. They made that “*significant alterations of the design about halfway through*”. He mainly used IF statements in the control section that he considered as “*almost on/off switches*” or “*a big red switch*” to manage “*the actual program itself*” [Actuator: Sematectonic (control structures for decision making)]. Ryan explained that the design of the program into two sections emerged organically by his remixing the code from his earlier logistic program [Sensor: Marker-based (re-use of prior instructional code)]. He said, “*we started with that and then we built on to it to make it suited for our problem [Actuator: Sematectonic (repurposing and adapting prior code as an occasioning tool)]; so, we mutated the original code that we had had from that previous until it worked for our purposes*” [Agent Dynamics (remixing and evolving prior work)]. In the following excerpts, he gave a specific example of this organic process in how he rationalized the placement of functions:

it was organic, like what dependencies does this function have? “Is it important that it be inside here? And in these cases, the answer was no, it's not because it doesn't depend on X knot. So, there's nothing in any of these expressions that needs to be inside this loop. So, putting it in is wasteful, and it causes too much - It causes extra stress and takes more time to run the calculation than if you do that. There's no point.

He also described the process of developing his program “*a process of amendment; a as*

a result of our trial and error” [Dynamics: Agent-environment feedback loop]. His process is explained further in the following excerpt:

It was a process amendment because we started with one model which was from a previous part, a previous assignment, right. And we kept amending it. And through this constant process of amendment - I think even when we were running data, I was amending the program as I noticed things, constantly amending and this turned out to be just the most optimal solution [Emergent System Behaviour (refinement through real-time system interaction)].

A big part of his amendment process “was intuition as a result of watching the program run” and seeing the outputted information [Sensor: Sematectonic (real-time feedback from environment prompting reasoning)]. So, he ran “the program and... look at it and what's it doing..., ... And then...think..., how can we do this better? Is there a way we can do this better? ... Do we really need all these...? No, we don't. So, let's change it.” [Actuator: Sematectonic (program refinement based on environmental feedback)]. Ryan gave an example of the program outputting graphs:

So, what it was doing was it was outputting a graph at every time the WHILE loop would run. And that meant it would stop every time the graph came up, the program would stop and then you'd have the option to save the graph. But it was at that point we thought, well, we, we're gonna use all of like 4-3 graphs or two graphs here. Why do we need to output a graph every single time? [laugh]. So, that's when we changed it [Actuator: Sematectonic (optimization of program output based on evolving needs)].

In general, Ryan’s interview reveals that, beside being guided by the mathematics, which was “relatively stable throughout”, the “two big drivers” behind his sectional design were the

“*ease of use,*” and a desire to optimize or speed up the program [Sensor: Marker (performance goals guiding refinement)]. As he programmed, he wanted to easily “*find and change specific things in the program without*” having to change multiple parts of his program.

Definitely ease of use. Ease of use because this is really easy to use: you change these, you only have to change them once. Here they're easy to find. They're always at the very top and these change a whole bunch of different little things in this program like you see them mentioned all over the place here instead of going through and putting a number in each one of those spots, which is a pain. I don't want to do that. It's a variable...put it at the top and change that one variable and then it does all the work for you, you see
[Actuator: Sematectonic (use of variable abstraction to increase efficiency and maintainability)].

Paired with “*ease of use,*” Ryan’s desire to optimize the program for speed. As Ryan noticed that parts of his program were slow, he changed them [Agent Dynamics (adaptive refinement for performance)]. For example, he favoured a FOR loop over a WHILE loop that he was priorly thinking-with because when he “*went back and optimized the program to make it run faster, it was taking a long time*” [Sensor: Sematectonic (runtime feedback triggering performance evaluation)]. The current state of the program prompted him to ask, “*why don't we figure out which is the method that runs the fastest*” and they determined that the FOR loop was “*the fastest one that works if you know the beginning and end*” [Sensor: Sema (evaluating execution efficiency)]. Another change that he made was declaring a variable when he “*noticed*
[Sensor: Sematectonic (runtime feedback triggering performance evaluation)] *that calling a function repeatedly was causing a slowdown.*” He said, “*so I had the idea: Instead of having this*

function called over and over again, why don't we assign it to a variable once so that we save time at the cost of space” [Actuator: Sema (efficiency trade-off in design logic)].

Ryan believed that watching his program run and process data while making changes was the best way to ensure efficiency [Sensor: Sematectonic (perceiving live performance to guide decisions)] as *“there's no real good substitute other than putting it [the program] into the field, trying to run it as... it is intended to be run in exactly the situation.”* They *“started collecting data with an inefficient program, and then... [as they realized the] ...inefficiencies [they] ... programmed differently... [for].it to work better for... [their] needs”* [Actuator: Sematectonic (modifying program based on environmental interaction)]. He found that *“it required some digging around”* and thinking in order to optimize the program. He said, *“thinking when I'm programming thinking when I'm not programming as well. A lot of thinking”* [Agent Dynamics: Persistent cognitive engagement with problem-solving].

Interestingly, given that Ryan was remixing his previous code from another assignment that he built *“from nothing, with the help of the course material”* [Sensor: Marker-based (initial instructional scaffolds)], his recollection of his creative process shows the improvement of his programming technique. He was able to see many inefficiencies in his prior program as he made amendments to his first codes for *“ease of use”* and optimization purposes [Agent Dynamics (retrospective refinement)]. He said, *“We actually went through started collecting data with an inefficient program...we realized as we were collecting that these were the inefficiencies and that it could be programmed differently to work better for our needs.”* [Actuator: Sematectonic (iteration and refinement from testing)]. However, despite his changes to satisfy those criteria, he never had challenges learning program. He thought it was *“rather straightforward,”* making *“pretty good sense”* [Emergent System Behaviour: Growing perception of programming as

accessible and logical]. He said that programming is “a procedure you see.” Elaborating further, he said the following:

the program is a bit like a production line for the data. You put some piece of data in - raw data - or some expression in and step by step one thing happens to it, one thing at each spot. And so, each of these lines has one purpose only. This line has one purpose. This line has one purpose. This line has one purpose. And by this step-by-step process you arrive - your data is manipulated into something that you want, something desirable [Environment: Dynamics (stepwise transformation of data)].

Part of Ryan’s resolve to program for “convenience” or “ease of use” was recognizing that “*the machine can do...tedious work*” for them [Emergent System Behaviour: Recognition of automation as a functional affordance of programming]. Given that they had a limited timeframe to submit the program, he recognized that making his program convenient could help them maximize their time. He said, “*We want the program to be convenient to use because we don't want to waste time and have to trudge through something we don't want to do, like change all these instances of this number right in the big thick program*” [Actuator: Sematectonic (designing for maintainability and ease of change)].

One important thing that Ryan “*learned new from working on this project*” that he thinks can easily be missed, “*is that even if you have two functions that are on their own inefficient, sometimes...when you use them together, ...they're more efficient together than any other options for functions that you would have*” [Emergent System Behaviour: Recognition of emergent efficiency through composition]. He believes that he probably would not learn that outside of a programming environment because “*you don't run mathematics.*” He said that programming

makes mathematics accessible to the real world [Emergent System Behaviour: Realization that programming enables execution of abstract mathematics]. He explained:

All of mathematics is always there all the time. But it's inaccessible to us, you see. It's all out there, but we just can't get at it, so we program instead, and we use math. We push it into the program...then we run the program, and the program is kind of like an imperfect reflection or partial reflection you could say of some kind of mathematical ideal world [Emergent System Behaviour: Realization of programming as a medium for expressing mathematical abstraction].

In response to the question to give his overall perspective of programming for mathematics learning, he explained in the following excerpts:

Well. It [programming] really is a flawed study. A study of flaws, a study with lots of flaws. But that's the whole point in some way... because mathematics is too perfect for the real world. And for real applications. And so, bringing it down to Earth requires that we introduce approximations, which is in the pure mathematical world, a flaw, right approximation is a flaw. It's not perfect [laugh]. For our practical considerations, it doesn't really matter. We want something that we can use, which means it doesn't have to be perfect. It just has to be close and programming for mathematics really is about that. It is about approximation. It's about flaws [Emergent System Behaviour: Conceptual shift toward valuing approximation in applied mathematics].

He further explained that as a mathematical physicist (a theoretical physicist) who wants perfect solutions, before programming for mathematics, he was skeptical about it because he doubted that programming could give him perfect solutions. However, from his experience with programming, he had to “come to terms with the fact you can't always get perfect solution.” He

is of the impression that programming “*sacrifices perfection for usefulness.*” He said that while he is “*still very much interested in pure solutions to problems,*” by “*being forced to program*” he has “*learned some appreciation for what it is good for what it does, how it can be useful*” and now that “*the world is only getting deeper into computers,*” he supposed he might use it one day. Also, while he understood the usefulness of and “*know enough about how to use functions and loops to get what programming,*” he felt that he “*barely scraped the surface.*” Thus, he rated himself an eight for proficiency in using programming for mathematics learning, up from a five before he started the MICA courses. He attributed the improvement in score to the opportunity he got, “*thinking about trying to solve practical problems using mathematics and abstractions*” [Emergent System Behaviour: Realized growth in programming fluency through applied mathematical inquiry].

Finally, he used the metaphor of a toolbox to give overall perception of programming. He said of programming:

It allows you to get a better insight of how to use the tools of programming because if you know what you want to do and you're given a toolbox, then you're going to start rummaging through the toolbox to see what you what want to do, if you can do something with the tools, right? Because I was given this sort of objective and handed this toolbox It helped me realize the way to use these tools and how to best use these tools which I wouldn't have realized any other way, I don't think [Actuator: Marker (tool-oriented learning through task-driven exploration)] [Emergent System Behaviour: Metaphorical framing of programming as guided exploration of available tools].

Amanda’s Story. Amanda [(Agent)] is a Year 3 concurrent intermediate/senior education student, majoring in mathematics. She did both MICA I and II before doing MICA III.

Her story was developed based on data relating to her final project for MICA II, the structured DNA sequence assignment, she developed her program using Python Programming Language. The assignment required the class to create a program to read the DNA sequence, clean and count the number of nucleotides, and then determine a specific proportion of nucleotides based on outputs from the program [Environment: Dynamics (mathematical structure and algorithmic expectations embedded in the task)]. She was immersed in the learning environment with her peers, instructor, and teaching assistant (TA) [Agent: Instructor, Peers, and teaching assistant (supporting Amanda's development through social interaction and feedback)]. Her interaction within her environment was prompted by a collective discussion with her class about the assignment and a piece of code that she was given [Sensor: Marker-based (instructional starter code as external scaffold)]. The code was given to the class because they had not learned how to get the program to read a file. She stated the following in her interview:

So, we actually started designing it a bit in the class. We, the teacher, just asked us, how do you think you would do these things? And then we discussed with classmates how we think we would do it: write the code algorithmically and like what you would need and then we had a class discussion about it after, so the initial design of it I would say was like a class part, but it was very vague I would say. So, it wasn't like specific: this is the type of code like, it wasn't the exact syntax, it was more like "you would need a loop here or Oh we think you would need a conditional statement here" [Sensor: Marker-based (collaborative sketching of algorithmic components)].

Following the discussion, Amanda subsequently developed her program in response to mostly perturbations from her environment, but acting on the discussion in class was not straightforward

for her. From the in-class discussion, she gathered that she needed to *"first get it [the programming] to read a file and then have it read the script and recopy it"* [Sensor: Marker-based (interpretation of peer-instructor discussion guiding initial planning)]. Assessing the state of the code given, she determined that it was not sufficient for the code a program to read and recopy the file; she determined that it could *"read a file and call up a file,"* but she needed to learn how to *"have it read the code"* to recopy the file, *"like what the Syntax would be"* [Sensor: Sematectonic (recognition of incomplete system behaviour prompting further inquiry)]. Therefore, she decided to do additional research online to learn how to get the file to recopy the script [Actuator: Marker-based (external digital research guiding problem resolution)].

In researching she determined that a FOR loop with a nested IF statement would allow her to recopy the file by reading the required genomes into a new string [Actuator: Sematectonic (algorithm design to fulfill program goal)]. However, before settling with that code to read the genomes into a new string, during the discussion session in class, she thought to delete the numbers and the spaces from the DNA sequence, but then she rejected that thought when she reached the lab and was working on her program with others [Sensor: Sematectonic (evaluation of affordances in alternate strategies)]. She recognized that she did not have sufficient programming knowledge to do what she thought of before. She noted, *"I realized, oh, it would be easier doing it the other way that other people were thinking on doing it"* [Emergent System Behaviour: Strategy shift through peer-based insight and self-assessment]. Subsequently, she developed the section to read and recopy the file using the programming knowledge that she already had. Amanda explained, *"I realized I didn't have a method of deleting something from a file, but I knew a way of creating a new string"* [Actuator: Sematectonic (leveraging known strategy for implementation)].

After creating the first section, Amanda did not have many challenges completing the rest of her program. She had to consult her teaching assistant once, but other times she applied various program concepts and practices, she learned from doing other programming tasks beforehand [Agent Dynamics: Drawing on prior programming tasks to complete new assignment]. In relation to counting the frequency of nucleotides into a string using an IF/Then statement, she explained the following:

So, it's counting the frequency. So, I had used similar tools like this before ... counting tools before, just not in the form of it reading a string, but rather if, like. ... Yeah, I've used similar counting tools before saying IF something is equal to this then add to the frequency one. So, because I've used this tool before and other code I was using again here. So, like the idea of reuse, modify, create [Actuator: Sematectonic (reuse and adaptation of known conditional logic)].

Furthermore, Amanda consulted her teaching assistant once when the slow movement of her program to displaying her graph prompted curiosity. Amanda explained the following:

It took it a really long time to process the data and to display these graphs here. So, I was talking to my TA [teaching assistant] about it, and then they were "like, oh, you should actually use an array." And "I was like, ohh yes, that would be a better idea" because they were like, "if you use an array, then it's not going to be trying to recreate a new graph every time," which is what it was doing before. And so, the array allowed it to process the data quicker and I'd used arrays previously. I was just using a different method for plotting the graph [Sensor: Sematectonic (recognition of inefficient behaviour prompting expert consultation)] [Actuator: Marker-based (TA suggestion adopted to improve efficiency)].

Amanda designed her program using different sections following the sequence of the assignment because, *"the assignment was almost already set up in that algorithmic thinking way"* and because she found it less overwhelming to think of the program in parts and for ongoing debugging [Sensor: Marker-based (task sequence shaping code organization)] [Agent Dynamics: Managing complexity through modular design].

For the mathematics investigation section of her program, Amanda also did not experience many challenges as she was familiar with the mathematics that was reviewed in class. She developed the required proportion in her program and made one mathematical change to it based in the results it was giving [Actuator: Sematectonic (refinement of mathematical model based on output)].

In general, in conversation with her peers, instructor, and teaching assistant, Amanda made a few changes, clarified certain aspects of her program, and assisted where necessary [Emergent System Behaviour: Collaborative refinement and distributed problem solving].

Her overall perception of programming is that it is *"a great tool"* to learn mathematics in a concrete way. For one thing, she believed that although debugging did not contribute to new knowledge about programming for this assignment, it could help her *"overall mathematics learning in the sense of applying the idea of double checking your work throughout the process"* [Emergent System Behaviour: Cross-disciplinary transfer of programming practices into mathematical habits]. In addition, comparing results with her peers reinforced the idea that *"there are different methods of doing things, but you can get the same output"* [Emergent System Behaviour: Recognition of multiple solution paths through peer comparison]. She also believed that programming *"could make things go quicker, and it's a good way to also test conjectures,"* hypotheses, and double checking with results obtained by doing mathematics by hand [Emergent

System Behaviour: Recognition of programming as a mathematical experimentation tool]. Two big new takeaways for Amanda in terms of programming concepts is that she learned how to read a file and that *“arrays are quicker than using other methods”* she used earlier [Emergent System Behaviour: Integration of new programming tools into conceptual repertoire].

In reflecting on her self-development, Amanda believed that in comparison her nervous self before starting MICA courses, someone who knew nothing at all about programming (who *“was not quite sure about stuff,”*) she is a lot more confident in thinking computationally to use computer science for mathematics [Agent Dynamics: Development of computational confidence through iterative experience]. She attributed her confidence to having a visual knowledge of programming obtained from the MICA courses. She could even visualize like how she could use programming to teach certain concepts in mathematics like function and graphing and recommended programming for other mathematics learners as a *“character-building tool”* among other things [Emergent System Behaviour: Visualization of programming for pedagogical application]. Overall, she gave herself an eight out of 10 for using programming for mathematics learning when she compared herself to her classmate. Complimenting herself she said, *“For example, when we created our...For our final project, we had to make a lesson. So, with that I did that all by myself and didn't rely on anything else really. Not too much”* [Emergent System Behaviour: Expression of self-efficacy through independent project design].

Finally, she commended the teaching assistants and instructors' approach to helping, saying *“Leading questions, as opposed to just giving the answer and pointing you in the right direction where to explore your errors was also beneficial”* [Sensor: Marker-based (instructor cues prompting learner reflection and action)] [Actuator: Sematectonic (follow-up action based on reflective guidance)].

Nancy's Story. Nancy is a 4th-year concurrent intermediate/senior education student, with Math and Biology teachable. She did both MICA I and II prior to doing MICA III. She participated in this research with data in relation to Assignment 3 from MICA III, the structured DNA sequence assignment; and developed her program using Python Jupyter programming software. The assignment required the class to create a program that could read a DNA sequence, clean it, count the number of nucleotides, and then determine specific proportions of nucleotides based on the program's output [Environment: Dynamics (structured algorithmic task blending biology and computation)]. To develop her program, she built on the lectures to gain *"understanding [of] the topic of the assignment. So that had to do with nucleotides, knowing what an A G C T was relating to the DNA sequences, the human body"* [Sensor: Marker-based (lecture materials providing disciplinary grounding)]. This also included watching *"a short maybe 5-to-10-minute video clip."* She explained that she *"already had a bit of an advancement in understanding kind of the A G C T when learning about it in lecture"* due to her minor in biology, but she *"wouldn't say that it gave... any advantage when ... conducting the actual coding program because the coding program didn't really have much to do with biology"* [Sensor: Marker-based (distinction between domain content and programming task)]. She described the lecture discussions as receiving a set of puzzle pieces, along with end-goal instructions. She explained this in the excerpt below:

So, she gave us a few little puzzle pieces and then asked us to kind of build the puzzle together and building that puzzle was building the program. So, a couple of those puzzle pieces was... she didn't necessarily like here up on my screen say that, you know, you need to set them all equal to 0, but she might have discussed it more via words. And then

we had to be able to interpret and understand that ourselves. So, I had notes within lecture, and I would write a series of things. And she would say, OK, you're going to have, like, a string. And then she'd have an underline with a missing text. So, we didn't actually know what to write in that code. We had to be able to do that ourselves. But we knew what we wanted to happen, so having that kind of almost end goal, we had to have that: How do you get to that end goal? [Sensor: Marker-based (interpretive prompts guiding learner autonomy); Actuator: Sematectonic (self-driven reconstruction of logic based on implicit cues)]

Althea explained that this assignment “*wasn't as stressful...[as] the rest*” –not having many difficulties – because her “*professor did a good job with giving... bits and pieces of the assignment*” [Sensor: Marker-based (perception of instructional scaffolds through fragmented task cues)] and it was a natural progression for other MICA assignments [Sensor: Marker-based (interpretation of assignment sequence as scaffolded learning)]. She said, “*because I had a lot of prior knowledge from my other assignments, I was able to almost use my knowledge and skills that I've built through assignments one, two and three to... jump within assignment four*” [Agent Dynamics: Recursive application of prior learning experiences]. Her most “*challenging part... was creating the first one[section]... using FOR an IF/THEN statement*”. She knew after the first one worked, she “*would just do the exact same thing and just change the letters for G C & T*” [Actuator: Sematectonic (repetition and modification of working logic)]. She further that she was comfortably with this assignment because they (TA and professor) “*had built a lot of our knowledge and skills ... promoting that Agency of student driven learning,*” [Emergent System Behaviour: Growing independence as result of scaffolded instructional model].

She explained a few things that she believed is responsible for that: [Agent Dynamics: Reflection on conditions fostering independent programming identity]

1. Greater guidance at the beginning of the course – She said, *“What I mean by a lot more guidance...we were given some starter packages, ...maybe 5 to 6 lines of code where we just ... run that program, see what it gives us and then work towards getting to where our Professor/TA wanted us to get to.”* [Sensor: Marker-based (starter code cues interpreted as scaffolding strategy)]
2. Leading questions – She explained, *“When asking questions, we weren't even given answers. We were given kind of “What do you think? Or did you try it out or have you tested it?” Those types of questions ... get [me] to the point of jumping into assignment 4 a lot quicker than the rest of the assignments.”* [Sensor: Marker-based (interpretive prompting shaping agent decision-making)].
3. Experience modifying her own codes or redesigning her program to get desired results. [Actuator: Sematectonic (strategic code redesign informed by prior failures)]. Below she explained her actions in previous coding assignments when faced with challenges:

I realized “OK, this is not working. This is not doing what I'm asking it to do.” So, I needed to scratch it all, delete it all, and start with something fresh, and come up with a different idea because what I was doing before wasn't working. So that's what ended up happening because in Assignment four, I've had so much more experience with those other programs, I was almost able to kind of...I was almost just like kind of a pro at programming in that way where I was able to look at it and be like I know that these things didn't work. So, I'm not gonna even start with that. I'm gonna start with what I think is going to be right. If it's right, great. If it's not, I'll change it and modify it later.

Furthermore, Nancy explained that through these methods, she “*was able to actually understand...her code while reading it*” [Emergent System Behaviour: Development of self-monitoring and code reasoning skills]. She also developed many techniques to code. [Sensor: Sematectonic (environmental feedback guiding iterative debugging)] For example, “*comment things out to see...Is there any purpose? Does it change my program in any way?*” [Sensor: Sematectonic (use of code isolation to detect unintended behaviour)] [Actuator: Sematectonic (intentional removal to test conditional effect)] or put things in “*different cell and then ... work cell by cell with just a little bit of code to see... how... it was being responsive*” or responding to the specific location of a bug – a feature she noted as unique to Python Jupiter in comparison to the other software she encountered [Sensor: Sematectonic (environmental feedback guiding cell-by-cell inspection); Actuator: Sematectonic (experimental coding techniques for localized troubleshooting)]. She also used comments to tell herself what she wanted the computer to do as she noticed things and as notes to communicate to her professor/TA [Actuator: Marker-based (use of comments as internal and external signaling tool)]. She explained:

the comments is what I was telling myself What I want the computer to do. So, if I notice the number was too big or too large, that's not doing what I wanted to do. So, my comments were portraying that interpersonal domain whereby I was communicating to my computer, “OK, I need this computer to tell me how many A nucleotides are in this DNA sequence. This is what I want the computer to do. So now I look at my program, I compute, and I come up with these statements. After then I look back at it and I'm like, “is this what my computer's doing?” These are just questions I would personally ask myself in my head and if it does it great; If it doesn't, then I need to change something up again, so my comments were essentially also beneficial to my professor and TA [teaching

assistant] on grading. [Sensor: Sematectonic (reflective questioning informed by program output)] [Actuator: Marker-based (comments used as instructional and diagnostic scaffolds)].

Given the agency that Nancy developed, she “*felt enthusiastic*” to this assignment and wanted to do this assignment herself without “*seeking guidance or help from any professor or TA [teaching assistant] or peers in general*” [Agent Dynamics: Expression of self-directed confidence]. However, a big part of doing her assignment was “*having that cooperative learning with others.*” She used the computational concept “*based off of what was in...[her] strengths*” [Agent Dynamics: Strategic alignment of programming choices with personal strengths] and suggestions from the class lecture [Sensor: Marker-based (interpretation of instructional cues to guide coding decisions)], but then “*realized like a lot of them [her classmates] did something different.*” In many cases, when “*comparing their final results*” their results individual changed their program. She explained:

I went through comparing their final results. If they were off by a number or two, or if something was going on, then we realized OK, we have a problem. “What? What is wrong with it?” For example, if the count number of T was wrong, maybe I forgot to add a certain number to it, right? Because it's still outputting a value, so it had to do a lot with debugging and understanding, “OK, maybe something is wrong” ... so we never knew whose was right. So, we had to both interpret both of ours because we didn't have an answer key, right? It was understanding both of our codes and seeing, OK, what's the problem if it wasn't debugging in that way. It was debugging through my process of programming because sometimes my line I would miss an indent. [Sensor: Sematectonic (recognition of unexpected program output prompting investigation)] [Actuator: Sematectonic (collaborative debugging through comparison and code inspection)]

[Emergent System Behaviour: Interpretation of correctness through peer discourse and iteration]

She explained that her values only vary in one instance, but she attributed it to the nature of Python.

There was only one instance where one number was off but that was, I think just when you run Python over and over and over again, sometimes it modifies the number by a number 2 but aside from that there was nothing I personally have to change but some of my peers did have to change theirs because they found as though their program was wrong. [Environment: Dynamics (unpredictable behaviour due to system-level variability)] [Agent Dynamics: Trust in program design and situational debugging judgment]

Generally, for debugging for this program she just had to bug for syntax errors. In past assignments, she had to seek help from a professor, peer, and teaching assistant when she was “constantly getting an error and error an error” [Agent Dynamics: Progression from guided troubleshooting to independent debugging].

Nancy also explained that the professor’s hints, and analyzing the instruction in the assignment, helped her to organize her program into sections and to choose the selected computational concepts. For example, in analyzing she said “So, reading this assignment first, I was like, “OK, we’re working with four different variables. So, I must have four different if statements...So, I have four over here because of the four different nucleotides.” She later said that her professor told her that “You’re going to have four different IF statements,” but she never told them what the “IF statements are gonna look like.” So, “that’s how...[she] kind of knew that that’s the right way that... [she] was going” Sensor: Marker-based (instructional suggestion

interpreted to guide algorithm design)] [Actuator: Sematectonic (self-generation of conditional logic structure)].

Nancy pointed out that she did not need a lot of math to develop this program as she did with other assignments, *“for example, ... the Bertrand paradox”* where she *“had to use a lot of trigonometry.”* She explained that the math came in when she had to use her program outputs to answer guided questions, such as, *“compare the proportions, conduct a statistical test.... What do you notice?... to be able to correspond the coding aspect to the math aspect”* [Emergent System Behaviour: Integration of code-based output with mathematical reasoning]. From doing this assignment she understood that *“the whole point of actually doing this with Python Jupiter was... to kind of show us that it actually is a lot more efficient to code something than it is to manually do it by hand, because there's just somethings in life you can't do by hand”*[Emergent System Behaviour: Realization of computational efficiency for real-world problem solving].

In addition, based on the practical nature of this assignment, and Nancy’s disposition of always wanting *“to see what the purpose of that assignment is to the real-world,”* Nancy realized, *“WOW programming actually has a purpose to real-world applications that you know, like hospitals can use it, doctors can use anything along those lines”* [Emergent System Behaviour: Recognition of programming’s real-world relevance].

Furthermore, Nancy emphasized that she was prepared well in class for the mathematical aspect of things. She explained:

So, we were given, you know, the probability - these are the types of math - where in lecture we were given questions, and we were able to solve them mathematically. So now with this program we were given the question, but we had to solve the mathematically based off of a program we created ourselves [Sensor: Marker-based (mathematical preparation informing computational interpretation)].

Having been prepared for the mathematics, Nancy was able to make inferences about the randomness of the nucleotides. For example, she said:

I noticed that there's a significant difference between the conditional probability and the proportion of the GC. So that's why I said portraying that the nucleotides are not independent in random. So, the only reason I could come to this concluding statement was through my understanding of what the numbers were [Emergent System Behaviour: Inference from statistical outputs informed by mathematical insight].

Nancy “*finished it quite early*” and spent a lot of time trying to make her program “*more efficient with my... more user friendly*” [Actuator: Sematectonic (optimization of output and structure for clarity and responsiveness)] [Agent Dynamics: Self-motivated refinement to improve quality]. She felt that by doing this assignment, the computational practices and concepts learned throughout the course was “*grandly improved upon...more so than any other assignment for the reason that...she was able to get that practice and problem solve a problem; she said, “solve myself*” [Agent Dynamics: Self-efficacy through authentic problem solving].

Through her engagement, Nancy concluded that programming is important for students to learn because it helps them see the connection between formulas in the textbook and technology. She emphasized that programming supports students in “*build[ing] those skills of that interpersonal/intrapersonal domain,*” and that it “*promoted that agency, audience, remix and reuse — all of those types of concepts*” [Emergent System Behaviour: Cross-domain reflection on programming’s cognitive and pedagogical value]. Having been nervous about programming at the start and seeing no purpose of programming in mathematics, Nancy left believing that programming was “*actually really important*” and could “*allow students to grow as a learner and understand mathematics*” [Emergent System Behaviour: Evolved disciplinary perspective

through iterative programming practice]. Thus, it is something, as a future teacher, she would want her students to learn. She even started to envision units in the curriculum that she could integrate programming into, and the pedagogy to teach it [Emergent System Behaviour: Vision for pedagogical integration]. On a scale of 0 to 10, she rated her proficiency in using programming for mathematics learning as 8 given that she feels that she *“still have not been exposed to all the programming software there are...[and] there's still so much that [she] need to grasp.”* However, she chose 8 out of 10 because she feels like she still had a *“strong understanding of how to use the programming software, how a program, how to use certain commands and everything”* [Emergent System Behaviour: Perceived fluency emerging from iterative engagement].

Gina’s Story. Gina [Agent] is a 4th-year mathematics concurrent intermediate/senior education student with teachable in Mathematics and French. She did both MICA I and II before doing MICA III. Gina's story was developed based on data collected for her third assignment for MICA III using Python programming language. The assignment required the class to create a program to read the DNA sequence, clean it, count the number of nucleotides, and then determine a specific proportion of nucleotides based on outputs from the program. By cleaning the DNA sequence, they had to get rid of the spaces so that they could be able to count the nucleotides. For example, the nucleotides were A, G, C, and T; so, they had to remove the spaces between the sequence and count the number of A, G, C, and T. [Environment: Dynamics (structured algorithmic task blending biology and computation)].

Gina got started developing her program by focusing on the main objective: cleaning the DNA sequence so it could be read accurately. She began the design by having the program read the sequence and filter out anything that wasn’t a nucleotide, ensuring it only retained G, C, T,

and A. [Actuator: Sematectonic (algorithmic filtering to isolate nucleotide data)] From there, she structured the next part of the program to count how many times each of those nucleotides appeared in the sequence. She stated, *“The design was basically from it reading the sequence and picking out that it would only keep G, C, T & A. That was the initial part of it, like that's how I got started. And they were just counting the number of times it read G or C or T or A.”*

[Actuator: Sematectonic (counting routines structured through conditionals)]

Gina began working on her program after an initial discussion with her instructor and other students in class, despite having received the assignment beforehand. [Sensor: Marker-based (instructional prompts triggering design planning)] The conversation with her instructor led her to use an IF/THEN/ELSE statement nested within a loop. She explained, *“When she had given the assignment, she had, like us brainstorm, as a class, how we would do it. And then I just use the idea that in my project”* [Actuator: Marker-based (integration of peer and instructor ideas into structure)].

Gina had little difficulty completing her EO. Guided by the assignment script, she sequenced her assignment in different sections [Sensor: Marker-based (structured prompts guiding design logic)]. She explained the following: I just followed the order listed like the questions on the assignment, so I just went from like, it had listed, *“this is like you need to do first and this is what you're going to do”*. I just made my program follow the same organization in terms of this is what it's going to do first and then it's going to do these things because it was based off the order of the questions [Actuator: Marker-based (task-structured program sequencing)].

The section was based on the five different species she was asked to clean. She noted that the code to read the first DNA sequence was similar to the other four sequences; therefore, she

copied and adjusted the first section in separate sections to accommodate the other sequences. [Actuator: Sematectonic (reuse and modification of functional code for scalability)]. She explained that she did her program in sections because she found it easier to recognize “*what parts of the code was related to what species*” [Sensor: Sematectonic (breaking up code to make it easier to follow)]. She found the program to be “*very nice*” because she got more familiar with the programming language, and she had no challenges getting it to run. She attributed her comfort with the programming environment to her prior experience using it—particularly her practice with troubleshooting and debugging in earlier assignments, especially in MICA 2—which reflects her history of structural coupling with programming [Agent Dynamics: Comfort and fluency emerging from sustained practice].

Gina explained that “*while every assignment [in the course was] very isolated from one another,*” each allowed her to gain a better understanding of programming. [Emergent System Behaviour: Accumulated fluency across structurally distinct tasks] Gina also attributed her comfort with programming to the experience she gained in the last two MICA courses. The main challenges she faced earlier in her programming journey were “*very technical.*” Reflecting on her second MICA course, she shared, “*I would do the code, there would be nothing wrong with the code, but it just wouldn’t run.*” The problems weren’t with her logic, but with the hardware: “*Like hardware wise opposed to code wise... a lot of hardware issues as opposed to like coding issues.*” [Sensor: Sematectonic (detection of unexpected failure due to environmental affordances)] Support from instructors played a key role in helping her through, and she also highlighted the accessibility of programming help online, saying, “*Coding is something you can find online. You can always find a piece like something that would help you out online.*”

[Actuator: Marker-based (leveraging external code resources for resolution)]. She provided a brief summary of her learning across those courses:

I would say it's more because of the practices we had done over the past two years.

We've done like a lot of practicing in terms of structured assignment as well as with final projects which is more like you code something by yourself and get it to work type of thing. So, it was more of that nature where I've done a lot of coding on my own because of those final projects, so, therefore, it was kind of easy in terms of knowing where I was going because I've done this before where I have to do something by myself, whereas this is just a structured assignment, so it was kind of simple in doing so [Agent Dynamics: Confidence developed through autonomous coding experiences].

Gina noted that for this assignment, she did not have any problems with the random-walk graph she had to create, even though she had faced challenges with graphing in previous assignments. She reflected on the possible reasons for this, suggesting that it might have been due to increased familiarity with the programming environment, or perhaps because the task involved generating a random graph rather than working with data. [Sensor: Sematectonic (reduction of complexity through randomness vs. data mapping)] Gina explained:

It was normally when it came to graphing, there was a lot of bugs in terms of, like in previous, whereas in this time when graphing it was kind of straightforward because like I said, like I wonder like maybe it is the familiarity, or it was just because it was more of a random graph. It wasn't a graph based off the data. It was just a random one. So, it was much easier to do whereas in previous assignments like it was more of graphs based off the data. So, sometimes, there were bugs when the graph was reading the data [Emergent System Behaviour: Increased confidence with graphing due to reduced data complexity].

Gina used her completed program to generate data for calculating the proportion of specific nucleotides using conditional probability, a concept she was already familiar with. [Sensor: Marker-based (application of prior mathematical concepts in computational context)] [Actuator: Sematectonic (implementing known math logic into program structure)] She noted that she didn't have to do much debugging and found her peers especially helpful in verifying the accuracy of her program's output, since everyone was working from the same dataset. She explained, *"It would mostly be because like confirming numbers, because we all had the exact same files, so we knew like if majority got this number, majority like most likely it is right."* [Sensor: Sematectonic (anticipation of verification needed to ensure accuracy)] [Actuator: Sematectonic (validation through peer comparison of identical data sets)]. Her results matched those of the majority, so she did not need to make any changes to her program [Sensor: Sematectonic (perception of accuracy based on peer-aligned output)].

Gina didn't feel the assignment taught her anything new because she had already developed fluency in key programming concepts like loops and conditionals, and those skills helped her complete the task with minimal effort [Emergent System Behaviour: Fluency resulting from repeated task engagement]. While she explained that this program did not stretch her abilities, she believed that it affirmed to her just how far she had come [Emergent System Behaviour: Recognition of growth through effortless completion]. Having been indifferent about programming before beginning the MICA courses, she completed this assignment with the perception that programming is useful as a *"nice reinforcer in terms of mathematics practice of going back, checking your work"* and for doing things that are difficult to be done by hand, like reading a long sequence of DNA [Emergent System Behaviour: Recognition of programming as a mathematical tool for automation and verification]. She rated herself a seven for proficiency in

using programming for mathematics learning because even though she believed she had a good knowledge of the different programming languages like Python and Scratch, she needed to gain experience with catering to students for mathematical learning [Emergent System Behaviour: Recognition of current fluency and need for pedagogical extension].

Result of Thematic Analysis

The purpose of this study is to explore the nature of interactions between learners and their environment as they interact with a programming environment for mathematics learning. Specifically, the study investigated the co-actions emerging from the interactions of students enrolled in MICA courses in which they design and implement interactional environments for mathematics investigations in an Ontario university. The research question guiding this study was focused on exploring the nature of interactions between learners and their programming environment as they do mathematics in a programming environment. Given the exploratory nature of the questions, and informed by the phenomenology research method, data in this study were collected qualitatively, using three different methods: semi-structured stimulated recall interviews with EOs, field notes, and student reflections. The data were analyzed using thematic analysis, following Braun and Clarke's (2006, 2020) six-phase framework. The process began with familiarization, involving repeated listening to interviews and annotating transcripts using a colour-coded method inspired by Towers and Martin (2015) to capture lived stigmergic experience and context. Initial codes were developed both inductively from the data and deductively using established theoretical frameworks. These codes were iteratively refined and clustered into candidate themes. Themes were reviewed, defined, and named to ensure they were meaningful, distinct, and relevant to the research questions. The analysis was recursive, involving ongoing reflection, bracketing of assumptions, and revisiting of data to ensure the

themes authentically represented participants' experiences. The following sections discuss the six themes that emerged from analyses of the three sources. Altogether, fourteen sub-themes are used across the themes to provide depth and further detail. Each theme is introduced with illustrative data extracts, offering a coherent narrative that conveys participants' engagement with mathematical concepts in a programming context.

The six themes identified in the analysis were: Pedagogical Traces, Environmental Semantics and Social Traces, Learning Strategies and Student Agency, Dynamic Problem-Solving, Perspective and Empowerment, and Development of Proficiency. Table 5 outlines each theme and its corresponding subthemes, highlighting the layered structure of the analysis. The themes are further organized into three categories based on their nature: (a) Perturbations: Trace-Driven Learning; (b) Agents' Actions and Perceptions; and (c) Emergent System Behaviour: Proficiency, Perspectives, and Environment Shaping. These categories structure the organization of this section.

Table 5

Emerging Themes and Subthemes From Co-Action

Theme	Subthemes	Sub-subthemes
Perturbations: Trace-Driven Learning		
1. Pedagogical Traces	<ul style="list-style-type: none"> • Guidance and Instruction from Task and Direct Instruction • Other Teaching Strategies and Techniques 	
2. Environmental Semantics and Social Traces	<ul style="list-style-type: none"> • Environment State and Changing Semantics • Ad-hoc Traces from Other Agents 	

Theme	Subthemes	Sub-subthemes
Agents' Actions and Perception		
3. Learning Strategies and Student Agency		
4. Dynamic Problem-Solving	<ul style="list-style-type: none"> • Adaptive Problem-Solving • Collaborative Dynamic Problem-Solving • Problem-Solving Using Computational Tools • Problem-Solving Using Mathematical Tools • Integrating Mathematical and Computational Thinking in Problem-Solving 	<ul style="list-style-type: none"> - Foundational Computational Concepts - Iterative Development Practices - Mathematical Operations, Processes, and Dynamics
Emergent System Behaviour: Proficiency, Perspectives, Empowerment, and Environment Shaping		
5. Computational Perspective	<ul style="list-style-type: none"> • Pluralistic and Creative Approaches • Programming for Mathematics Understanding • Learning Disposition and Mindset 	-
6. Development of Proficiency in Programming for Mathematics Learning	<ul style="list-style-type: none"> • Agent Dynamics and Experience • Subjective Relationship and Comfort 	

Perturbations: Trace-Driven Learning

The first two themes, Pedagogical Traces and Environmental Semantics and Social Traces, share a common focus on the environment perturbations that prompted participants' engagement with their environment. These signs initiated and continuously prompted structural coupling between the learner and the environment, setting the stage for participants to create a world of personal significance.

Pedagogical Traces

It was evident that participants perceived a range of signs or traces from their environment, including assignment guidelines, structured descriptions, classroom discussions, and instructional prompts, which perturbed their thinking and supported the development of their EOs. These perturbations are described as pedagogical because participants attributed them to the actions or influences of professors, instructors, and teaching assistants. Although not directly observed from the instructors' perspectives, participants interpreted these prompts as strategic, offering specific opportunities for interaction with the task. The overall instructional approach, as understood by participants, reflected inquiry-based learning (IBL) and was supported by project-based formats. These perceived strategies were categorized into two broad themes: Guidance and Instruction from Tasks and Direct Instruction, and Other Teaching Strategies and Techniques. Both contributed to participants' agentic engagement and exploration within the programming environment. The signs or traces were perceived as basic level perturbations, referring to cultural and contextual elements that invited students into shared, situated practices of mathematical modeling, exploration, and interpretation.

Guidance and Instruction From Tasks and Direct Instruction. Throughout the analysis, it was evident that participants were initially perturbed into inquiry of their EO through tasks that, from their perspective, were supported by directed instruction in lectures. The instruction and guidance they perceived, whether presented in structured written form or delivered orally, were interpreted as significant signs that prompted general action and inquiry, such as guidance encouraging exploration. These perturbations influenced how participants selected and framed their topics, segmented and developed tasks, and pursued mathematical and computational ideas.

For Amanda, Gina, and Nancy, who completed the more structured DNA task, the assignment itself was seen as instrumental in shaping their EO development. The following excerpts illustrate how participants described their perception of the structure and expectations of the written assignment task, suggesting that pedagogical strategies functioned as indirect perturbations that guided their computational and mathematical inquiry.

Amanda: So, for this project, we were told to create a program that would read the code, read the DNA sequence and then it would display DNA walk and would also read the proportions of certain parts of the DNA genomes and then randomly generate a walk. So, we were given what the goal was of the assignment.

Gina: I just followed the order listed - like the questions on the assignment; so, I just went from it had listed...I just made my program follow the same organization in terms of this is what it's going to do first and then it's going to do these things because it was based off the order of the questions.

Nancy: So, after creating...that human DNA, it's asking here in the next couple of questions, "compare the proportions, conduct a statistical test. What do you notice? Does this occur more or less?" Those were the types of questions that actually were like, "OK, now I need to understand the math that's behind what I just coded."

From the above, it may be seen that participants interpreted structured elements of the assignment as meaningful signs that shaped their actions and thinking. These perturbations helped clarify objectives, sequence tasks, and prompt deeper engagement with mathematical concepts. Amanda described being guided by the assignment's stated goals, Gina used the order of the task as a framework to organize her program, and Nancy was drawn into exploring the

mathematical reasoning behind her code through specific questions embedded in the task. These perceived prompts supported systematic progression and encouraged critical engagement during co-action.

In contrast, when assignments were more open-ended, participants developed their own task outlines and goals to guide their process. In doing so, they drew on available class resources to support their development and exploration. For example, participant Luca described how materials shared during class helped him formulate the direction of his task.

Interviewer: How did you get the idea of your project?

Luca: So, for our final project for math 1P40, ... the professor provided some resources, kind of like a guideline of what you wanted your final project to be about. It was open-ended. ... You know we had already done an assignment on a different kind of cryptography called RSA encryption, and I found that a lot of fun, so I decided I'd read another encryption program. So, like I chose to do a hill cipher encryption program.

Interviewer: How did you get started designing your project?

Luca: I'd have an example already written out, so I knew kind of the steps of how it should work like on paper, and then what, like the solution should be like what the message should encrypt to and what it should decrypt to and then I kind of tried to do it with that given example in the program. And then from there like you can kind of extend that to any other code you want to do.

Luca's response shows that he drew on class resources to guide and inspire his project. He referred to the guidelines provided by the professor and connected his idea to a previous assignment on RSA encryption that he found enjoyable. Building on that experience, he chose to

explore a related topic, hill cipher encryption, for his final project and developed and outline to guide him. This demonstrates how he used available class materials as a foundation for further mathematical and computational investigation.

In addition, all six participants noted that they drew on prior lectures as sources of guidance in developing their EOs. Participants perceived signs from lecture, such as hints, shared ideas, or provided materials like starter packages, as useful affordances that created conditions for engagement. These conditions shaped the direction of their projects by opening up particular ways of thinking, suggesting productive entry points, or structuring problem solving. For instance, Ryan described how a particular lecture prompted him to collaborate with a classmate and formulate a plan on paper that guided the development of their EO, suggesting that the ideas presented in class influenced how he envisioned and initiated his project.

So, it was in lecture that the von Neumann Middle Square method was mentioned, and I was sitting with my classmate. ... And we were asked to find a suitable project to work on. I decided I was going to team up with her to do it. ... We decided to ... base [it] on assignment three, yeah. ... So, the purpose of this project is to analyze the von Neumann Middle Square method for randomness, or pseudo randomness, or non randomness. ... So, we have to figure out how to get this. And we walked through the idea - What is the middle square method? So, you take a number, and you want to square it, and you get this other number, and then you want to take its middle. But the problem here is, how do you get the middle? How do you select the middle using like a machine? How do you formalize that process into something that a machine can do or understand? ... We have to reason it out on paper.

Below, Nancy and Amanda, who did the DNA sequence, explained their perception of how the lecture prompted inquiry:

Nancy: So, to begin with this specific assignment 4, we were given some lectures in math 3P41 from the beginning. The lectures were (kind of) to get us to understand the topic of the assignment... throughout our lectures, our professor also gave us some hints, so I kind of like to use the analogy of a puzzle piece. So, she gave us a few little puzzle pieces and then asked us to kind of build the puzzle together, and building that puzzle was building the program. So, a couple of those puzzle pieces was... she didn't necessarily like -here up on my screen- say that you know, you need to set them all equal to 0, but she might have discussed it more via words. And then we had to be able to interpret and understand that ourselves. So, I had notes within lecture, and I would write a series of things. And she would say, OK, you're going to have, like, a string. And then she'd have an underline with a missing text. So, we didn't actually know what to write in that code. We had to be able to do that ourselves. But we knew what we wanted to happen, so having that kind of almost end goal, we had to have that: How do you get to that end goal? So that's kind of where she gave us that assistance.

Amanda: So, we actually started designing it a bit in the class. We, the teacher, just asked us, how do you think you would do these things? And then we discussed with classmates how we think we would do it: write the code algorithmically...the initial design of it I would say was like a class part, but it was very vague... it was definitely helpful with that class discussion. And then I actually forgot we were given this part of the code right here, just this part because we had never learned how to type in like how'd you

get the code to read a file and call up a file. So, we learned that part, but then I personally had to research online how to have it read the code. Uh, like this: what the Syntax would be.

Overall, participants' data demonstrated that such lecture-based affordances acted as initial perturbations that supported exploration and ignited curiosity. For instance, Ryan noted that the lecture on the von Neumann Middle Square method, paired with the project outline, prompted him to collaborate with a classmate and adapt a prior assignment into their EO. This sparked conceptual planning, allowing them to decompose the idea into executable steps and build a written plan that mapped onto their code. Similarly, Nancy described lecture inputs as "puzzle pieces" that she had to interpret and piece together through critical thinking and annotation. Amanda emphasized that the teacher's guided design process, including partial code for new concepts like file reading, offered a structured but open foundation that she extended through independent research. Collectively, these examples underscore how lecture-based elements acted as perturbing conditions that scaffolded students' inquiry and the development of their EOs.

Other Teaching Strategies and Techniques. The analysis of participants' accounts revealed that certain signs, attributed to the actions of professors, instructors, and teaching assistants, created significant moments of interaction that shaped students' learning and inquiry. While some of these signs were encountered during lectures or through direct instruction, many extended beyond formal teaching settings. In this study, these signs are understood as distinct instructional strategies and techniques, such as brainstorming, formative assessment, reflective practice, and guided discovery, that left traces within the learning environment. Whether introduced during a lecture or embedded in a task, these traces played a role in conditioning participants' ongoing co-action with the programming and mathematical environment. Table 6

presents excerpts that illustrate how participants responded to and acted on these pedagogically meaningful influences during their programming work.

Table 6

Excerpts From Interview Transcript Showcasing Teaching Strategies and Techniques

Teaching and Learning Strategies and Techniques	Example quote
Brainstorming	Gina: My instructor. We had brainstormed like before, like when she had given the assignment. When she had given the assignment, she had, like brainstormed, as a class, how we would do it, and I just use the idea in my project
Formative Assessment, reflective practice	Amanda: ...it actually my string for when the proportion of GC, but that was incorrect, and I hadn't realized it at the time when I had thought my whole code was running correctly. But then after when we were doing the write up of the report and answering questions and doing the mathematical aspect of it and using my code to uh do hypothesis testing..., that's when I was reevaluating my math here that I used and so it wasn't that the code was wrong. Like it was doing what I had said to do correctly. But the math was incorrect that I was using.
Exploratory learning; scaffolding	Althea: So, it was actually up to me to. This year he [instructor] would show us things, but he wouldn't tell us, "Oh, this is what this does" ...if you asked him, he would tell you. But like, in that moment, if you're not asking, he's not telling you, so it's like he's showing you everything and you can see what it does, but individually... I like to focus on everything I'm doing. What's it? What's the purpose, you know? So, that's definitely something I struggled with like going in and finding how everything works.
Exploratory learning; scaffolding	Amanda: Yeah, I feel like leading questions. When the professor or TA [teaching assistant] are trying to help me through. Leading questions, as opposed to just giving the answer and pointing you in the right direction where to explore your error was also beneficial.
	Luca: The professor provided some resources, kind of like a guideline of what you wanted your final project to be about. It was pretty open-ended. You know we had already done an

Teaching and Learning Strategies and Techniques	Example quote
Guided Discovery, scaffolding	<p>assignment on a different kind of cryptography called RSA encryption, and I found that a lot of fun, so I decided I'd do another encryption program...I'd have an example already written out, so I knew kind of the steps of how it should work like on paper, and then what, like the solution should be like what the message should encrypt to and what it should decrypt to and then I kind of tried to do it with that given example in the program</p> <p>Nancy: I found that with assignments one and two, we were given a lot more guidance than we were with the rest of our assignments moving forth. What I mean by a lot more guidance was that our Professor and TA [teaching assistant] actually gave us some beginning starting code. So, we were given some starter packages, and it would be like maybe 5 to 6 lines of code where we just copy and paste it into Python Jupiter and we're able to run that program, see what it gives us, and then work towards getting to where our Professor/TA[teaching assistant] wanted us to get to. So, we were almost given already, like quite a bit of the puzzle to complete, but not all of it. With this one I found like we were only given a couple scattered like puzzle pieces, and then we were able to interpret it</p>
Cooperative learning	<p>Ryan: So, it was in lecture that the von Neumann Middle Square method was mentioned, and I was sitting with my classmate Julie. And we were asked to find a project - a suitable project to work on. I decided I was going to team up with her to do it... I focused a bit more on the programming aspect and she focused more on writing up the report</p>

The excerpts from Table 6 demonstrate how participants interpreted teaching strategies and techniques as conditions that guided their thinking and actions within the programming and mathematics learning environment. Gina's experience illustrates how collaborative brainstorming and active engagement with the assignment prompted deeper thinking and problem-solving. By participating in class brainstorming sessions, she was able to generate and refine ideas, which directly influenced her approach to the project. Amanda's process shows how

formative assessment and reflective practice contributed to critical thinking. Realizing that her mathematical calculations were incorrect, she reassessed her method, demonstrating how ongoing evaluation guided her to refine her understanding.

Althea's account highlights how exploratory learning and instructional scaffolding supported independent inquiry. Rather than being given direct answers, she was asked leading questions, which encouraged her to investigate the purpose of each element in her work. This prompted her to explore the system more thoroughly. Luca's experience with project-based learning and differentiated instruction demonstrates how these strategies foster adaptability and creativity. The open-ended nature of the task allowed him to select and define his own direction, shaping both the structure and content of his EO.

Nancy's experience illustrates how guided discovery and scaffolded instruction supported incremental development. By receiving partial code and being encouraged to build upon it, she was able to piece together the broader task over time. This process promoted deeper understanding and strengthened her problem-solving abilities. Ryan's account underscores the value of cooperative learning, showing how peer collaboration facilitated joint problem-solving and shared decision-making. Teaming up with a classmate enabled him to combine different insights and strategies, enriching his overall learning experience and contributing to a more effective project outcome.

Environmental Semantics and Social Traces

Beyond perceived instructional guidance, participants also responded to environmental signs related to the evolving state of their programs and the changing semantics of their actions, specifically, shifts in input-output relationships triggered by mathematical or computational modifications. These semantic changes were interpreted as conditions that invited code refinement or redirection. Additionally, participants picked up traces from other agents in the

environment, including peers, teaching assistants, and instructors. These interactions functioned as ad-hoc prompts, influencing participants' decision-making and shaping the development of their EOs.

Environment State and Changing Semantics. Beyond perceived instructional guidance, participants also responded to environmental signs related to the evolving state of their programs and the changing semantics of their actions, specifically, shifts in input-output relationships triggered by mathematical or computational modifications. These semantic changes were interpreted as conditions that invited code refinement or redirection.

For instance, participant Nancy explained how the program “spoke” to her, prompting her into action:

...in general, when I program, I try to do it line by line and then the program tells me if it's working or if it's not working... And also, sometimes with Python Jupyter, it's pretty good when it comes to bugging because it'll actually sometimes tell you, “Look at line 52, space error or indent error or capitalization error,” so it'll kind of tell me specifically within it.

Changes in code lines affect the overall program semantics, which Nancy interpreted and made decisions about debugging. Similarly, participant Ryan described how observing the program run informed his understanding, guiding the changes he made to improve its function:

...we actually originally used a WHILE loop but when I went back and optimized the program to make it run faster because we were running like a million cases trying to get all the all these numbers printed and it was taking a long time. So, I said why don't we figure out which is the method that runs the fastest and it just so happens because we knew that we were gonna have a beginning and an end that a certain spot. That's the

reason we chose FOR. FOR is the fastest one that works if you know the beginning and end.

Ryan's actions were guided by the need to optimize the program, a need triggered by sensory feedback from observing the program's behaviour. When he noticed that the program was running too slowly, he made practical adjustments to improve its efficiency. Specifically, he observed that the WHILE loop he initially used caused delays when processing a large number of cases. This prompted him to reconsider the structure of his loop and switch to a FOR loop, which he found more efficient when the starting and ending points were clearly defined.

Likewise, Althea described how feedback from her program supported her analysis while running multiple simulations to examine variability and recurring patterns in a stochastic model of greenhouse gas emissions. She explored how two different emission rates affected atmospheric conditions over time. By observing the graphical outputs, she selected the plots that best represented her overall findings. Her sensory engagement was shaped by the evolving semantics of the program, specifically, the relationship between inputs (e.g., number of simulations) and outputs (e.g., graphs). These environmental shifts prompted her to further analyze her results and present those that most clearly captured the dynamics of the stochastic process she was modeling.

You know, just this was me now running the individual simulations for, you know, I did five. I did 10, I did 20, and I did 50 just so we could get more variants in my calculations. And then I noticed when I was doing it, there was like, these - even like these were plots that I did. But then whenever I reran them because the probabilities kept changing, you know, then because again, this is very random situations, is not always going to be this

way. These plots did change, but I chose these at the end of the day because they were more, like there were more forthcoming, they were more common in my results.

Likewise, Luca's sensory engagement was shaped by the evolving semantics of the program. While developing a sub-procedure to generate matrices for encrypting and decrypting text, he encountered an issue where the program consistently decrypted inputs into a string of 'Z's—an indication that something had gone wrong. Through interaction with the programming environment, including Visual Basic's error messages and his own deductive reasoning, Luca traced the issue to how negative numbers were being handled in modulo 27 operations. Recognizing that the incorrect output stemmed from the matrix generation step, he resolved the problem by implementing a modified modulo function that converted negative results into positive values, thereby resolving the issue.

This sub procedure was a big challenge, and it didn't work for a while. And then like you enter a letter, or a word and you encrypt it, and it would decrypt just about whole bunch of Zs. Like every word, and that was pretty aggravating for a while. ... You need the matrices in order to do it, to do the encryption and the decryption... everything has to be done mod 27. But you can't have negative numbers, so every negative number has to be converted to a positive number mod 27... When you run a program in Visual Basic and there's an error, it will tell you where the error is. It won't always tell you what the error is. ... If there's a problem with it decrypting, then there must be a problem with the matrices... I kind of try different things and be like, 'OK, well, does this work for some values but not others?' And that was how I discovered it only didn't work with negative values.

In general, participants' sensory experiences were significantly shaped by the dynamic nature of the programming environment and the evolving semantics of their code. Nancy emphasized the value of incremental coding and debugging, noting that she relied on feedback from the program—whether it was functioning or not—to guide her next steps. Ryan described how observing the program's performance revealed inefficiencies, prompting him to replace a slow-running WHILE loop with a more efficient FOR loop. Althea used her completed program to run multiple simulations of a stochastic process, closely analyzing graphical outputs to identify representative patterns in the data. Luca encountered issues with matrix generation for text encryption and decryption, initially receiving uniform "Z" outputs. Drawing on error messages and deductive reasoning, he identified the problem as stemming from the treatment of negative numbers in modulo operations and implemented a revised function to correct it. Across all four cases, participants demonstrated how sensory input from co-action with the programming environment, such as runtime behaviour, visual output, or error feedback, prompted targeted adjustments that advanced their inquiry and problem-solving.

Ad-hoc Traces From Other Agents. Participants also responded to emergent traces left by other agents in the programming environment, including professors, teaching assistants, and peers. These traces, such as expressed ideas, modeled practices, or modifications to the environment, acted as perturbations that shaped participants' thinking and action. Although some of these traces originated from educators, they are distinct from pedagogical traces, which are perceived to be more structured or intentionally designed strategies such as guided discovery or formative assessment. In contrast, ad-hoc traces from other agents refer to spontaneous, informal, or unintended cues or influences left by others (agents) in the environment that shape a learner's thinking or behaviour. This was evident across all participants. For example, the following

excerpt illustrates how Althea's thinking was perturbed by instructional traces she encountered through her instructor's engagement in the environment.

Althea: I had actually gotten to the end when I had done the expected value...of 780... So, I was like "ohh how many months will it take for Canada to reach 780" and then at that point in my head I was done. I was like, "OK, this is the end of my assignment." But then I asked myself [to] wait, this is something we have discussed, and I want to actually see this happen...like "how long?" So even if it takes about 11 years and 1 month.

Interviewer: You said you finish your program and then that came to you. So why do you think that came to you? Do you think it was because of something in the programming environment or what caused you to start thinking that way?

Althea: I found the teacher to be interesting because he thinks a lot when it come to these types of programs. There were so many things that I would say weren't in the course program or I would say in the course content but he wanted us [the class] to think so I was thinking....let me think from the point of view of... [Jones (pseudonym)], that was his name, "how can I explore my project more" cause at this point I was like I think I'm done, but let me see if I can consider something he has asked before. "Like how I can actually measure, you know, this kind of situation?"

Althea initially believed she had completed her assignment after calculating the expected value of 780 and estimating the number of months it would take for Canada to reach that threshold.

However, as she reflected on earlier classroom interactions, she became responsive to new possibilities for exploration. Drawing from shared experiences during prior discussions, she re-

engaged with the task, questioning not only how long it might take but also the probability of it happening within a specific timeframe. When asked what led to this shift, Althea pointed to the instructor's orientation—one that opened space for going beyond what was formally required. She described viewing her project through the teacher's perspective as a way of re-entering the problem and extending her inquiry.

Participants also responded to traces left by peer agents. Althea, for instance, described how a friend's approach to programming, such as disassembling the code into manageable parts and experimenting through repeated trials, occasioned a reorganization of her own workflow. This collaborative presence, even when indirect, shaped how she navigated and refined her process through a shared field of engagement:

... I had a friend in the course who like I whenever we were studying together... I would see the way his mind is working and he's just like saying, like, "oh, I'm going to see how this works." And I said I could learn a thing or two from him. And then I started doing that in my assignments. I would take all the bits and pieces that teacher had given us to work with. ... Like just take little bits and pieces and actually understand how Python works, which is why I have the background works. And then once I understand the function of something, then I'll take the actual part of the code I need and also put it there and then see how it works and then bring it back change the values if I need to, you know, things like that. So that was definitely how I played around with the probabilities to decide on 0.005 percent and 0.3 percent.

Similarly, Amanda revised the computational concept she was working with after encountering traces of alternative strategies among her peers. During classroom discussions, she initially intended to use a FOR loop to remove spaces and numbers from the file. However, she realized she lacked a method to effectively delete the spaces. While working in the lab

environment, Amanda recognized that a different approach, rewriting the file and constructing a new string by adding only the desired characters, would be more manageable. This shift in strategy emerged from observing how others around her were engaging with the same problem.

In Amanda's words:

...it was more when we're in the classroom discussion that my original plan and idea was to have it erased. And then when we got to the lab the next day and started the actual project, I realized ohh, it would be easier doing it the other way that other people were thinking on doing it

In other cases, like with Luca and Ryan, participants worked in peers, bouncing idea off each other to develop their EO. In the transcript below, Ryan explained how he worked closely with his one friend on their project

Interviewer: OK, beautiful. You mentioned that works with your peer, but did you seek help from any other, from your instructor or from any other of your peers?

Ryan: No, mostly it was pencil and paper, talking with my one friend who I collaborated with on the project and just thinking about it, thinking about it a lot, almost to the point of obsession, I suppose [laugh].

Ryan also noted that he spent considerable time reflecting on the project, suggesting that his thinking may have been shaped by traces left in the collaborative process with his peer.

Overall, participants demonstrated how they responded to emerging traces from various agents in the programming environment, including professors, teaching assistants, and peers. These interactions were not directive but created conditions that perturbed their thinking, prompted further exploration, and contributed to the co-evolving of their understanding as they engaged with their environment.

Agents' Actions and Perceptions

The next two themes, Learning Strategies and Student Agency and Dynamic Problem Solving, focus on how participants acted upon and made sense of perturbations that emerged within the programming environment. These themes highlight how learners perceived opportunities for action and responded adaptively by developing strategies, exercising agency, and refining their approaches through ongoing dynamics problem-solving within changing conditions.

Learning Strategies and Student Agency

Throughout the analysis, it became evident that participants' sense of agency was enacted through their ongoing coordination with the programming environment, engaging with signs that they responded and assigned significance to through interaction, thereby developing learning strategies to guide their inquiry and progression. Signs included environmental features such as error messages, starter code structures, instructor prompts like "What do you think?" and peer approaches to debugging or remixing. Strategies included techniques (such as note-taking), approaches (such as inquiry-driven exploration), and methods (such as cooperative engagement), all brought forth in response to the evolving demands of the environment. This process resembled a form of bootstrapping, where learning strategies were not applied as predetermined tools but unfolded through engagement. Agency, in this view, was not simply about independent decision-making, but about the participants' capacity to orient themselves within a shifting field of affordances and act accordingly. In the excerpt below, Nancy described how her sense of agency emerged through embodied interaction with the programming environment and co-regulation with other agents during problem-solving:

But because it was near the end of the semester, we had built a lot of our knowledge and skills of having that...agency of student-driven learning, asking questions rather than being given the answers. When asking questions [to Professor or teaching assistant], we weren't even given answers, we were given kind of “What do you think? Or did you try it out, or have you tested it?” So having some sort of code and modifying it – so that use-modify-create was a great skill that was constantly used throughout this course that I strongly used Having that, the interpersonal and intrapersonal domains that were taught in class as well, understanding what it is you want the computer to do. Asking yourself...if you can't program right from the start, taking it to a piece of paper, writing it all out, and then transforming that into a program. Those were all skills that were used primarily. I would say the agency, the reuse-modify-create, the debugging process, and the reusing and remixing were all strong skills that were constantly being developed through having that programming experience.

As suggested in the excerpt above, agency manifested through various learning strategies and techniques, including engaging in self-directed learning, posing questions to guide personal discovery, and employing a reuse–modify–create approach to carry out goal-oriented activities. Across participants, a range of strategies emerged, such as collaboration, peer teaching, and chunking information into manageable parts. These learning strategies often appeared to be enacted in response to pedagogical structures, interaction with other agents or within changing semantics in the mathematics programming environment.

Notably, pedagogical moves seemed to prompt participants to develop adaptive strategies that reflected their agency. For example, the use of Socratic questioning such as “When asking questions, we weren't even given answers... we were given kind of ‘What do you think? Or did

you try it out, or have you tested it?’” appeared to stimulate Nancy to think critically and act accordingly. She developed her own approach: “Asking yourself... if you can't program right from the start, taking it to a piece of paper, writing it all out, and then transforming that into a program.”

Table 7 outlines some of the learning strategies and techniques found in the data that participants enacted in response to perturbations.

Table 7

Excerpts From Interview Transcript Showcasing Learning Strategies and Techniques Participants Enacted in Response to Perturbations.

Learning strategies and techniques	Excerpts from interview transcript
Self-directed learning	Gina: So, it was more of that nature where I've done a lot of coding on my own because of those final projects, so, therefore, it was kind of easy in terms of knowing where I was going because this is just a structured assignment, so it was kinda of simple in doing so.
Reuse-modify-create	Ryan: So, this one is based on assignment three, so that's where we got this code from. This is my code from assignment three that then adapt for this purpose. Where did I get that? Well, that - I did that myself. I built that from nothing, with the help of the course material. I was advised by going to the tutorials and seeing how they solved the problem in assignment 3.
Metacognitive strategies	<p>Ryan: and just thinking about it, thinking about it a lot, almost to the point of obsession, I suppose [laugh].</p> <p>Interviewer: So, when you were thinking about it, were you programming too?</p> <p>Ryan: Yeah, and the thinking always. But sometimes thinking without programming just thinking like on my spare time when I was eating or something like that. These problems, they bother me.</p>
Note-taking, chunking	Nancy: And then we had to be able to interpret and understand that ourselves. So, I had notes within lecture, and I would write a series of

Learning strategies and techniques	Excerpts from interview transcript
	<p>things... But we knew what we wanted to happen, so having that kind of almost end goal... How do you get to that end goal?</p>
	<p>Amanda: I definitely did it in sections. I think a part of it's because it's overwhelming trying to think of it all in one component and it's easier to be like, "Oh, I got to do this part. And then I have to do this part."</p>
Research	<p>Interviewer: So, did you immediately know that you were supposed to use those mathematical concepts?</p>
	<p>Luca: [O]nce I had researched it, I knew this is how this is supposed to work, so these are the concepts I need to use.</p>
	<p>Luca: ... I'd have the thought in my head like I need some way to have a letter converted into a number, so I you know, Google it and I'd watch YouTube videos and read online forums about other people and their problems and then kind of figure out where to go from there with that, you know.</p>
Collaboration	<p>Nancy: Sometimes it was just more accessible to access my peers than it was my professor or TA[teaching assistant]...but I did visit office hours as well if I did find that I was struggling...</p>
	<p>Amanda: if you hit a wall, I guess you could say then you sometimes seek help from others and be like, "hey, like, where did you go from here in this aspect". So, I would say that's part of it and then, when other resources don't work, then you're... Yeah, double checking also, I would say is another aspect of working with peers, being like, "did you get the same thing as me? I just like to want to make sure I'm on the same page." And then if you have different answers, you're looking at why you have different answers and re-evaluating your code, and then also re-evaluating what they have as well in their methods.</p>
Peer-teaching; collaboration	<p>Nancy: Collaboration. So, working with peers, having that cooperation with one another, being able to ask each other questions and help each other out almost to the point where we're their teacher and we won't necessarily give them the actual answer, but we'll ask them, So, tell me what your program does. So, what is this specific line?' Instead of actually telling my friends the problem, I would encourage them... because if my code wasn't working, they would do the same for me. If mine wasn't working, they would say, 'OK, [Nancy], I'm not gonna tell you what the answer is, but let's go through your code line by line'</p>

Learning strategies and techniques	Excerpts from interview transcript
	<p>and it was just this idea of working together... Sitting beside those students, I saw how we were helping each other out, not by giving answers, but by asking guided questions to help determine the answer themselves.</p>
Collaboration; step-by-step problem solving	<p>Ryan: And I was going over with her - with my teammate - about how the von Neumann Middle Square method works. So, we have to figure out - the first step I think was figuring out this this formula right here. Here it is. This is what the entire project revolves around.</p>
Inquiry-based learning	<p>Althea: Like further down I made some questions of my own just two short questions. You know, in the assignment itself and then the last question wasn't something that we actually like, covered, but then I was really curious as to see how I could do this. So, I had to ask and say like, oh, how can I compute this, you know, and show this in my work.</p>

Table 7 illustrates how participants' sense of agency emerged and evolved through interactions with the programming environment and other agents, driving the enactment of diverse learning strategies. Gina exhibited self-directed learning by independently applying her coding knowledge to complete her assignment. Ryan's obsessive thinking and his employment of the reuse-modify-create approach demonstrated agency in problem-solving. Nancy's methodical notetaking and chunking highlighted her agency in planning and strategizing towards achieving her end goal, and Luca's proactive research to understand the mathematics reflected his agency in applying proactive learning strategies. Deciding to collaborate, such as through peer teaching and comparing results, fostered shared learning and critical thinking, as demonstrated by Nancy, Amanda, and Ryan. Althea's IBL showed her proactive drive for deeper understanding.

Overall, participants demonstrated a dynamic and situated sense of agency, employing a range of learning strategies in response to evolving conditions in the programming environment. These strategies reflect how guided inquiry enabled both cognitive and embodied engagement in their learning processes.

Dynamic Problem-Solving

Participants responded to perceived traces within the programming environment by continuously analyzing evolving situations, incorporating new information, adapting their strategies, and often collaborating to modify and redesign code. They engaged in iterative problem-solving using mathematical and computational tools. This theme highlights key aspects of this engagement, including adaptability, collaboration, and ongoing refinement. Together, these elements demonstrate how participants navigated and addressed complex challenges through sustained interaction and responsive adjustments within the environment.

While there is some overlap between Learning Strategies and Student Agency and Dynamic Problem Solving, as both involve participant-initiated action, the two themes differ in focus. Learning Strategies and Student Agency emphasizes participants developed strategies in response to prompts, reflecting a sense of agency that emerged through their interaction with the programming environment and other agents. Here, agency refers to how learners oriented themselves and made use of environmental affordances through strategies shaped by experience. In contrast, Dynamic Problem-Solving highlights how they adjusted their approaches, applied computational and mathematical tools, and collaborated to refine their work through real-time, ongoing interaction. That is, this theme focuses on the moment-to-moment process of responding to perturbations through continuous decision making, problem solving, and code refinement, rather than the broader development of learning strategies.

Adaptive Problem-Solving. The analysis revealed that as participants engaged in designing their EO and investigating its use, they adapted their thinking and actions by utilizing tools and learning strategies to develop non-routine solutions to encounters in the programming environment. This adaptation was driven by a learning and feedback mechanism, where participants responded to prompts from the environment and other agents to meet assignment requirements. This subsection begins by addressing problem-solving as it relates to learning, feedback mechanisms, and the adaptation of strategies. A following section explores dynamic problem-solving through collaborative action, while subsequent sections examine the occasioning tools participants employed in their problem-solving processes.

Across participants, there was a clear process where agents gather information and insights from their actions and the environment to continuously refine and adjust their problem-solving strategies, thereby facilitating the evolution of an adaptive problem-solving process. That is, through a learning and feedback mechanism, participants dynamically responded to perturbations during interactions, which informed their thinking and further shaped their subsequent actions. The following excerpts from Ryan highlight this mechanism:

The basis of this big part was an earlier assignment in the course, because this problem that we're solving, is modeled after a related problem that we were doing with the logistic map. So, that was the basis: we started with that and then we built on to it to make it suited for our problem. So, we mutated the original code that we had had from that previous until it worked for our purposes. It got a little bit messy(laugh), but you can see here, we realized quickly that the von Neumann Middle Square method was non-random; so, we immediately wanted to program this in such a way that it was going to tell us the length of cycles and find cycles for us.

Ryan's experience illustrates the learning and feedback mechanism by showing how he and his teammate used insights from previous work to inform their current problem-solving process. Initially, they built upon a previous assignment involving the logistic map, using it as a foundation for their new problem. This approach highlights a continuous refinement process, where prior knowledge was adapted and improved based on new challenges. As they worked on the problem, they dynamically adjusted their code in response to feedback from their initial attempts. For instance, they quickly identified that the von Neumann Middle Square method was non-random and made modifications to address this issue. This iterative process involved modifying their original code and making necessary changes to better suit their needs. Through this adaptive approach, Ryan and his teammate demonstrated how feedback from their earlier work and ongoing interactions with the task guided their problem-solving strategies, leading to a solution that fits their purpose. This reflects a clear learning and feedback mechanism, where insights from past experiences and real-time adjustments facilitated their progression in solving the problem.

To expand on how participants engaged in dynamic problem-solving through interaction with the programming environment, consider Nancy's explanation of her general approach to programming.

So, the reason that I had to change it was because I went into it with an idea. I tried to work with this idea, and then it wasn't working. The lines wouldn't be producing the same way...When we use Scratch, I'd use different building blocks, and even with those blocks, I realized, OK, "this is not working. This is this is not doing what I'm asking it to do" So, I needed to scratch it all, delete it all, and start with something fresh and come up with a different idea... It had to do with understanding that, OK, if you're running into a

bug, that's probably a syntax error, and that's just kind of throughout my entire way through every single assignment with working with Python Jupiter was understanding Where is it that's going wrong? What is it that I typed that was incorrect? What is it that you know I'm defining that's not actually correctly defined as...

Nancy explained that for “every single assignment,” she had to engage in a dynamic process of problem-solving where she had to engage cognitively and metacognitively to achieve her objectives (i.e., doing what she was asking it to do). Sensing that the program “wasn’t working,” Nancy demonstrated self-directed learning, taking charge of her own learning, applying cognitive and metacognitive strategies, leading her to debugging her codes toward accomplishing the goal-oriented-activity of developing an EO. Nancy engaged cognitively, logically deducing issues based on a broader understanding of programming logic: “if you're running into a bug, that's probably a syntax error” (see above). Metacognitively, she analyzed her own learning process and understanding, considering what might be causing errors (e.g., syntax errors) and reflecting on her own actions and definitions within her code. Here, the adaptive problem solving emerges in response to the changing situation of co-action. It is enacted together with learning strategies and techniques such as self-directed learning, metacognitive strategies and reasoning (i.e., analyze information, draw conclusions, and make decisions based on evidence and logical thinking).

Likewise, Luca’s explanation of his experience, below, shows a dynamic process of adapting his strategies due to encounters in the programming environment.

What I found the most challenging part about coding things in general is you code up your program, you'll think it will work. Everything looks good, and then you run it, and then it doesn't work, and then you're like, why doesn't this work? This looks like it should

work. Everything seems fine. And then you spend like 2 hours trying to figure out what's wrong with... then you finally realize what's wrong and it's been staring you right in the face the whole time.

Luca's experience demonstrates adaptive problem-solving through his dynamic, goal-oriented approach to coding. He explained his general experience of tackling non-routine problems, where his program appeared to be correctly coded but failed to run as expected; engaging cognitively by logically thinking through his program and metacognitively by thinking about his thought process based on the current state of the program; and debugging his code, thereby also exhibiting self-directed, inquiry-based and active learning. Overall, he applied established coding principles deductively as he analyzed and reassessed his code to identify and correct errors. This iterative process of coding, testing, and revising exemplifies adaptive problem-solving, where he continuously adapts his approach to overcome challenges and achieve his goal.

Collaborative Dynamic Problem-Solving. In addition to individual effort, the analysis reveals that adaptive problem-solving was enacted collaboratively, when, based on prompting from the environment, an individual initiates collaboration to address challenges encountered in the environment, as they work individually. The effort to solve the problem involves actively engaging in communication, sharing diverse perspectives, and adapting strategies based on real-time feedback and changing conditions to achieve a common goal, even though groups or a single individual may also work separately. In short, collaboration is initiated as a learning strategy for problem-solving arises in the programming environment. The following excerpt from Amanda and Nancy depicts this:

Amanda: I feel like it remains individual and unique at first because you're trying to work on it on your own at first, and then if you hit a wall, I guess you could say then you sometimes seek help from others and be like, "hey, like, where did you go from here in this aspect." So, I would say that's part of it and then, when other resources don't work, then you're.... Yeah, double checking also, I would say is another aspect of working with peers, being like, "did you get the same thing as me? I just like to make sure I'm on the same page." And then if you have different answers, you're looking at why you have different answers and re-evaluating your code, and then also re-evaluating what they have as well in their methods. Does that answer your question?

Nancy: So, if I'm constantly getting an error and an error, I'll do everything I can, and when it gets to the point where I've tried everything I can think of, I go and seek guidance at that point...I then realize, OK, I need to talk to my professor or my peer or my TA[teaching assistant].

She added:

Nancy: I never really went through my code and compared that with my peers. I went through comparing their final results. If they were off by a number or two, or if something was going on, then we realized OK, we have a problem. What? What is wrong with it? For example, if the count number of T was wrong, maybe I forgot to add a certain number to it, right? Because it's still outputting a value, so it had to do a lot with debugging and understanding, "OK, maybe something is wrong."

Amanda and Nancy's experiences demonstrated collaborative, adaptive problem-solving through their approaches to tackling challenges. Initially, Amanda worked individually on a problem, embodying a self-directed approach. Upon encountering difficulties, she sought help from her peers, marking the start of collaboration. She asked others how they had approached the same aspect of the problem, and when other resources failed, she engaged in peer review, double-checking her work against others to ensure accuracy and consistency. If discrepancies arose, they collaboratively re-evaluated their codes and methods. Similarly, Nancy tackled persistent errors by first exhausting her own strategies and then seeking guidance from professors, peers, or teaching assistants. She compared final results with peers to identify discrepancies. Both Amanda and Nancy engaged in an iterative process that combined individual efforts with collaborative review, leveraging both personal and collective insights to adapt and solve problems effectively.

Problem-Solving Using Computational Tools. As participants engaged in adaptive problem-solving, they creatively and iteratively occasioned computational tools in response to environmental dynamics. Computationally, tools include foundational computational concepts (such as conditionals, operators, and variables), iterative practices (such as step-by-step code building, testing, modifying, remixing, and reusing), and techniques (such as the use-modify-create framework), all supported by an emerging computational perspective.

Foundational Computational Concepts. As a dynamic response to prompting in the programming environment, participants occasioned computational concepts. Computational concepts were occasioned based on intuition developed from an experience of interaction in the programming environment or in novel ways in the process of adaptive problem-solving. Luca's

experience with the programming environment illustrated in the excerpt below depicts how he occasioned computational concepts intuitively:

A lot of them are a lot of this type of stuff we'd learn through our labs in this class, so... Like the first couple weeks was all dedicated to loops and IF statements and stuff and then we went on to more complex things like arrays and stuff like that ...I found after taking the labs, you know when you'd have to apply these concepts. Like when I started doing this program, I'd know OK, "this is going to have to be an IF statement or then this is going to have to be a loop, you know, and things like that... You develop an intuition for how and when to use different objects, I guess."

Luca's experience (above) shows how computational concepts emerged through his active engagement with the programming environment in response to prompting from interaction. Initially, structured activities on computational concepts such as loops and IF statements, followed by more complex structures like arrays, helped him develop an embodied understanding. Thus, Luca intuitively enacted specific concepts when prompted since they had been enacted through his previous interactions.

The excerpts below show how Luca also occasioned computational concepts in a novel manner, together with learning strategies and interactions with external resources:

So...this program, in specific, did involve a lot of research in general because I know what I want the program to do and I kind of have a rough idea of how it should work but it's more "I don't know if there's a specific way for it to work," like when I spoke about the dictionary. Before writing this program, I didn't know that there was such a thing in Visual Basic that would assign a letter to a number, and it's called a dictionary... I'd have the thought in my head I need some way to have a letter converted into a number, so,

Google it and I'd watch YouTube videos and read online forums about other people and their problems and then kind of figure out where to go from there with that, you know

The excerpt above illustrates how novel computational concepts can emerge from targeted research and engagement with external resources, driven by the practical demands of a programming task. Luca had a specific goal in mind for his program and a general sense of how it should function, but lacked knowledge of the precise methods to achieve it. He needed to convert letters to numbers but was initially unaware of the dictionary feature in Visual Basic. This necessity prompted him to research online, watch YouTube tutorials, and read forums. Through these activities, Luca discovered the dictionary concept and learned how to apply it to his problem.

Iterative Development Practices. In adaptive problem-solving, participants utilized various practices to iteratively and creatively respond to prompting in the programming environment. This includes testing, modification, remixing, debugging and modularizing code. When asked about how he developed his program, Ryan expressed how he occasioned computational practices to a state where it worked most efficiently:

We do refer to functions a lot. ...Which involves jumping around. So, this we defined here a function and then we actually use self reference a lot. ...We also refer to this little collection of nice little modular functions up at the top here... it was organic, like what dependencies does this function have? 'Is it important that it be inside here?' And in these cases, the answer was no... It was a process amendment because we started with one model which was from a previous part, a previous assignment, right. And we kept amending it. And through this constant process of amendment - I think even when we were running data, I was amending the program as I noticed things, constantly amending

and this turned out to be just the most optimal solution. ...It just turned out this way as a result of our trial and error I guess, while we were working on it, even as we were working with it...

Ryan's experience illustrates how he utilized various practices to iteratively and creatively respond to prompts in the programming environment. While working on his project, Ryan engaged in a process of continuous amendment, where he started with a model from a previous assignment and kept refining it. He integrated modular functions and employed self-referencing to streamline the code, making it more efficient. This organic development involved asking questions and making decisions about dependencies and optimal placements within the code. As he encountered issues during data runs, he amended the program to optimize performance. This trial-and-error approach, combined with running the program fully to collect data and making adjustments based on observations, exemplifies his adaptive and dynamic problem-solving process. His iterative method of coding, testing, and revising highlights how he creatively responded to the programming environment's challenges using computational practices to achieve the most efficient solution.

Problem-Solving Using Mathematical Tools. As participants engaged in adaptive problem-solving, they continuously enacted mathematical knowledge to develop algorithms, solve problems, understand and apply mathematical models, adhering to formal constraints. This involves the occasioning of mathematical operations and processes dynamically. This theme highlights the specific mathematical techniques, including standard (e.g. addition and subtraction) or more complex operations (iteration and transformation), and emphasizes the cognitive processes (e.g. abstracting and reasoning) that participants occasioned in applying mathematical concepts within a programming concept.

Mathematical Operations, Processes, and Dynamics. The study revealed that participants occasioned various mathematical operations and processes as they were involved in the dynamic process of co-action. Luca and Ryan's experiences exemplify this. Responding to the question of what mathematical formulae and concept were used in developing and implementing his EO, Luca states the following:

Yeah, modular arithmetic, matrix algebra, finding the inverse of a matrix. Specifically finding the inverse of a matrix mod 27...So, with this form of encryption, called the Hill cipher, everything needs to be encrypted mod 27. So, that was not the case, however, for the other encryption program that we had previously been assigned. ... We chose Mod 27 here because it allows you to use a space when you encode something so like, you can encode like a whole sentence or a paragraph instead of just like having things all stuck together without spaces. But normally you'd use mod 26. That was just kind of an extra thing that we did...

The transcript indicates that Luca occasioned several mathematical operations in their work on encryption, particularly focusing on modular arithmetic and matrix algebra. Luca used modular arithmetic, specifically mod 27, as part of the Hill cipher encryption method, which involves finding the inverse of a matrix under this modulus. These combined operations in term involves varying operation such as enumerating (e.g., 0 to 26 for mod 27), identifying relationships (e.g. understanding character-to-number correspondence, matrix inversion, and modular arithmetic relationships., transforming elements), and transforming (encrypting plaintext to ciphertext through matrix multiplication and mod operations, and decrypting ciphertext back to plaintext using the inverse matrix).

Having settled on their topic of exploration, Luca researched the topic and knew specific mathematical concepts that he was supposed to use. He said, “once I had researched it, I knew this is how this is supposed to work, so these are the concepts I need to use.” Luca first responded to the task requirements by researching the Hill cipher encryption, which involves understanding and occasioning specific mathematical concepts. His research then prompts further exploration and refinement of his approach, as he discovers additional details and considerations (such as the use of mod 27) that inform his implementation. This iterative process of researching, engaging with mathematical concepts and operations based on prior and new insights, and refining the approach in response to task requirements demonstrates how mathematical operations were occasioned in the process of adaptive problem-solving.

Participants’ interactive development of their EO involved several mathematical processes. For example, specializing, conjecturing, generalizing, convincing, problem-solving, reasoning and proving, reflecting, connecting, communicating, representing and selecting tools and strategies. Luca’s excerpt below illustrates participants’ occasioned mathematical processes. Luca’s data exemplified specializing, representing and reasoning in adaptive problem solving:

I kind of did it was like I'd have an example already written out, so I knew kind of the steps of how it should work like on paper, and then what, like the solution should be like what the message should encrypt to and what it should decrypt to and then I kind of tried to do it with that given example in the program.

He began by specializing in specific examples of encryption and decryption, using predefined cases to understand and clarify the steps involved. Luca represented these steps and their expected results on paper, creating a detailed record of how the encryption and decryption

should function theoretically. He then implemented his program, which prompted him to occasion other mathematical processes, for example, reasoning and problem-solving:

Like as I spoke about before, the dictionary that was mainly to get the matrix algebra to run smoother...Because then like each letter can be added to a list and then that can be turned into an array and then we can multiply arrays together. ... We had tried multiple other ways to do that task, and this was the only one that we could find that would effectively work...and so that one was strictly due to mathematics. Another change here is this sub procedure that's called check even. So, basically, what it does is if you entered a message that had an odd number of characters, it will check if the message has an odd number of characters or even, and if it has an odd number of characters, it will repeat the last character twice. Because if that didn't happen, then the code wouldn't run properly because the matrixes wouldn't multiply together properly.

By implementing the steps that he represented on paper in his program and comparing the program's output with the known solutions, Luca engaged in reasoning to verify that his implementation performed as expected. This process also involved problem-solving as he adjusted and troubleshooted the program to ensure it aligned with the theoretical results of his mathematics.

Moreover, from the above data, mathematics dynamics were at play during interaction. Prompted by the program dynamics (i.e., the relationship between inputted letter and output encoded data), Luca leveraged the *iterative process of mathematical thinking* to determine how to refine the code to meet the mathematical requirements of multiplying the matrix. The computational tool called dictionary was introduced to facilitate matrix algebra to convert letters into arrays, allowing for smoother multiplication of arrays. To do this, he had to recognize *the*

pattern that each letter can be represented as an element in an array. He also recognized that the program was only producing results for letters with an even number of characters, so he implemented the "check even" sub-procedure to check the parity of the message length and adjust it, if necessary, so that the matrices multiply correctly. By understanding the cycles of manipulation between variables in the mathematical model, Luca engaged in an iterative process of addressing mathematical constraints, and refining the code to achieve his goal. In this dynamic process, Luca emerges from a vague understanding to a greater capacity to co-act with programming for mathematics learning, stating that "Before writing this program, I didn't know that there was such a thing in visual basic that would assign a letter to a number, and it's called a dictionary."

Similarly, Ryan and Althea's data illustrated how participants occasioned mathematical operations and processes in designing their EO. For example, Ryan leverages various operations, which collectively allow for the manipulation and reduction of a number's digits through a structured mathematical approach:

The first step I think was figuring out this formula. ... This is what the entire project revolves around. We started with the intuition of "we want the middle" ... in general, cutting off the first quarter and last quarter of whatever number of digits you got. And from there we jumped to how do we cut off digits? So, on paper, still we were working. We divide by the power of 10 and then take the floor... what it does is it takes a decimal, and it destroys the decimal part... The int has been used instead of a floor. And because we're only dealing with positive numbers, they do the same thing.

Ryan occasioned mathematical operations and processes that collectively aided him to complete his task, including enumeration, identifying relationships, and transformation. He

enumerated the digits and powers of 10 needed for his calculations, applied division and the floor function (or int) to repeatedly cut off digits from the number, and identified the relationship between digit positions and powers of 10. Recognizing that the floor and int functions are equivalent for positive numbers, he transformed the number through division to reduce its scale and used the floor function or int to discard the decimal parts, effectively cutting off specific digits to achieve the desired results.

Similar to Luca, Ryan engaged in mathematical processes as he dynamically responded to the prompts of the programming environment. His processes included specializing in specific examples to understand how encryption and decryption should work, representing steps on paper, reasoning and proving, and problem-solving as he planned and implemented the theoretical steps of his Hill cipher. He connected mathematical concepts to practical applications, communicated methods and results through his report, and selected tools and strategies for implementation.

Likewise, for the DNA Sequence Assignment, participants occasioned mathematical operations and processes as they interacted with the programming environment in a dynamic process of adaptive problem-solving. Being triggered by her completed program that she was using to explore proportion and probabilities, Athea's transcript depicted the occasioning of mathematical tools:

So, with this part, finding the proportion of the GC I actually was having difficulty with it in my mind, and I had it wrong for a while. And then I was rethinking about it, and I was like the denominator is wrong because I originally had it, the frequency of GC being divided by the regular length of the string. Same as like the proportion of A is divided by the length of my string, but then I realized it wouldn't be that because the number of

possible GCs isn't the same as the number of possible "A"s in the sequence, because it's a single letter, so that I had to work through. And I did that just on paper creating a mini example for me of like 6 letters and I counted how many possible combinations of two letters you could make, and I realized it was always one less than the number of letters in a string. Does that make sense?

Althea engaged in mathematical operations in an iterative act of problem-solving to find proportions, identifying relationships between elements to revise her approach. She applied the process of specializing, representing specific examples on paper to focus on a smaller, more manageable subset of the larger problem. She engaged in reasoning and proving, realizing that her initial understanding of the denominator in the proportion calculation was incorrect, which she eventually corrected.

Overall, the study showed, as illustrated by Luca, Ryan, and Althea that participants engaged in a range of mathematical operations and processes in a dynamic act of problem-solving as they interacted with the mathematics programming environment.

Integrating Mathematical and Computational Thinking in Problem-Solving. In some cases of dynamic problem-solving, participants had to balance both mathematical and computational thinking to develop their EO. Luca acknowledged this when asked whether he learned more about mathematics or programming:

Oh yeah. Definitely about programming. Also, too about mathematics cause like those, especially with like encryption, those kind of go hand in hand cause like you need the mathematics and you need the programming for it to work. So, like I would have never guessed that this type of encryption wouldn't have never worked with a negative number

Responding to a question about why he has to change around parts of his program, Luca shed more light on how he combined mathematical and computational thinking to achieve a functional and efficient encryption system:

When I call dictionary one, it gives me the letter A, and if I call dictionary two, it gives me the letter B. And so on and so forth. You can use that for any arbitrary string too, like you could assign dictionary one to the word apple or anything else, but we did it with the alphabet because that was the easiest way to do it. We had tried multiple other ways to do that task, and this was the only one that we could find that would effectively work...Like as I spoke about that was mainly to get the matrix algebra to run smoother. Because then like each letter can be added to a list and then that can be turned into an array and then we can multiply arrays together.

Luca balances mathematical and computational thinking to develop his encryption operation by using dictionaries to encode letters, ensuring efficient data management within the program. He iterates through multiple methods, selecting the most effective one, which demonstrates his computational problem-solving skills. Additionally, he applies matrix algebra to convert letters into arrays and perform matrix multiplication, integrating mathematical operations into the computational process.

Ryan alluded to this when he responded to a question about how he got started designing his EO for the von Neumann Middle Square method:

In order to figure this out, you have to kind of rationalize like a more practical thing that you do on paper into programming: The von Neumann Middle Square method - what it is? - a method of generating random numbers. You take a number...and then you square it, right? And then you take its middle. That means you cut off the first two and last two

digit and you're just left with another long number. That's your new number... But that works nicely on paper, but not in programming, you see. You could - you can't, really. It doesn't make sense to ask what's the middle of a number unless you know how to find the middle using mathematical formalism.

Ryan balances mathematical and computational thinking by translating a manual mathematical method, the von Neumann Middle Square method, into a functional programming algorithm. He starts by rationalizing the practical steps involved in the paper-based method, squaring a number and extracting its middle digits.

After rationalizing her plan for her EO on paper to use a stochastic difference equation to predict different rates of GHG emission, Althea's description of how she coded the probabilities into her program to reflect different emission rates throughout the year illustrates the application of both mathematical and computational thinking:

I set up stochastic difference equation. So, the way this equation is set up, it has two random variables that are really important. We have X_n which is the amount of greenhouse gas emissions after n months and R_n which is the random variable representing the monthly rate. But there's a probability involved..., and we have two different rates. ...I coded this into my program to reflect that the probability every other time of the year is 75% ... and then when it's 25%, which is mostly during summer months... I had to ask the instructor for, like, just a bit of background here and they're like, "ohh just do a random number and then, you know, do an IF or ELSE loop into this: So, if any number before 75 comes out, you're going to like to return the value of R_n " and that makes a lot of sense... which is exactly the equation I just have in my reflection here... and most of these things were things that the teacher already like give to us in class. So, it was just like me changing it up to meet my expectations for this. Then the

part where the real calculations begin, which I stated in this project is the code I have here. You know, just this was me now running the individual simulations for Althea's work exemplified an integration of mathematical and computational thinking. She incorporated mathematical concepts such as random variables and probabilities into her code by defining two rates for greenhouse gas emissions: a standard 0.05% rate and a higher rate during summer months. This probabilistic approach was translated into her code using conditional statements, where a random number determines which rate to apply. For example, if a randomly generated number is less than 75, a lower emission rate is used; otherwise, a higher rate is applied. Additionally, she adjusted the model to simulate emissions over 120. Thus, the integration of her model prompted her to think about the mathematics and think computationally simultaneously.

Interestingly, for Althea, another mathematical formula, a deterministic difference equation, offers her interaction, but she could not determine the computational tools to support her thinking to develop her EO. Althea's explanation is in the excerpt of data below:

... this was just getting around the part of including the probability, because at first, I was completely stumped. I was like, "how do I code in a probability as well as you know, do the equation because ... it's very similar to ... a deterministic difference equation situation which the professor actually pointed out when I turned in my project. He was like, "oh, this is interesting". But he would also like to see this in a deterministic difference equation format, and I was like "oh yeah, I actually thought of that," but I couldn't bend my head around like the probability as well, which is why I started decided to go with the stochastic difference equation. And that's just the portion we have from here to here. So, I would say that the IF OR ELSE loop really helped with putting in the probability for this situation...

At the end, Althea believed that going through the process increased her understanding of mathematics and her capacity to do mathematics in a programming environment. She stated, “it was crazy to me that just like this small portion alone carries the weight of having the probability, you know, like getting the probability function in this equation.” Althea’s statement suggested that both mathematical and computational capacity must align to engage in interaction.

In summary, in aligning mathematical understanding with practical coding strategies, participants effectively bridged the gap between theoretical models and their computational implementation.

Emergent System Behaviour: Proficiency, Perspectives, and Environment Shaping

The next two themes, Mathematical Computational Perspective and Development of Proficiency share a common focus on what emerges from the system as participants interact with the programming environment and other agents. As participants engaged in sustained co-action to iteratively develop their completed EOs, they shaped the environment and, in turn, developed increasingly sophisticated understandings of programming as a mathematical problem-solving tool, gained proficiency in using it to address problems, and cultivated adaptive dispositions. Together, these themes illuminate how mathematical and computational perspectives and proficiency is enacted, shaped by learners’ evolving interactions, motivations, and experiences within the environment.

Mathematical Computational Perspectives

This theme captures how participants developed integrated mathematical and computational perspectives through embodied interaction with programming tasks. Drawing on Brennan and Resnick’s (2012) concept of computational perspectives, which refers to the attitudes and viewpoints individuals form about computing, its purposes, and its broader impact, and Turkle and Papert’s (1992) idea of epistemological pluralism, which highlights the

recognition that there are multiple valid ways of knowing and solving problems through programming, this theme considers how participants' perspectives emerged dynamically through engagement with both mathematical and computational tools. As participants engaged in programming for mathematics learning, they developed creative and pluralistic approaches by exploring and appreciating diverse strategies and adapting code flexibly. They came to view programming as a tool for mathematical investigation and real-world modeling. Emotional responses such as curiosity, confidence, and frustration emerged alongside learning dispositions including persistence, openness, and reflective thinking, shaping the nature of participants' ongoing interaction with the environment. Collectively, these dimensions show that participants' mathematical and computational perspectives were not imposed externally but emerged through situated and embodied participation in the programming environment. This theme is articulated using four subthemes: Pluralistic and Creative Approaches, Programming for Mathematics Understanding, Affect and Motivation, and Learning Disposition and Mindset.

Pluralistic and Creative Approaches. As participants' sense of self and their orientation to other agents and the environment shifted, they became more attuned to multiple ways of knowing, enabling them to test, revise, and adapt their approaches in response to challenges. This included knowing when to collaborate, how to integrate various computational strategies, and how to adjust existing ideas in light of new insights. The excerpt below illustrates how Nancy and Gina's evolving understanding of programming as a personal and creative process informed their engagement with the environment and others. Nancy was asked about the computational concepts she used, and Gina was asked whether she sought support from the instructor, teaching assistant, or peers.

Nancy: So, I would say loops and conditionals- those were all commonly used. It's what was used a lot in my assignments one and two that I felt more comfortable going

through. Now there are many different ways that I could have gotten to this end, and when comparing with my friends and kind of having that cooperative learning with others, I realized like a lot of them did something different. And we all did something different. So, it was based on what was in my strengths.

Gina: It's mostly my peers. ... It would mostly like confirming numbers, because we all had the exact same files, so we knew like if majority got this number majority like most likely it is right.

It appears that both Nancy and Gina's evolving understanding of programming as a personal and creative practice, along with their openness to epistemological pluralism, shaped how they developed their EOs and interacted with peers. Gina's recognition that there were multiple valid pathways toward a solution fostered a collaborative and exploratory stance. She engaged in cooperative learning by comparing her methods with those of her peers and acknowledging the value of diverse approaches. Her reliance on peer interactions, rather than primarily seeking input from instructors or teaching assistants, suggests that she perceived these peer exchanges as meaningful opportunities for shared problem-solving. Through these interactions, participants confirmed numerical results and maintained consistency in their work, using the shared dataset as a basis for validating each other's thinking.

Additionally, Althea's response to the question about how she sequenced her program reveals how her evolving perspective shaped her engagement with the programming environment. Her reflections demonstrate a creative and adaptive process, marked by flexibility and iterative refinement:

I definitely change things around ... cause even now like you see; I'm changing some things to comments. I'm like moving some things around. There are definitely some parts

I have to like, omit or scrap out when I move on to the next question. And I do feel that in the long run maybe I could implement a way to make them all be available on the code without having to change some things to comments or things like that. But in that frame of mind like in that period of time I could only like make some things comments or like completely scrap things out, because whenever I'm coding, I always have two extra documents open. So, like I called this background work just to like to see how things work and then I remove things here and move things there before I actually work.

This excerpt illustrates how Althea's evolving understanding of programming as both a personal and creative practice conditioned her interaction with the programming environment. Rather than following a fixed sequence, she adapted her approach based on the unfolding demands of the task. Her use of background documents and real-time code manipulation reflects a mindset that was open to experimentation, reorganization, and ongoing sense-making. These practices point to a deepening mathematical-computational perspective, one in which programming was not only a technical activity but also a process of exploration and continual adjustment, shaped by her growing confidence, awareness, and interaction with the environment.

Programming for Mathematics Understanding. The analysis revealed that participants' emerging understanding of programming for mathematics (its uses, implications, and role in the world) took shape through their ongoing interactions with the programming environment throughout the MICA courses. Their understanding was not fixed but enacted through co-action, as participants engaged with tasks, tools, and others to navigate complex problems. Their view of programming evolved into that of a versatile and exploratory mathematics problem-solving tool, which conditioned how they interacted with the environment and oriented their responses. The excerpt in Table 8, drawn from the experiences of Nancy,

Ryan, Amanda, Althea, and Gina, illustrates how this enacted understanding shaped both their programming processes and their broader perspectives on computing.

Table 8

Excerpts Showing Participant Perspectives on the Usefulness of Programming in Mathematics Contexts

Participant	Excerpt
Amanda:	I did really good in MICA 1 and then some parts were definitely more needing help in from the TAs[teaching assistant] and stuff. And then last year was just odd because it was online, but I feel like I gained some more experience and stuff and then I feel like this year I felt a lot more confident in MICA, like computational thinking and using computer science for mathematics...I feel like I knew it was useful, but I didn't know exactly how it would be useful if that makes sense. Like I knew it could be, but I didn't know how and so I didn't have a visual aspect of how it would be useful...I would say I have a visual aspect now of how it would be useful in different areas... I see useful in testing conjectures. And then I see it useful in like changing Functions and then having it graph plotting functions, testing hypothesis. And then implementing it in the classroom
Althea	OK, so when I was doing my research for the assignment, I was really inspired by the things the professor/the instructor did during the course, you know, and most of it was just about, like, measuring population. Like, that's what we even did with deterministic equations. That's what we did with stochastic probabilities... Obviously, it's not a perfect science... So, like I was inspired to say like 'OK, how can I do something real- world like in the real-world sense that really can show how this topic really works.
Nancy	... merging them [textbook-written math formulas and technology] together is what programming and coding is; So, that understanding of both...But I see that strong importance in my personal view on it just seeing how important it is for students at a young age to be exposed to it. Now, this doesn't just mean through by Python Jupiter. I think Scratch is an amazing software program that all students should be exposed to... ...with DNA sequences... I saw the real-life application programming actually has I realized, "WOW programming actually has a purpose to real-world applications that you know, like hospitals can use it, doctors can use anything along those lines, but comparing it to, you know, some sort of proof that was done before ... You know, I

Participant	Excerpt
Ryan	<p>didn't see as much of a purpose in it because I'm always a student who wants to learn to see what the purpose of that assignment is in the real world.</p> <p>Thinking about trying to solve practical problems using mathematics and abstractions. Use abstractions for practical purposes. Really, the process of approximation, you can sum it up as because this is what programming is all about approximating. ... Developing this philosophy is ...and it allows you to get a better insight of how to use the tools of programming because if you know what you want to do and you're given a toolbox then you're going to start rummaging through the toolbox to see what you what want to do, if you can do something with the tools, right?... But I suppose by being forced to program I have learned some appreciation for what it is good for what it does, how it can be useful because it can be useful. In fact, that's the whole point that it be useful. It sacrifices perfection for usefulness.</p>
Gina	<p>Overall, like I think it's useful like it makes somethings go quicker and like sometimes like it's a nice reinforcer in terms of mathematical practices. Of like going back, checking your work or like and those types of things or like can provide like a more effective way instead of doing something by hand.</p>

Amanda reflected on how her confidence in computational thinking and using computer science for mathematics grew over time, influencing her particularly after gaining more experience and support from teaching assistants and others during her MICA courses. She then states that interacting with the course helped her to see the usefulness of programming for mathematics learning. Althea's exposure to real-world application of programming led her to occasion programming to analyze the growth of greenhouse gases in Canada, driven by her curiosity about climate change. Her approach involved researching real-world data and interpreting it through programming. Likewise, based on an emerging perspective developed during the course, Nancy emphasized the importance of merging textbook math formulas with technology and advocated for early exposure to programming tools like Scratch. She also

highlighted the utility of programming in various fields. Being skeptical about the MICA initially, Ryan viewed programming as a means to approximate solutions to practical problems, allowing him to appreciate programming's ability to solve real-world issues despite its imperfections. Moving away from being indifferent towards programming, Gina believed that programming can allow one to check their work efficiently and provide a more effective way to accomplish tasks compared to doing them manually. Collectively, their responses suggest that their understanding of technology as a versatile problem-solving tool was developed through the MICA courses, conditioning how they adapt to problem-solving while programming for mathematics learning.

Affect and Motivation. This subtheme explores how emotional experiences—such as anxiety, frustration, disappointment, annoyance, and enjoyment—emerged during co-action and mediated participants' ways of thinking and acting. Rather than existing in isolation, these affective states were intertwined with participants' developing understanding of themselves as learners and problem-solvers. The following excerpts from Luca and Amanda illustrate this dynamic: Luca's sense of enjoyment and confidence in programming informed the direction he took with his task, while Amanda's nervousness appeared to constrain her initial engagement with MICA 1, shaping how she approached the learning environment.:

Luca: You know we had already done an assignment on a different kind of cryptography called RSA encryption, and I found that a lot of fun, so I decided I'd write another encryption program. So, like I chose to do a hill cipher encryption program... I generally really like to solve problems and stuff. That's why this kind of thing is fun to me. But yeah, it can be annoying at times, especially when you sit there for hours and then you

finally realize what's wrong and it's been staring you right in the face the whole time...

Amanda: for MICA 1, I was definitely very nervous about it and because I would say, "I'm not the most technological person" and then so yeah, first year of MICA, I would say nervous and was not quite sure about stuff but got through. In some respects I did really good in MICA 1 and then some parts I was definitely more needing help in from the TAs[teaching assistant] and stuff. And then last year was just odd because it was online, but I feel like I gained some more experience and stuff and then I feel like this year I felt a lot more confident in MICA, like computational thinking and using computer science for mathematics. And yeah, I was much more confident and could do a lot of it by myself without any help and just

Luca and Amanda's experiences illustrate how affect shaped their motivation and interactions with the programming environment. Luca's enthusiasm for problem-solving, along with the enjoyment that emerged from a previous project involving RSA encryption, sustained his interest in creating a new encryption program using the Hill cipher. His positive affect made programming a fulfilling experience. At the same time, he encountered moments of frustration—particularly when resolving issues that seemed obvious in hindsight. Rather than deterring him, these challenges appeared to reinforce his drive to persist and find solutions. Amanda's experience highlights a dynamic interplay between affect (including nervousness, anxiety, and growing confidence) and motivation (such as determination and the desire for competence). Despite initial uncertainty, she remained committed and gradually built her confidence, even amid the challenges of the previous year's online learning format. As her nervousness shifted

toward a sense of confidence, she became more capable of engaging with computational thinking for mathematics, enabling her to complete tasks with increasing independence and assurance.

Learning Disposition and Mindset. While aspects of learning disposition and mindset may be inferred from the preceding theme, the researcher chose to distinguish them due to their nuanced differences. Learning disposition and mindset refer to the ways participants' orientations toward learning unfolded through their co-action with the programming environment and others. While closely related, each concept highlights a distinct facet of how participants engaged in mathematical problem-solving. Learning disposition captures the habitual tendencies, such as curiosity, persistence, and exploratory engagement, that participants enacted as they encountered and responded to challenges. Mindset, in contrast, reflects participants' evolving beliefs about their capacity to grow and learn, shaped by their lived experience with the environment. Rather than being static traits, both disposition and mindset emerged dynamically through participants' situated interaction with tasks, tools, and peers. Together, they supported co-action by sustaining openness to new strategies, flexibility in thinking, and persistence in navigating uncertainty. The following excerpts from Table 9 illustrate how these dispositions and mindsets were occasioned through co-action with the programming environment.

Table 9*Excerpts Illustrating Participants' Dispositions and Mindsets*

Participant	Excerpt
Luca	I really like learning and learning about this type of thing specifically, so for me it was a lot of fun... It can also be a bit of a challenge when you don't know what you're supposed to learn—like, you don't know what you're supposed to be learning, or you don't really know what you're looking for, but you're kind of just going out there and seeing what you can find that might help you. So, like in that respect, it can be a challenge... But yeah, like it's something I enjoy, and I like doing that.
Ryan	No, mostly it was pencil and paper, talking with my one friend who I collaborated with on the project and just thinking about it, thinking about it a lot, almost to the point of obsession, I suppose.
Althea	So, I said I'm going to take and do like four sets of simulations so I can better understand, you know, the variances and things like that because again I'm more interested in statistics side of things as well... So, for this, the 5th simulation, this is where I was like, OK, this is something really interesting because now you can see the dispersion and the way the population grows over, you know, 120 months. So, I was really interested in seeing how I could use the FOR loop to really go in. So, for each simulation that's run, it just goes in and plots scatter plots, and then it runs a random—it runs a random loop just so I can get one value... I was like, I'm not really interested in having zero, you know. I wanted to show something real even though zero's a real situation. But I was like, 'Canada has never experienced above 757, above 760, so 800 might be a far stretch. Let me come back down.' So, I would say that this question kind of made me—because of the success of my previous parts—this question made me come back, sit down, and actually reconsider...
Nancy	So, every individual programmer is going to come across certain problems. Mine was always it's capitalized sensitive, it's indent sensitive... So, if I'm constantly getting an error and error and error, I'll do everything I can, and when it gets to the point where I've tried everything I can think of, I go and seek guidance at that point and learning from, quote and quote, mistakes.

The experiences of Luca, Ryan, Althea, and Nancy illustrate how learning dispositions emerged and shaped their interactions within the mathematics programming environment. Luca's enjoyment of learning and problem-solving emerged through his active exploration of encryption, where even moments of frustration became part of a rewarding discovery process.

His enthusiasm energized his continued co-action with the environment. Ryan's intense focus and persistent engagement with his project reflected a disposition toward deep, reflective inquiry. His habit of thinking through problems (even outside of class) highlighted a commitment to iterative refinement that guided his programming decisions. Althea's curiosity about statistical patterns and her decision to run multiple simulations revealed a deliberate and inquisitive stance toward understanding variability. Her methodical experimentation reflected a learning disposition that took shape through repeated interaction with her evolving code. Similarly, Nancy's persistence emerged through her willingness to revisit errors, seek support, and refine her understanding over time. Her disposition for problem-solving became visible in the ways she engaged with both the program and her peers. Across these cases, learning dispositions were not simply applied, they were enacted and shaped through dynamic problem-solving, sustaining participants' engagement and adaptability within the programming environment.

Development of Proficiency and Environment Dynamics

From the analysis so far, it may be evident that participants' cognitive and physical actions, including their occasioning of strategies and tools, as well as their affective and learning dispositions, emerged through co-action, facilitating their progression from novice to expert in programming for mathematics learning while developing their EOs. However, this theme highlights that participants' history of engagement with programming, along with their broader experiences, is closely tied to their development of proficiency and evolving environment dynamics of co-action. To illustrate this, two sub-themes are discussed in this section: Agent Dynamics, Experiences, and Completed EOs; and Subjective Relationship and Comfort.

Agent Dynamics, Experiences, and Completed EOs. As participants engaged in ongoing action and perception, responding to prompts from environmental prompts, they

iteratively adjusted their programs, influencing the environment's dynamics and gradually completed their EOs. That is, through this sustained co-action, their behaviours shifted dynamically, shaping both their evolving experiences and the environment itself. This sub-theme explores how participants' histories of interaction, grounded in structural coupling with the environment, supported their ability to bring forth completed EOs. Table 10 presents excerpts from Nancy, Ryan, Gina, and Amanda, who each indicated that their prior engagements meaningfully shaped how they interacted with and responded to the programming environment.

Table 10

Participants' Reflections on the History of Interaction With Programming Concepts

Participant	Excerpt
Nancy	I think because I've been so exposed to MICA courses ever since my first year ...if/then, loops, FOR loops and if/then statements and conditionals and all of that, I've been exposed to right from the very beginning...of my courses. So that's what allowed me to be able to when being taught in class again or being refreshed it as well, I'd be like, 'Oh yeah, I know this...and it made very much sense to me in the moment when programming... There were instances because we were never exposed to Jupiter Python before, I didn't know how to use this program, so that's what I struggled with more. But the if/then statements in FOR loops I was able to understand because we had that pre-knowledge from prior courses from prior assignments... I struggled with more with the beginning assignments...I didn't have that comprehension in my other assignments or courses as much as I did with this one... I wouldn't say I had many difficulties with it because I already had a lot of familiarity with my prior assignments to understand how to modify and change and I was able to actually understand my code while reading it...
Ryan	I kind of already had a good idea of what the purpose of a loop was from the third assignment or the second assignment early on...But, during the course, I learned some interesting thing about loops... I learned...that even if you have two functions that are on their own inefficient, sometimes - it depends on your use I suppose - but sometimes when you use them together, even if they're more inefficient on their own, they're more efficient together than any other options for functions that you would have. And this is important because this can be something missed... If you want the best outcome, you have to consider macroscopic behaviour of the entire program, not just each function. This I found very fascinating. That it didn't depend just on the optimality of the functions themselves,

but also on the optimality of their cooperation... Functional optimization, it really is related to programming. It is a programming concept. I guess it is not found outside of programming. You don't optimize functions, unless you're trying to save time and you're only trying to save time if you're running something and running things is a part of programming, right? You don't run mathematics... All of mathematics is always there... But it's inaccessible to us, you see... We can't get at it, so we program instead, and we use math. We push it into the program and places. And then we run the program, and the program is kind of like an imperfect reflection or partial reflection you could say of some kind of mathematical ideal world.

- Gina I would say it's more because of the practices we had done over like the past two years. We've done a lot of practicing in terms of structured assignment as well as with final projects which is more like you code something by yourself and get it to work type of thing. So, it was more of that nature where I've done a lot of coding on my own because of those final projects, so, therefore, it was kind of easy in terms of knowing where I was going because I've done this before where I have to do something by myself...
- Amanda I guess just doing it myself and working through the problems. And definitely just experience would be the biggest contributor because experience is where I feel like I learned the most, not necessarily when you're seeing someone else do it, but when you're doing it yourself ... Yeah, just the whole experience of working on the assignments I would say is the biggest contributor to it.
-

The excerpts in Table 10 illustrate how participants' histories of structural coupling within the programming environment shaped their ability to respond productively to new challenges. Nancy's early and sustained engagement with MICA courses allowed her to develop fluency with programming constructs like conditional statements and loops, which she drew upon when encountering new tools such as Jupyter Python. Her familiarity enabled her not only to adapt but also to actively reorganize code to suit the needs of her evolving tasks. In doing so, she did not merely respond to the environment but contributed to shaping it through iterative refinement and personally meaningful coding strategies.

Similarly, Ryan's evolving understanding of functional optimization emerged from embodied engagement across assignments. His recognition of how functions could interact more efficiently in combination reflects how programming knowledge was enacted through

experience. In developing his EO, Ryan continuously restructured his code, adjusting relationships between functions to enhance overall efficiency. These choices reshaped the semantic structure of the environment, making it more responsive to further inquiry as he completed his EO.

Gina's progression was grounded in repeated practice with structured assignments and independent projects, through which she developed an embodied familiarity with coding patterns. Her comfort in working independently illustrates how sustained interaction transformed the programming environment into a space she could navigate with confidence. Through this engagement, she constructed a workflow that supported flexible problem-solving and further exploration.

Amanda emphasized the importance of direct interaction with tasks. Her learning unfolded through the act of doing, where each engagement allowed her to build familiarity and confidence. By grappling with challenges directly, Amanda co-created her learning environment, modifying her code and pattern of engagement in ways that made future problems more approachable.

Overall, all six participants successfully completed their EOs. Their embodied histories of co-action supported the emergence of proficiency and simultaneously reshaped the environment's affordances. In doing so, they shaped their own processes and transformed the conditions under which further learning could unfold, enabling them to complete their EOs in a generative rather than passive manner. Each EO reflected an emergent process in which participants negotiated between mathematical ideas and computational structures, continuously refining their programs in response to evolving challenges and insights.

Luca created a Hill cipher encryption program, iteratively refining his code through deductive reasoning and collaborative dialogue. Althea built a simulation model using stochastic difference equations to investigate greenhouse gas emissions, modifying her program in real time based on evolving questions and data. Ryan developed a program to analyze von Neumann's middle-square method, adapting his structure for efficiency and clarity through cycles of observation and revision. Amanda created a DNA sequence analysis program, leveraging past knowledge and peer insight to improve structure and runtime behaviour. Nancy and Gina also completed similar DNA-based programs, drawing on instructional cues, peer collaboration, and prior assignments to build their understanding and refine their approach.

In each case, the EO emerged as participants acted within and upon the environment. Their iterative refinements, prompted by the behaviour of the system, peer and instructor interaction, and their own evolving insights, transformed both the code and their understanding. Through this process, they not only completed functional programs but also co-constructed meaningful learning experiences and their own proficiencies, shaping the programming environment as much as they were shaped by it.

Subjective Relationship and Comfort. Participants in this study developed subjective experiences and attitudes toward programming that emerged through their affective and cognitive engagement with programming tasks and languages. These experiences included growing comfort, confidence, intuitive familiarity with programming languages, and the capacity to navigate complex computational structures. Table 11 presents examples of how participants cultivated a personal relationship with programming environments, enabling them to engage more fluidly and develop a practical understanding of how these systems function.

Table 11*Subjective Relationship with Programming Software and Environments*

Participant	Excerpt
Nancy	...There was only one instance where one number was off but that was, I think just when you run Python over and over and over again, sometimes it modifies the number by a number, 2, ... So, I've had exposure to Visual Basics, so v.net. I've had exposure to Excel, Jupiter, and many, many programming software that you know I can... Scratch is another one. So, with all of them... the reason why I would like Python Jupiter the most only for debugging is because it actually identifies where your error is, whereas with other programming languages, it doesn't.
Gina	It was normally when it came to graphing, there was a lot of bugs in terms of, like in previous, whereas in this time when graphing it was kind of straightforward... maybe it is the familiarity, or it was just because it was more of a random graph. It wasn't a graph based off the data. It was just a random one. So, it was much easier to do whereas in previous assignments like it was more of graphs based off the data. So, sometimes, there were bugs when the graph was reading the data.
Ryan	OK so, a lot of the sub-procedures that were used throughout the entire code, I just put those at the top. They don't necessarily need to be at the top. That was just how I decided to structure, any of the functions that were recalled, like multiple times. I just threw at the top of the code. Also, just for convenience's sake and also to make...if the sub procedures run first before actually like any of the computations take place, it would just make the code more efficient...Just run better. Less chance of crashing.
Amanda	I feel like when it comes to writing in Python, I would say I'm pretty confident. If you were just to ask me about different things within the computer, though, and it wasn't related to code, I wouldn't say I'm not the most technological person.

Nancy's extensive engagement with a range of programming environments, including Visual Basics, Excel, Jupyter, and Scratch, shaped a nuanced, embodied understanding of each tool's affordances, leading her to develop a preference for Python. This was evident in her ability to anticipate subtle errors, such as identifying why Python might be causing numerical discrepancies, and in her familiarity with its debugging processes. Similarly, Gina's repeated interaction with graphing tasks occasioned an intuitive feel for the graphing functions in her programming

environment. Her improved ability to manage bugs over time suggests that she developed a situated, embodied grasp of how graphing algorithms and data visualization routines behave.

Ryan's structuring of his code, placing sub-procedures at the top, reflects a developing sense of how organization supports clarity and performance. His decisions were not merely technical but enacted through iterative engagement, revealing a practical understanding of computational optimization. Amanda's shift from initial nervousness to growing confidence in Python emerged from her continued co-action with the language. Over time, she developed fluency in its syntax and structure, allowing her to work independently and respond dynamically to challenges.

Together, the experiences of Nancy, Gina, Ryan, and Amanda illustrate how participants' situated engagement with programming environments gave rise to subjective, intuitive understandings. These understandings were enacted through repeated interaction and adaptation, shaping how they navigated, anticipated, and made sense of computational tasks.

Chapter Summary

This chapter presents the study's findings on how learners interact with programming environments during mathematics learning, referred to as co-action. Framed by enactivism and stigmergy, the research examined how learners' engagement with tools, feedback, and environmental prompts shaped their computational and mathematical thinking, and how they shaped the environment.

The chapter begins with narrative accounts of six participants, developed through colour-coded analysis of interviews, reflections, and EOs. These narratives provide insight into each learner's stigmatic experience. The second part of the chapter presents results from Thematic Analysis, following Braun and Clarke's (2006, 2020) six-phase model, which revealed six overarching themes: Pedagogical Traces; Environmental Semantics and Social Traces; Learning

Strategies and Student Agency; Dynamic Problem-Solving; Mathematical Computational Perspectives; and Development of Proficiency, and Environment Dynamics. These findings highlight how understanding and agency developed through learners' embodied and situated interactions with their programming environments.

CHAPTER SIX: DISCUSSION AND IMPLICATIONS

The purpose of this study was to explore the interaction between learners and their environment as they engage with programming to support mathematics learning. Integrating CT into mathematics education raises essential questions regarding the advantages of ICT, non-computer science students' understanding of computing, and how to design accessible CT learning environments (Drijvers, 2015; Guzdial, 2008). Although applying CT in mathematics can increase student engagement and reduce failure rates, challenges remain due to a lack of clarity among educators on effectively integrating CT skills into mathematics education (Boaler, 2016; Yadav et al., 2017). Research underscores the importance of well-designed digital tools, active teacher involvement, and appropriate contexts to maximize ICT benefits (Drijvers, 2015).

Frameworks such as constructionism and the instrumental approach offer valuable insights into digital learning processes; however, the advancement of technologies and deeper understandings of cognition call for continued research to move beyond the predominantly one-directional lens these frameworks provide, particularly in contexts where learners engage with dynamic programming environments and develop computational thinking through reciprocal, co-emergent interactions (Abrahamson & Sánchez-García, 2016; Papert, 1980). Addressing these research gaps is crucial to creating evidence-based policies for CT integration in education (Grover & Pea, 2013; Rich et al., 2019).

The chapters of this manuscript outline the study's structure. Chapter 1 introduced the study's motivation, while Chapter 2 reviewed literature on CT integration in mathematics education, highlighting gaps in the literature. Chapter 3 established the theoretical basis of the ontology of interaction. Chapter 4 described the methodology, and Chapter 5 presented the findings.

The study's key theoretical foundation, enactivism, posits that cognition is not simply a representation of a pre-existing world but a co-creation of both the world and the mind through a history of actions within that world (Varela et al., 2016). To align the analysis with this framework, the study also integrated Towers and Martin's colour-coded method with the theory of cognitive stigmergy. This combination facilitated identifying key interaction components within learner-environment interactions, including perturbations, occasioning tools, and the physical structures resulting from students' interactions with both the tools and their environment.

The literature review highlights the benefits of CT in enhancing problem-solving and cognitive skills in mathematics education, while noting ongoing debates over its definition and application. It explores CT's origins in algorithmic thinking and programming and contrasts enactivism with other theoretical approaches, underscoring enactivism's focus on dynamic interaction with the environment. The review discusses both the benefits and challenges of programming integration into mathematics curricula and points to the need for further research to refine pedagogical practices and better align theory with practice.

Using a phenomenological approach, this study explored students' experiences in mathematics programming courses, selecting participants from a Canadian university's MICA program through purposeful sampling to capture diverse perspectives. Data were collected through semi-structured, stimulated recall interviews, field notes, and student reflections, and analyzed using thematic and colour-coded techniques to uncover themes relevant to students' experiences. The major themes emerging from the analysis in the previous chapter include:

1. Pedagogical Traces
2. Environmental Semantics and Social Traces

3. Learning Strategies and Student Agency
4. Dynamic Problem-Solving
5. Mathematical Computational Perspectives
6. Development of Proficiency, and Environment Dynamics

This chapter discusses the study's findings and their implications for cognition and learning, particularly regarding learner-environment interactions in programming for mathematics learning. It begins by interpreting themes from the analysis and discussing their answers to the research questions, shedding light on how these dynamic interactions illuminate cognitive processes in mathematics learning. Additionally, the chapter explores how the findings align with or challenge existing literature.

The chapter concludes by discussing the implications of these findings for educators, particularly in preparing preservice teachers and refining pedagogical practices. It also provides recommendations for theory and future research to address identified gaps and further explore the effective integration of CT and programming in mathematics education. This chapter is organized under the following subheadings: How the Themes Address the Research Questions, Discussion of Themes, Implications, and Conclusion.

How the Themes Address Research Questions

The themes and subthemes identified in this study provide a nuanced understanding of how participants both shaped and were shaped by their programming environment in developing their EOs. Table 12 outlines how these themes align with the research questions and are further developed throughout this section. Together, they offer insights into the perturbations participants encountered, their actions and perceptions, and the reciprocal processes through which they engaged with and reshaped their learning environment. For this discussion, the six

themes with their sub-themes were grouped into the three categories used earlier in the analysis:

Perturbations: Trace-Driven Learning; Actions and Perceptions; and Emergent System

Behaviour: Proficiency, Perspectives, and Environment Shaping (See Table 12).

Table 12

How the Themes Address the Research Questions

Research question no.	Further explanation of research question	Themes and subthemes
Perturbations: Trace-Driven Learning		
1.1	What perturbs learners in a programming environment?	<ul style="list-style-type: none"> - What guides participants' actions to program/do mathematics within the learning environment.
		<ol style="list-style-type: none"> 1. Pedagogical Traces <ul style="list-style-type: none"> • Guidance and Instruction from Tasks and Direct Instruction • Other Teaching Strategies and Techniques 2. Environmental Semantics and Social Traces <ul style="list-style-type: none"> • Environment State and Changing Semantics • Ad-hoc from other Agents
Agents' Actions and Perceptions		
1.2	How do learners respond to the perturbations within a programming environment? (How are they shaped by the programming environment?)	<ul style="list-style-type: none"> - How participants value perturbation and contingently act them - The nature of the occasioning tools (computational and mathematical) that learners employ in action. - How learners gain fluency with tools to
1.2.1	In what ways do perturbations in the programming environment prompt	<ol style="list-style-type: none"> 3. Learning Strategies and Student Agency 4. Dynamic Problem-Solving <ul style="list-style-type: none"> • Adaptive Problem-solving • Collaborative Dynamic Problem-Solving • Problem-Solving using Computational Tools

Research question no.	Further explanation of research question	Themes and subthemes
1.2.2	<p>computational thinking?</p> <p>In what ways do perturbations in the programming environment prompt mathematical thinking?</p>	<ul style="list-style-type: none"> - Foundational computational concepts - Iterative Development Practices • Problem-Solving Using Mathematical Tools
1.2.3	<p>What perspectives do learners form as they do mathematics within a mathematics programming environment?</p>	<ul style="list-style-type: none"> - Mathematical Operations, Processes and Dynamics - Integrating Mathematical and Computational Thinking in Problem-Solving

Emergent System Behaviour: Proficiency, Perspectives, Empowerment and Environment Shaping

The corresponding theme and sub-themes in this row also address sub-question 1.2.

5. Mathematical Computational Perspective

- Pluralistic and Creative Approaches to Programming.
- Programming for Mathematics Understanding
- Affect and Motivation
- Learning Disposition and Mindset

6. Development of Proficiency, Environment Dynamics

Research question no.	Further explanation of research question	Themes and subthemes
1.3 How do learners shape the programming environment for mathematics learning?	- The world of significance enacted by the cognizing agent, with the physical products and the emergence of the tools.	<ul style="list-style-type: none"> • Agent Dynamics, Experience, and completed EOs • Subjective Relationship and Comfort <p>The question was not directly addressed by any specific theme, as the physical environment emerged continuously through co-action.</p>

Perturbations: Trace-Driven Learning

The overarching research question, “What is the nature of co-action between mathematics learners and their environment as they engage in doing mathematics through programming?” was broken down into sub-questions to address what guided participants' actions and how their cognitive selves and environment emerged from co-actions. The first sub-question, 1.1, explored what prompted participants' actions and was addressed by the first two themes: Pedagogical Traces, and Environmental Semantics and Social Traces.

Participants perceived and ascribed significance to various traces in their environment that appeared to reflect pedagogical strategies, which they associated with professors, instructors, or teaching assistants based on their experiences throughout the course. The analysis revealed that these perceived pedagogical approaches, such as structured tasks, guided discovery, and formative assessment, prompted their ongoing interaction, supporting their efforts to program and engage with mathematics within the learning environment as they developed their EOs. The overarching pedagogical orientation participants seemed to sense through their co-actional

experience was IBL, supported by problem-based learning (PBL), which together guided their actions. Inquiry-based approach emphasises the emergence of knowledge as learners actively engage in the learning process, using methods and practices similar to those employed by professional scientists such as self-driven exploration, questioning, and investigation (Pedaste et al., 2015). It involves discovering new causal relationships, with learners formulating hypotheses and testing them through experiments and observations, often requiring the use of multiple problem-solving skills—for example, analytical skills and collaboration (Pedaste et al., 2015). PBL is an instructional strategy where students independently explore and investigate through relevant projects, guided by an overarching question. As Saad and Zainudin (2022) note, PBL fosters knowledge construction and the development of transferable skills. Consistent with both perspectives, participants in this study described learning experiences that integrated features of IBL and project-based learning, which they perceived as shaping their thinking, actions, and progression in the mathematics programming environment.

Participants appeared to move through the inquiry process cyclically, encountering and acting upon evolving conditions, rather than following prescribed steps, as they developed and explored their EOs. These phases, although named differently across studies (Pedaste et al., 2015), unfolded as participant perceived and act in response to affordances and traces encountered from guidance perceived from written or oral tasks or instructions. For instance, the experience of Althea closely paralleled the 5E IBL model (engagement, exploration, explanation, elaboration, and evaluation). Her project began with engagement, as she sensed and ascribed significance to earlier class discussions on population growth, which prompted directions for inquiry. During the exploration phase, Althea researched, calculated, and tested possible modeling strategies, eventually choosing stochastic difference equations. In the explanation phase, she synthesized her findings and articulated them through her EO. In elaboration, she

extended her work by running simulations and refining her approach. Evaluation came when she tested her outcomes against expected values and adjusted her simulations to reflect more realistic scenarios.

This example, along with others in the study, demonstrates that the processes participants engaged in are consistent with descriptions of CT as a problem-solving process involving phases such as formulating, organizing, analyzing, automating, presenting, implementing, and transferring (Barr et al., 2011). Althea's journey of formulating her question, organizing and analyzing data, and iteratively refining her EO reflects CT as a problem-solving approach enacted within a dynamic environment, where perturbations such as pedagogical traces guided her actions. Both IBL and CT are often framed in the literature as emerging through phase-like processes, which contributes to their conceptual alignment (Saad & Zainudin, 2022). Saad and Zainudin argue that the inquiry-based nature of PBL naturally supports CT development. As a result, PBL has become one of the most widely adopted strategies for integrating inquiry into CT instruction across STEM disciplines (Hsu et al., 2018; Saad & Zainudin, 2022). In this study, participants sensed and enacted an overarching pedagogical approach rooted in project-based inquiry, using it to guide their learning and successfully develop completed EOs for mathematical investigation. These findings underscore the value of PBL as a pedagogical approach for fostering CT through inquiry in mathematics programming environments.

Importantly, the study underscores that IBL, particularly within the context of PBL and CT, is not a prescribed, uniform linear process but an emergent process shaped by perturbations within the learner's coupling with the environment. Althea's experience exemplifies this dynamic. As she reached the evaluation phase, her interaction with the programming environment revealed a mismatch between her simulation results and the real-world data she had

researched. This discrepancy did not merely prompt a revision; it perturbed her understanding, leading her to re-enter earlier phases of exploration and explanation. Her actions were not guided by fixed procedures but emerged through ongoing co-action with the environment. The program's output, her accumulated history of interaction, and the real-world constraints formed a nexus that redirected her inquiry. In this way, her understanding was not transmitted or applied but enacted through recursive, embodied engagement with the evolving system.

The emergent character of IBL, situated within PBL and CT, becomes especially clear through the subtheme of Environmental Semantics and Social Traces, which also addresses research sub-question 1.1: what prompted participants' actions. While participants assign significance to Pedagogical Traces they sense, the term pedagogical may suggest something pre-given or externally imposed. This theme helps to clarify that participants' inquiry was not sparked by predefined problems but brought forth through their embodied coupling with the programming environment as they encountered perturbations such as shifts in program output or traces of agentic habits. For example, in Althea's case, such perturbations shaped her ongoing activity, prompting transitions in her inquiry and reorganizing her interaction with the task. For example, she used program feedback and graphical outputs to analyze patterns in her stochastic simulations and selected results that best captured variability. Also, she adapted her approach after observing a peer's experimental strategy trace. These examples support the emergent character of IBL, situated within PBL and CT.

The emergent nature of problem-solving processes in this study underscores why research on IBL (Pedaste et al., 2015) and CT (Shute et al., 2017; Tsarava et al., 2022) often appears complex and varied. While different researchers use diverse terminologies and conceptual frameworks to describe phases of inquiry and definitions of CT, the findings here

suggest that such processes evolve through co-action with a dynamically unfolding environment. Without attending to this situated emergence, researchers risk framing inquiry cycles and CT in overly fixed or reductive ways, potentially obscuring how learners perceive and enact their learning guided by emergent perturbation.

For instance, while participants' engagement in this study suggests that CT aligns closely with problem-solving processes (Barr et al., 2011), their experiences also reflect educational definitions of CT as outlined by Weintrop et al. (2016) and Brennan and Resnick (2012). These educational perspectives emphasize core CT practices and concepts, such as modeling and simulation, data practices, and systems thinking, which emerged through ongoing interaction with evolving tasks, tools, and environments. This suggests that process-oriented definitions of CT should not be viewed as distinct from skill-based ones; rather, both are enacted in co-action with dynamic conditions and support understanding of learning interaction within a CT context.

Althea's example, for instance, illustrates how CT was enacted through a progression that resembled a process. Yet across all participants, including Althea, CT practices (e.g., abstraction, decomposition, debugging) and concepts (e.g., algorithms, loops, conditionals) emerged through their embodied co-action with the mathematics programming environments. Framing CT solely in terms of these educational definitions, however, risks obscuring its broader scope as a dynamic and situated way of thinking, problem-solving, and meaning-making in interaction with complex systems.

Furthermore, Amanda and Nancy also reflected on how programming supported their mathematical reasoning, signaling a shift in how they conceptualized computing, from a set of technical procedures to a tool for exploration and sense-making. These accounts also reinforce an understanding of CT as a thought process that is embodied in interaction with tools and

environments, where the learner's sense-making is shaped by their perceptual coupling with the evolving task landscape.

Given this, CT may best be understood as a situated and emergent form of cognition, particularly in project-based contexts. While frameworks such as those from Weintrop et al. (2016) offer valuable grounding, they should function as generative guides, not rigid prescriptions. Learning frameworks should support flexible, context-sensitive pathways (Guzdial, 2011; Papert, 1980), for engagement, grounded in learners' histories, environments, and evolving co-action with the world.

The discussion so far highlights that participants in this study were prompted by pedagogical traces aligned with inquiry-based and project-based learning, as well as by changing environmental semantics and ad-hoc traces from other agents. However, the discussion also highlights that these perturbations were not fixed or pre-given; rather, they emerged through participants' embodied interaction with the environment and became meaningful as participants ascribed significance to them. Their interaction with their mathematics programming environment was initiated and sustained by perturbations. As objects (e.g., oral or written tasks) afforded interaction, their history of structural coupling shaped the coordination between mind and environment, setting off a non-linear, self-organizing cycle of perception and action, an idea explored further in the next section and consistent with enactivist learning.

Saad and Zainudin (2022) advocate for the integration of project-based learning and computational thinking (PBL-CT) as a powerful strategy to foster student autonomy, critical thinking, and active knowledge construction. While this integration holds promise for deep learning, they note that educators often face challenges due to a lack of practical frameworks to support implementation. Their systematic review identified the benefits of PBL-CT in improving

both teaching and CT skills, while also underscoring the urgent need for frameworks that support experiential learning and guide future research.

This study suggests that an enactivist perspective, when paired with the concept of stigmergy, can help address that gap. Enactivism foregrounds the role of embodied participation and sense-making in dynamic environments, emphasizing that learning unfolds through real-time, situated interactions. Stigmergy, which focuses on how actions leave traces that guide future actions, complements this by making visible how learners co-construct knowledge in iterative and collaborative ways. Together, these perspectives provide a structured yet flexible orientation to support the kind of experiential, inquiry-driven learning that PBL-CT seeks to promote, offering a robust framework for both understanding and enacting computational thinking in educational practice. Considering these research perspectives help address two key challenges in programming-based mathematics learning: the planning paradox, where clearly defined learning goals may lead students to diverge in unexpected ways (Misfeldt & Ejsing-Duun, 2015), and the play paradox, where open-ended, playful engagement can enhance learning but complicates assessment, as students often repurpose tools for their own goals (Noss & Hoyles, 1992).

Agents' Actions and Perceptions

Sub-question 1.2 examines how participants engage with and respond to perturbations, develop fluency with computational and mathematical tools, form perspectives, and build computational thinking and mathematical understanding within a mathematics programming environment. This inquiry is explored through Themes 3 to 6: Learning Strategies and Student Agency, Dynamic Problem-Solving, Mathematical Computational Perspectives, and Development of Proficiency and Environment Dynamics. These themes are closely

interconnected, collectively illustrating how participants actively engaged with and adapted to their evolving environment through ongoing feedback loops of perception and action, gradually building proficiency in programming in a mathematical context. As a result, they are discussed in an integrated manner rather than as discrete categories.

These themes also address Sub-question 1.3, which investigates how learners shape the programming environment for mathematics learning. The data show that participants enacted a world of significance through embodied interaction, bringing forth both tangible products and emergent tools. These tools, shaped through use and sustained engagement, gained meaning within the unfolding activity and became integrated into the shared practices of the learning community, reflecting the co-emergence of form and function (Varela et al., 2016).

This section focuses primarily on Themes 3 and 4, Learning Strategies and Student Agency, and Dynamic Problem-Solving, while the next section turns to Themes 5 and 6 to further address the research questions.

Themes 3 and 4 demonstrate that participants' responses to emerging perturbations were not passive acts of following instructions or executing predetermined plans. Rather, their computational and mathematical understanding emerged through interaction, shaped by perturbations, enacted through embodied strategies, and refined through feedback loops between action and perception within reflective co-action.

As participants engaged in tasks, a network of interdependent physical and cognitive actions emerged within a socially situated context. These dynamic interactions introduced continuous perturbations, which in turn conditioned their responses and shaped the trajectory of their learning. For instance, Luca described moments when his code appeared logically sound but failed to run, prompting hours of debugging. His experience underscores how expectation

violations became sites of iterative adjustment and sense-making, triggering a self-organizing, nonlinear cycle of perception and action, central to enactivist learning.

From this perspective, it becomes evident that participants enacted active learning through their responses to perturbations. Their experiences align with Vale and Barbosa's (2023) description of active learning as involving intellectual, physical, and social engagement, requiring learners to reflect on both their actions and their thinking. They define it as an instructional method that actively involves learners in the learning process, requiring them to reflect on their actions and thoughts. However, an enactivist lens moves beyond viewing active learning as a mere instructional method. Rather, based on the analyses in this study, it is how learner perceive and act on perturbations, triggered by the instructional strategies, that determines whether the learning experience is active. For instructional strategies to be considered active, learners must not only engage with the material but also value the pedagogical encounter in such a way that it sparks intentional action, involving intellectual, physical, and social reflective.

While educators may employ specific strategies or discourse moves to encourage engagement, these actions do not guarantee that learning will be active. Determining whether a strategy has effectively prompted active learning relies on assessing learners' feedback and responses. Therefore, active learning is best understood as learning that organically emerges and reveals the characteristics of active learning as students respond to the perturbations introduced by the educator. It is the learners' cognitive and affective engagement with these challenges that truly brings active learning to life, rather than the instructional method used by the teacher.

This view of active learning, therefore, carries important implications for assessment. Rather than evaluating learning solely by pre-defined outcomes or the effectiveness of

instructional strategies, assessment must attend to how learners respond within the unfolding activity. This includes observing how students take up and transform instructional prompts, the nature of their participation, and how their cognitive and affective engagement evolves in relation to perturbations introduced in the environment. Assessment, therefore, becomes an interpretive act that seeks to understand learning as it emerges in action, through students' sense-making, adaptation, and the enactment of significance in context, rather than as a static measurement of knowledge transmission. This view of active learning again helps to address both the planning (Misfeldt & Ejsing-Duun, 2015) and the play (Noss & Hoyles, 1992) paradoxes.

In sum, participants responded to perturbations not as passive recipients of instruction but through agentic, situated, and iterative engagement shaped by prior interactions and the evolving environment, an enactment of active learning through embodied strategies, explored further through their agentic responses to these perturbations in the following paragraphs.

Student Agency

In the context of this study, active learning embodies the enactment of agency, where various learning strategies, intellectual, physical, or social, intersect with mathematical and computational tools and psychological capacities to address challenges in dynamic, adaptive problem-solving. By engaging with their environment and leveraging available tools, participants co-enact significance and refine their problem-solving skills in a reciprocal, ongoing process.

The study shows that participants' agentic responses to perturbations were nurtured through a history of interactions with the learning environment, where they strategically employed tools and enacted learning strategies in response to evolving conditions. Gina's prior experience supported her EO development, while Ryan refined his approach through feedback

and iteration. Althea's collaborative engagement and real-world connections also cultivated her agency. Rather than applying fixed strategies, participants adapted their approaches through co-action with the environment, peers, and instructional prompts. As Nancy described, her agency emerged through reuse–modify–create cycle, modification, and debugging, triggered by perturbations such as unexpected outputs or open-ended tasks that called for critical thinking and adaptive problem-solving.

This notion that agency is a dynamic process influenced by historical interactions is supported by Moses et al. (2020), who build upon the work of Bandura (1997, 2001), Bourdieu (1977, 1990), and Giddens (1984) on human agency and self-regulation. They define agency as the capacity of students to take intentional actions and make informed decisions, guided by their belief in their ability to influence outcomes, a concept known as self-efficacy. As shown in this study, the capacity for agency, along with the belief in one's abilities, develops over time through interactions.

This study offers valuable insights into how student agency manifests in inquiry-based, PBL-CT environments, particularly within the context of programming for mathematics learning. When faced with challenges, participants employed a variety of learning strategies, including metacognitive and critical thinking techniques, which align with the characteristics of self-regulatory learners (Cho & Heron, 2015). Their experiences reflect key elements of student agency identified in Moses et al.'s (2020) literature review, such as intentional decision-making, self-efficacy, self-regulation, planning, monitoring, and adaptability.

For instance, Luca intentionally using resources like Google and YouTube to research how to convert letters to numbers in Visual Basic. This critical approach to learning highlights his ability to plan and monitor his own learning process, key aspects of metacognition. Similarly,

Gina and Amanda's reflections on their progression in the MICA course revealed a growing confidence in their computational thinking abilities and their capacity to apply these skills to mathematics. This progression underscores their developing metacognitive awareness as they increasingly believed in their ability to influence outcomes through self-regulatory learning. Amanda's experience, marked by her improved understanding and practical use of computational tools, demonstrates the cognitive growth that stems from sustained engagement. Ryan's data emphasizes his self-regulation and flexibility, as he iteratively refined his program under time constraints, optimizing it by eliminating unnecessary elements like redundant graphs. Althea's experience also exemplifies metacognitive skills, particularly in planning, monitoring, and adapting her approach while working on a stochastic difference equation. Her transition from a deterministic model to one that incorporated feedback illustrates her ability to reflect on and adjust her strategies in real time. These examples from the study emphasize the importance of students developing the skills to plan, monitor, and assess their learning progress to develop agency, including the ability to consider options carefully, self-regulate behaviour and emotions, and reflect on their capabilities and goal.

Moreover, Moses et al. (2020) identifies three key themes in literature on agency that support student agency: pedagogical decisions, problem-solving, and student choice and voice. They suggest that teachers can foster student agency by adopting constructivist pedagogical strategies that emphasize personalization, self-motivation, and student-driven exploration, making learning relevant, and providing opportunities for collaboration. They also posit that possessing problem-solving skills and a problem-solving identity is crucial, as both promote independence, reduce reliance on teachers, and help students develop a sense of efficacy and engagement. By offering students choices and space for their voices to be heard, teachers

encourage deeper learning and greater ownership of education. These three key themes (pedagogical decisions, problem-solving, and student choice and voice) were also common in this study as participants respond to perturbation. Participants appeared to perceive the learning environment as shaped by an inquiry-based orientation, occasioning the use of learning strategies such as questioning, critical thinking, self-directed exploration, and collaborative problem-solving. These strategies, prompted by pedagogy, emerged through their co-action with the environment and other agents, supporting active learning and enacting student agency. This suggest that a learning environment that supports agency must be responsive and open-ended, allowing learners to engage with tasks in ways that invite exploration, foster meaningful choices, and enable the emergence of strategies through interaction with tools, peers, and evolving conditions.

Moreover, adaptive problem-solving emerged as a dynamic, goal-oriented process requiring participants to demonstrate flexibility, cognitive and metacognitive engagement, and the ability to adjust strategies based on changing circumstances and new information (Greiff et al., 2012). This process was grounded in a learning and feedback mechanism, where participants sense and dynamically responded to perturbations. For example, Ryan consistently demonstrated flexibility by amending his program by watching it run to develop efficient codes to investigate the von Neumann Middle Square method. Similarly, Nancy engaged both cognitive skills, such as reasoning, and metacognitive skills, such as monitoring and evaluating her own thinking and learning. She applied inductive and deductive reasoning to adapt to environmental prompts and develop her program for her DSA. She deduced errors based on a broader understanding of programming logic, reasoning that “if you're running into a bug, that's probably a syntax error.”

Additionally, she made inductive conclusions about the correctness of her code and her peers' code based on specific results.

Student choice and voice were evident in participants' descriptions of their interactions within the mathematics programming environment in this study. According to their accounts, the tasks provided multiple pathways for engagement, allowing them to think independently and collaborate with peers. The analysis revealed that participants perceived the open-ended nature of the tasks as affording creative problem-solving, enabling them to explore diverse approaches even within the structure of assigned work. Participants like Luca, Ryan, and Althea noted that while the task guidelines provided a general structure, they perceived the freedom to choose projects that aligned with their personal interests. Similarly, Gina, Amanda, and Nancy described how, despite working within structured assignments, they shaped their own learning outcomes and engaged collaboratively to compare solutions and debug code. These experiences suggest that participants interpreted task features as affordances for personal exploration, fostering ownership and creativity. Rather than being passively directed, learners enacted significance through their engagement with tasks and peers. This aligns with the characteristics of rich mathematical tasks described by Boaler (2016), Meyer (2015), and Piggott (2011), which emphasize learner-centered experiences that are accessible, open-ended, and support diverse problem-solving approaches. In this study, participants' enactment of learning within both open and structured tasks illustrates how such environments can support agency and foster meaningful mathematical and computational inquiry by promoting student choice and voice.

The findings from this study suggest that computer technology, when integrated with well-designed programming tasks, can effectively support mathematics learning, specifically, as

it relates to the enactment of mathematical content and processes, which is seen as the “how” of students acquiring and applying mathematical knowledge (Ontario Ministry of Education, 2020).

Moreover, research underscores the growing importance of student agency in education, not only as a desired outcome but also as a process that empowers learners to navigate the uncertainty (Stenalt & Lassesen, 2022; Vaughn et al., 2020). Mathematics, by its very nature, involves dealing with ambiguity, exploring multiple solution paths, and embracing problem-solving challenges (Boaler, 2016). Student agency is central to understanding how learners take ownership of their learning; adapting to these challenges with the context of learning plays a crucial role in shaping this agency (Moses et al., 2020).

Given that this research suggest that mathematical processes and student agency often emerge in tandem and reinforce one another, it is worth considering how agency can be integrated into the conceptual framework of mathematics curricula. Just as mathematical processes and the exploratory nature of mathematics are incorporated into curriculum frameworks such as those in Nova Scotia and Ontario (Nova Scotia Department of Education and Early Childhood Development, 2020; Ontario Ministry of Education, 2020), embedding agency can further empower learners to navigate uncertainty and develop a more adaptive, resilient approach to mathematical thinking.

Different theoretical perspectives offer varying approaches to agency, particularly in relation to structure, such as norms, rules, and institutional arrangements (e.g., curriculum guidelines, school policies, cultural expectations); material or technological frameworks (e.g., access to digital tools, classroom design); and social relationships and power dynamics (e.g., teacher–student hierarchies, peer groups) (Inouye et al., 2023). In literature overview on theories on agency, Inouye et al. (2023), compared the different perspectives from which agency is approached. Social theories often link agency with structure, with differing views on individuals’

relationship. For example, Archer's morphogenesis theory suggests that while structures influence human actions, individuals have the capacity to act independently of these structures. (Inouye et al., 2023). Giddens's structuration theory posits that agency and structure are interdependent, with structures serving as both the medium and outcome of social practices. In this view, individuals internalize structures but also have the capacity to reproduce or transform them through their actions (Inouye et al., 2023). In this case, structures can be a fixed entities to be internalized or reproduced. Bourdieu's habitus theory offers a middle ground, asserting that agents are empowered to act but their autonomy is constrained by the structures they inhabit, which are reinforced through their actions (Inouye et al., 2023). The interaction between the agent and the environment, therefore, is more about the reproduction or transformation of structures through habitual action. In contrast, Inouye et al. (2023) argue that some psychological theories, focus on individual capacities driving behaviour, often without considering structural constraints (Inouye et al., 2023). Stenalt (2021) challenges the idea that student agency in digital education is solely shaped by external structures or individual dispositions, emphasizing instead that agency is formed through relationships, particularly the interdependence between students, educators, and peers.

While these perspectives capture key aspects of enactivist perspective, enactivism offers a distinct approach. Enactivism emphasizes that agency arises through embodied interactions with the environment, rather than being merely a product of internal dispositions or external structures. In this view, agency is not something individuals possess or enact independently of their context; instead, it emerges dynamically from the continuous, reciprocal engagement between the individual and their environment. This perspective aligns with Stenalt's (2021) emphasis on relational interdependence, evident in how participants in this study were guided by supportive structures, such as guidance from professors, teaching assistants, peers, and online

resources, which helped them navigate challenges and enhance their learning experiences. However, enactivism goes further by suggesting that cognition and action are co-constituted through these interactions as structure simultaneously emerges. It challenges the dichotomy between agency and structure by viewing them as inseparable aspects of an ongoing process of sense-making and action.

In a nutshell, participants responded to perturbations through embodied, agentic engagement, adapting their actions in real time as they co-evolve with the programming environment. Using an enactivist framework to explore student agency in a technological and educational perspective has profound implications. This approach emphasizes the co-emergence of cognition and environment, meaning that student agency is not just a matter of individual capability or external structures but is dynamically constituted through interactions with technological tools, peers, and educators. It suggests that agency arises from the ongoing, reciprocal relationship between students and their learning environments, where cognition is shaped by and shapes the context in which it occurs. In a digital education context, this perspective encourages the design of learning environments are not merely settings where learners react to predefined content but are dynamic spaces where knowledge and meaning emerge through co-action, leveraging the affordances of the environment. This orientation invites the development of educational technologies that are not only adaptive but participatory, enabling learners to shape, and be shaped by, their interactions. In doing so, such environments become sites of empowerment where agency is continuously enacted and transformed.

Computational and Mathematical Tools

At this point, it is important to highlight how computational and mathematical tools were occasioned in response to perturbations. Throughout the analysis and discussion so far, it may be evident that participants enacted the three dimensions of computational thinking described by

Brennan and Resnick (2012), alongside mathematical thinking processes aligned with the emergent perspectives of Burton (1984) and Mason et al. (1982, 2010), in direct response to perturbations they encountered. Participants employed computational practices such as abstraction, decomposition, and debugging as they responded to disturbances such as such as unexpected program output, logic errors, runtime behaviours, and feedback from peers or instructors. They began to see computational structures such as loops, arrays, and conditionals not as abstract concepts but as useful tools for resolving breakdowns, gradually developing intuitions about control, efficiency, and modularity.

Similarly, when mathematical models failed to produce expected results, such as unexpected values in stochastic simulations or flawed logic in encryption algorithms, participants engaged in mathematical reasoning. They tested formulas, adjusted parameters, verified assumptions, and interpreted graphical outputs. In these moments, they employed mathematical tools such as proportions, probability, algebraic manipulation, and statistical estimation, along with mathematical processes like problem-solving, reasoning, and communication (Ontario Ministry of Education, 2020). These processes, considered critical for applying mathematical knowledge across grade levels, were enacted through co-action in the programming environment.

Importantly, these ways of thinking did not exist beforehand as static skills. Rather, they emerged dynamically through participants' embodied interaction with tasks, tools, and others in through the environment. Computational and mathematical thinking arose as part of problem-solving activity, co-developing with participants' agency. This suggests that the design of tasks, the flexibility of tools, and the structure of the course played a vital role in fostering these ways of thinking and enhancing teaching and learning.

This conclusion aligns with existing research showing that mathematical learning is deeply shaped by the educator, the nature of the tasks, and the instructional strategies employed (Vale & Barbosa, 2023). The findings from this study also suggest that integrating computational thinking into mathematics education can strengthen students' problem-solving skills by developing logical reasoning and algorithmic fluency, skills increasingly important across disciplines (Aydin, 2005; Garzon & Bautista, 2018; Grover & Pea, 2013; Li & Ma, 2010; Misfeldt & Ejsing-Duun, 2015; Papert, 1980).

It is essential to emphasize that both computational and mathematical thinking tools are emergent. They take shape through students' ongoing interactions with the environment and the unfolding learning context. This emergent nature highlights the importance of a responsive educational approach, one that supports these ways of thinking through thoughtful task design and engagement, while also making space for computational and mathematical tools to be occasioned as students act agentially in response to perturbations.

To emphasize the emergent nature of computational thinking and its integration with mathematical thinking, this study reconceptualized Brennan and Resnick's three dimensions of computational thinking as: emerging mathematical–computational concepts, iterative development practices, and emerging mathematical–computational perspectives. These reframed dimensions better capture the fluid and evolving character of participants' interactions with tools, representations, and problems in their learning environments. Rather than unfolding in isolation, these dimensions emerged dynamically through co-action with the mathematics programming environment.

Although computational and mathematical thinking sometimes appeared to arise independently, such as when Luca debugged a syntax error or when Althea applied probability

theory to refine a simulation, participants were ultimately working toward the same goal: making their systems function effectively. Achieving this required an ongoing coordination between both domains. Even when one mode of thinking was more prominent, the success of their EOs depended on the integration of computational and mathematical reasoning. In this sense, mathematics and programming were not applied separately but enacted together in the creation of coherent, functional systems. This integration was not just a convergence of skill sets but a situated, evolving coordination of thought and action that gave rise to meaningful mathematical–computational activity.

Luca’s reflection on his encryption project illustrates this interdependence. When asked whether he had learned more about programming or mathematics, he noted, “you need the mathematics, and you need the programming for it to work,” emphasizing the co-emergence of computational and mathematical understanding. Ryan’s experience with the von Neumann Middle Square method similarly involved translating a manual mathematical procedure into an algorithmic structure. His sequencing of steps, which involve squaring a number, extracting its middle digits, and iterating the process, demonstrated the coordination of mathematical reasoning with programming syntax to develop a structured EO.

Luca’s use of the dictionary object in programming also highlights the dimension of emerging mathematical–computational concepts. His decision to use this construct allowed him to assign letters to numbers creatively, integrating a computational structure with matrix algebra. Althea’s project further illustrates this integration. She used conditional statements to code varying emission rates into her simulation, adjusting probabilistic values based on observed outcomes to better reflect real-world data. Through this process, she gained mathematical insight by engaging computationally with patterns in the program’s output.

Together, these examples highlight how participants balanced computational and mathematical tools in co-action, using both to make sense of and shape their evolving tasks. These findings echo work by Gueudet et al. (2020), Gueudet et al. (2022), who, drawing on the instrumental approach (Rabardel, 2002), identify three types of cognitive schemes in students' programming for mathematics learning: m-schemes (mathematically driven), p-schemes (programming-driven), and (p + m)-schemes, which integrate both. However, rather than alternating between these modes, participants in this study demonstrated interwoven practices, with mathematical and computational reasoning emerging simultaneously through real-time engagement with evolving tools, constraints, and goals.

In sum, participants responded to perturbations by occasioning computational and mathematical tools in intertwined and evolving ways, shaped by their co-action with tasks, and agents in the programming environment.

Emergent System Behaviour: Proficiency, Perspectives, and Environment Shaping

This discussion so far highlights how participant co-action generated feedback loops between perception and action. These loops not only supported dynamic problem-solving and agency but also contributed to emergent system behaviour. As participants adjusted, refined, and made sense of their work, their understanding and engagement with tasks evolved recursively. This section examines emergent system behaviour in relation to Agent Dynamics, Experience and Completed EOs, and Subjective Relationship and Comfort, also addressing Sub-question 1.2 (How do learners respond to the perturbations within a programming environment?) and Sub-question 1.3 (How do learners shape the programming environment for mathematics learning?).

Over time, the co-evolution of learner and environment led to the development of novel problem-solving strategies, improved mathematical and computational fluency, completed EOs,

and new affordances for learning. Analysis revealed that this emergent system arose from recursive, decentralized co-action with the mathematics programming environment. That is, rather than following a centralized instructional path, participants' agentic responses to perturbations were shaped by evolving tasks, tools, peer interactions, and prior experiences.

In the process, participants left traces, such as refined code, debugging strategies, and shared methods, that influenced future learner behaviour. For example, Althea adopted a peer's iterative testing strategy, while Luca built on his prior experience with RSA encryption. Participants also transformed the environment by integrating tools into their workflows, such as note-taking, modularizing code, or referencing background work, making these artifacts functional components of the mathematics programming environment. This aspect will be explored in more detail in the following subsection. Through this decentralized process, participants developed agency and proficiency via recursive feedback loops, gradually shaping both their own understanding and the learning environment's affordances, ultimately culminating in completed EOs.

This recursive process also supported subjective relationship and comfort which supported navigating increasingly complex problems. For instance, Gina and Amanda similarly credited their comfort and proficiency to accumulated experience with structured assignments and independent projects. Thus, proficiency unfolded as an emergent property of co-action, rooted in prior history, dynamically enacted through problem-solving with computational and mathematical tools, and reinforced by growing agency in adapting to and shaping the learning environment.

Moreover, participants' involvement in adaptive problem-solving seemed to have led them to view programming as a creative and personal endeavour. Nancy and Gina, for instance,

demonstrated an appreciation for their peers' work by collaborating in an iterative debugging process, suggesting that they valued programming as a means of personal expression in mathematics learning. Similarly, Amanda's evolving understanding of how computational tools could be used to test conjectures and plot functions reflected her growing appreciation for the utility of programming in mathematical contexts. Given that these perspectives emerged from the interplay between programming and mathematics, it is insightful, based on the study, to reframe Brennan and Resnick's (2012) third dimension as an emerging mathematical-computational perspective. Overall, in attempting to bridge the gap between theoretical models and their computational implementation by aligning mathematical understanding with practical coding strategies, participants demonstrated the interdependence of mathematical and computational tools in adaptive problem-solving, developing a perspective that is both computational and mathematical.

The analysis revealed participants' affective states such as anxiety and enjoyment, along with learning dispositions like curiosity and persistence, which reflect a growth mindset (Boaler, 2016; Dweck, 2015), mediated how participants approached programming tasks. Luca's enjoyment of encryption challenges, Amanda's progression from nervousness to confidence, and Ryan's consistent focus on problem-solving all demonstrate how emotional responses and learning dispositions emerges as a function of the system, shaping their engagement. These examples align with the findings of Forbes et al. (2019), who highlighted how differing mindsets influenced students' engagement in a mathematics programming environment. For instance, one student, Sydney, initially felt nervous about programming due to her limited prior knowledge and appeared to avoid challenging tasks, which seemed to reflect fixed mindset. In contrast, another student, Jim's, enthusiasm and confidence in tackling programming challenges, despite

his limited initial knowledge, appeared to have demonstrated a growth mindset. The findings in this study underscore how mindset significantly affects students' attitudes and approaches to learning programming, aligning with the broader themes identified by Forbes et al. (2019) and Dweck (2015).

Overall, the emergent system behaviour is the co-evolution of learners and the programming environment into a self-organizing, inquiry-rich system that supported adaptive, agent-driven mathematical and computational learning.

Environment Shaping

Consistent with the concept of structural coupling and the discussion of emergent system behaviour, the mathematics programming environment in this study was not static but continuously reshaped through participants' ongoing engagement. Participants shaped the environment through co-action, where embodied interaction with tools, tasks, and other agents dynamically reconfigured its structure. Their actions were not mere responses to instruction but acts of sense-making that redefined what the environment could afford.

Participants modified the programming environment by generating artifacts, such as loops, functions, data structures, and graphs that reflected their evolving computational and mathematical understandings. These artifacts did not simply record learning; they became functional components of the environment, recursively influencing future action. For instance, Ryan restructured his program to optimize function calls, placing sub-procedures at the top of his script. This decision not only enhanced computational efficiency but also opened new pathways for exploration and refinement. Similarly, Luca's creative use of the dictionary object in Python was not a straightforward application of a predefined tool, but an act of occasioning that allowed him to reshape the environment to suit the demands of his encryption task.

Participants also shaped the environment by incorporating learning strategies prompted by perturbations. For example, when learners responded to pedagogical traces such as instructor prompts or classroom discourse, by taking notes, their strategies extended beyond passive documentation. Note-taking became an embodied and cognitive activity, with participants integrating paper-and-pencil methods into their programming workflow to scaffold thinking and externalize reasoning. Over time, these strategies became part of the functional infrastructure of the environment, allowing for new forms of interaction and understanding. What began as an individual coping strategy evolved into a situated, co-created element of the learning system.

As participants responded to shifting semantic conditions, such as runtime feedback, syntax errors, or unexpected outputs, their interpretations of the problem space and the role of mathematics changed. Althea's iterative adjustment of probabilistic values in her greenhouse gas simulation illustrates how learners enacted significance through real-time interaction. Her changes did not simply update code; they redefined the purpose and structure of the model, transforming the program into a dynamic tool for mathematical investigation. Through such engagement, programming constructs, like loops, conditionals, or graphs, became artifacts of significance appropriated and transformed through action.

Furthermore, in line with Turkle and Papert's (1990) findings, participants contributed personal significance and diversity of thought to the mathematics programming environment. Their strategies, refinements, and comments influenced peer interactions and helped shape the learning culture of the environment. This was evident when Althea extended her project in response to recalling her professor's disposition toward exploration, or when she adopted a peer's technique of iterative background testing. These traces contributed to a shared culture of inquiry that enriched the environment for all learners.

The idea that computational, mathematical, and strategic artifacts function as tools aligns with Shvarts et al.'s (2021) view of tools as components of a non-centralized, multilevel functional system. In this system, tools become integrated within the agent–environment coupling, forming a “body-artifact system” that extends the agent’s potential for action. Rather than being centrally regulated by the brain, tool use is shaped through adaptive co-action with the environment. In this study, while tools were not initially intentional extensions of the agent, they became part of the structural dynamics of the learning environment, expanding its affordances as learners ascribed significance to them

Consistent with Hegedus and Moreno-Armella’s (2010) notion of bi-directional interaction, participants both influenced and were influenced by the environment. Amanda’s growing confidence in Python, for example, allowed her to reorganize code structures independently, transforming her environment into a space of intuitive exploration. Gina’s comfort with graphing tools, developed through repeated prior engagements, enabled her to approach new challenges with confidence, reframing unfamiliar tasks as familiar ones.

From an enactivist perspective, the mathematics programming environment was enacted through learners’ histories of structural coupling. Their ways of sensing, acting, and sense-making were not isolated events but emergent properties of their ongoing relational engagement. In shaping the environment, they simultaneously shaped themselves, co-creating both their learning context and their developing identities as mathematical and computational thinkers.

In sum, as each participant responded to perturbations through recursive cycles of perception and action, they enacted an individual world of significance—shaped by their experiences, strategies, and evolving relationship with tools. At the same time, their contributions transformed the environment into a shared, adaptive space where collective

learning and evolving practices emerged without centralized control. This dynamic interplay between individual meaning-making and environmental transformation reflects the principles of stigmergic coordination and characterizes the emergent system behaviour observed in the study.

Implications for Theory

This study was motivated by the increasing importance of CT in education, particularly within mathematics education. While recent research indicates a growing effort to integrate CT into K–12 classrooms and teacher education programs (Lee et al., 2020; Shute et al., 2017; Wing, 2008), there remains a need for a stronger theoretical and pedagogical foundation to effectively advance CT alongside robust mathematical understanding (Abrahamson & Sánchez-García, 2016; Buteau et al., 2017). Through an enactivist lens, stigmergic lens, this study explored the interactions between learners and programming environments in mathematics, focusing on what guides learners' actions during these interactions and the associated learning possibilities. The findings provide a critique of current theoretical frameworks that guide technology integration in the classroom, addressing the gap between learning theories and the dynamic practices they aim to support.

The theoretical landscape in education has shifted with the increasing use of technology, transitioning from behaviourism, which focused on rote memorization and predictable responses, to constructivism, which emphasizes active creation of knowledge. While constructivism has been widely embraced, it too has limitations, such as its reliance on a pre-given world, its one-directional view of learning, and its ambiguity in practical application. The study's findings highlight the need to account for these limitations when applying constructivist frameworks in educational research. Specifically, the dynamic, bidirectional interaction between learners and their environments, which is especially captured by themes such Environmental Semantics and

Social Traces and Dynamic Problem-Solving, suggests that the way learners interact with their environment is emergent, continually shaped by both the learner's actions and the changing environment. Thus, one-directional frameworks are not sufficient to unravel the dynamic interaction that emerges as learners engage in co-action with their environment.

The enactivist framework, combined with the concept of stigmergy, which remains underutilized in educational research, offers a more suitable theoretical model for understanding learning in interactive contexts, such as technology-rich environments where learners collaborate to solve problems. Unlike constructivism's one-directional approach, enactivism, with stigmergy, emphasizes a bidirectional relationship through structural coupling, where both the learner and the environment co-shape each other through continuous interaction, focusing on the signs or traces left in the environment. This enactivist approach parallels Papert's constructionism, particularly in the context of mathematics learning through programming, where learners engage in a dialogue with the computer, which Papert described as a "mathematical speaking entity" (Papert, 1980). The enactivist framework has the potential to expand Papert's theory, offering researchers and practitioners a clearer understanding of how these interactions unfold. This perspective provides deeper insights into how learners' actions shape their environment and, in turn, how their environment shapes the learner.

The co-action between learners and their environment has significant implications for how CT evolves in the 21st century, especially given the rapid turnover of technological tools. The study highlights the need to rethink key constructs of constructivism, such as problem-based inquiry, active learning, and agency. While these approaches are recognized as central to engaging students in rich mathematical experiences, the research reinforces that the unfolding of these processes are neither rigid nor linear but rather emergent and iterative. The findings offer

new insights into how these constructs can be refined to better reflect the complexities of learning in environments, especially when technology is a central element.

As educational contexts continue to evolve, this study suggests that current theories such as constructivism, much like behaviourism before it, may no longer fully capture the fluidity of learning in contemporary settings. The findings highlight the need for theories that can accommodate the fluidity observed in learners' interactions with technological environments like programming environment. This research underscores the value of the enactivist perspective, which, for example, challenges dichotomous views of agency and structure and proposes that cognition and action co-evolve through reciprocal relationships with the environment. By framing learning as a dynamic interplay between learners and their environments, enactivism offers a robust theoretical lens for understanding how students adapt, shape, and are shaped by their programming environments.

Finally, the study's findings on the integration of computational and mathematical thinking emphasize the need for theoretical models that account for the interdependence of these domains. The concept of emerging mathematical-computational perspectives points to the necessity for theories that acknowledge how mathematical and computational skills coalesce through iterative problem-solving processes. This approach, grounded in enactivism, offers a more comprehensive framework for understanding the complexities of learning in technology-driven classrooms, suggesting that future research should continue to refine theoretical models to better capture this evolving landscape.

Implications for Practice

As the emphasis on CT continues to grow, teachers, educators and other stakeholders, such as curriculum developers, administrators, and policymakers, need clear, actionable

strategies to help students develop computational and content understanding in tandem. This study suggests several practical implications for integrating CT into mathematics education and other non-computing subjects, often reinforcing findings from researchers like Boaler (2016). A key point emerging from the study is the importance of carefully considering pedagogical strategies and learning environments to guide students effectively, particularly in programming contexts. IBL and PBL were identified as effective approaches for fostering active, self-directed learning, where students hypothesize, explore data, and use iterative problem-solving to deepen their understanding.

Educators should aim to create learning environments that prioritize student inquiry, with problem-based tasks, particularly when integrating CT into mathematics. The study revealed that real-world contexts and meaningful, open-ended tasks played a crucial role in engaging students deeply with mathematical programming. When students are given tasks that offer multiple pathways for engagement, it fosters a sense of ownership and empowers them to explore independently. Moreover, designing tasks that intentionally blend both mathematical and computational thinking can help students appreciate the interconnectedness of these skills. For instance, project-based learning that involves solving real-world problems using programming tools necessitating mathematical reasoning is particularly effective, as seen in this study.

The study also found that learners' actions were often shaped by their interactions with peers and their environment, supporting the enactivist view of learning as a collaborative, evolving process. As a result, educators should foster collaborative learning settings where students share ideas, provide feedback to one another, and work together on complex tasks. This collective problem-solving approach, coupled with iterative refinement of ideas, was crucial in guiding both CT and IBL. Additionally, cultivating an environment where students have agency over their learning processes can promote deeper engagement and critical thinking, as agency

was a driving force behind participants' actions in this study. Encouraging students to act as active problem-solvers who reflect on the challenges posed by programming tasks can enhance student problem-solving skills while also nurturing a sense of ownership and agency in their learning.

The findings also highlight the importance of ongoing formative assessment in shaping learning trajectories. Feedback not only from instructors, teaching assistants, and peers but also from the environment played a pivotal role in participants' developmental processes. Educators should, therefore, facilitate immediate, context-sensitive feedback that encourages further inquiry and refinement of students' approaches. Incorporating a variety of formative assessment strategies, such as self-assessments, peer assessments, and reflective journals, can capture students' evolving thinking and provide opportunities for reflection. This may support the development of self-efficacy and helps students engage with uncertainty, which is a natural part of mathematical problem-solving.

Moreover, since this study suggest that learning is an emergent process driven by interaction with the environment, educators must be attuned to this dynamic to provide flexible options for students. Flexible learning environments are crucial for supporting the diverse problem-solving processes that emerge during inquiry. Teachers should carefully assess how students perceive and respond to cues from their surroundings, offering personalized guidance as needed. Additionally, aligning with Papert's view of computers as "mathematical speaking entities," the study demonstrates how technology can serve as a tool for active engagement rather than merely delivering content. Educators should consider programming as a means for students to engage deeply with both mathematics and CT, as it provides real-time feedback and opportunities for experimentation.

Finally, the integration of CT into mathematics education has significant implications for teacher professional development. Teachers need more than a basic understanding of both content and pedagogy related to CT and mathematics to facilitate student exploration effectively. Ongoing professional development should focus not only on the use of programming tools but also on the pedagogical strategies that support deep learning in technology-enhanced environments. Flexible curriculum design and assessment strategies that allow for discovery and problem-solving at students' own pace are crucial. Formative assessments, in particular, can be used to capture students' evolving thinking and guide instruction based on real-time feedback. Additionally, recognizing the role of affective states in learning, educators should promote a classroom culture that fosters psychological empowerment by encouraging risk-taking, celebrating small successes, and normalizing challenges in learning programming. Such an approach helps students build confidence and resilience, equipping them to tackle complex tasks with a positive, growth-oriented mindset.

In general, the findings of this study also point to broader implications for education. As computational thinking becomes increasingly important, the principles highlighted here can inform how learning environments are designed across different subjects. An emphasis on inquiry-based and problem-based learning, opportunities for collaboration, and tasks that allow for multiple entry points can help learners in many fields build deeper understanding and confidence. More broadly, this study reinforces the value of creating flexible, dynamic environments where students' agency and interactions with tools and peers shape the direction of their learning. By recognizing learning as emergent, educators and researchers can better support the development of critical thinking, adaptability, and persistence that extend well beyond mathematics or programming.

By incorporating these practices, educators can create interactive, technology-rich learning environments that nurture both CT and mathematical problem-solving skills. These implications offer practical steps for operationalizing an enactivist approach to education, fostering dynamic and engaging learning experiences that promote over all (e.g., cognitive, social, physical) growth

Implications for Research

The findings of this study suggest several key implications for future research in CT and mathematics education. As CT becomes more integrated into K–12 and teacher education, researchers need to deepen their understanding of how students engage with mathematical and computational concepts in technology-rich environments. The application of enactivist theory, particularly with the concept of stigmergy, offers a novel framework for exploring the dynamic and interactive nature of learning. Future research should continue to investigate how students' cognition and actions are shaped by their interactions with programming tools, moving away from a one-directional view of learning and toward a more iterative, bidirectional perspective. This approach provides a more nuanced understanding of how students develop computational-mathematical perspectives over time.

Researchers should explore how domains such as agency, IBL, CT, and mathematics evolve together, highlighting the interdependence of these concepts. Studies that adopt a more phenomenological worldview, abandoning pre-given frameworks, could provide valuable insights into the emergent and evolving nature of learning. Longitudinal research, in particular, could shed light on how integrated thinking skills develop over time, focusing on the role of iterative problem-solving and its contribution to both mathematical and computational proficiency.

Another critical area for future research is the non-linear and emergent aspects of IBL. Investigating how learners navigate and adapt to complex problem-solving environments, considering the role of agency and other mediating factors, will help researchers understand how these processes co-evolve. Research into the enactivist perspective on agency, especially in diverse learning environments and with various technological tools, could provide valuable insights into how learners' actions and the tools they use influence each other.

Given the fluid and emergent nature of learning in programming environments, traditional assessments may not adequately capture students' progress. Future studies should focus on developing dynamic, real-time assessment methods that reflect the iterative and non-linear nature of learning. Research designs that track student interactions with technological tools in real time, using mixed methods approaches, can provide a more comprehensive picture of the learning process. This could include real-time data tracking, formative assessments, and portfolio-based evaluations that emphasize students' evolving understanding.

Additionally, future research should explore how different learning theories—such as enactivism, constructivism, and sociocultural approaches—align with the demands of technology-rich learning environments. Investigating how these theoretical models can be adapted to better reflect the realities of computational learning will provide educators with more robust frameworks for instructional design and pedagogy. Research that examines how teachers adopt and adapt pedagogical strategies in response to student interactions with programming tools could also offer insights into effective professional development.

As educational technologies continue to evolve, research must also keep pace with how these tools impact both teaching and learning. Studies should investigate the implications of emerging technologies such as artificial intelligence, virtual reality, and advanced programming environments on CT and mathematics education. Understanding how these tools reshape

learning experiences will be critical for developing relevant instructional strategies that support student engagement and learning outcomes.

In general, this study highlights the need for future research that bridges theoretical models with practical applications, focusing on the dynamic and interactive nature of learning in technology-rich environments. By exploring these areas, scholars can contribute to a more integrated understanding of how computational thinking and mathematics coalesce, ultimately guiding educators and policymakers in the ever-evolving landscape of education.

Conclusion

This study investigated the integration of CT within mathematics education, focusing on how learners engage with programming environments through the lens of enactivism. The findings revealed that learning in such environments is deeply shaped by the continuous interaction between learners and their environment. By highlighting the co-action between learners and their environments, this study contributes to a growing body of research that seeks to understand the evolving nature of education in the 21st century, where computational and mathematical thinking are becoming increasingly intertwined.

The critique of established learning theories, such as behaviourism and constructivism, emphasized the need to refine these frameworks to better reflect the complexity of learning in modern, technology-rich environments. While constructivism has been instrumental in promoting active learning, it remains limited by its focus on a pre-given world and its one-directional nature. The enactivist approach, by contrast, provides a more comprehensive view of how learners shape and are shaped by their surroundings, offering a bidirectional model of cognition that reflects real-world learning more accurately.

In practice, the study's findings underscore the importance of adaptive, iterative approaches to teaching CT and mathematics. They call for the development of flexible

pedagogical strategies and assessment tools that capture the non-linear, context-dependent nature of student learning. Moreover, the study reinforces the need for professional development that equips educators to navigate these complex learning environments effectively.

As educational technologies continue to evolve, it is essential for both research and practice to stay attuned to these changes. The study highlights the value of enactivism as a framework for understanding the dynamic interplay between learners and their technological environment, suggesting that future research and educational practices should embrace this fluidity. Ultimately, this research offers insights into how we can better support students in developing the computational and mathematical skills necessary for success in a rapidly changing world.

References

- Abrahamson, D., Nathan, M. J., Williams-Pierce, C., Walkington, C., Ottmar, E. R., Soto, H., & Alibali, M. W. (2020, August). The future of embodied design for mathematics teaching and learning. *Frontiers in Education*, 5, Article 147.
<https://doi.org/10.3389/feduc.2020.00147>
- Abrahamson, D., & Sánchez-García, R. (2016). Learning is moving in new ways: The ecological dynamics of mathematics education. *Journal of the Learning Sciences*, 25(2), 203–239.
<https://doi.org/10.1080/10508406.2016.1143370>
- Abrahamson, D., & Trninic, D. (2015). Bringing forth mathematical concepts: Signifying sensorimotor enactment in fields of promoted action. *ZDM*, 47(2), 295–306.
<http://doi.org/10.1007/s11858-014-0620-0>
- Ackermann, E. (2001). *Piaget's constructivism, Papert's constructionism: What's the difference?*
https://learning.media.mit.edu/content/publications/EA.Piaget%20_%20Papert.pdf
- Ainley, J., Pratt, D., & Hansen, A. (2006). Connecting engagement and focus in pedagogic task design. *British Educational Research Journal*, 32(1), 23–38.
<https://doi.org/10.1080/01411920500401971>
- Almazroa, H., & Alotaibi, W. (2023). Teaching 21st century skills: Understanding the depth and width of the challenges to shape proactive teacher education programmes. *Sustainability*, 15(9), Article 7365. <https://doi.org/10.3390/su15097365>
- Alqahtani, M. M., & Powell, A. B. (2016). Instrumental appropriation of a collaborative, dynamic-geometry environment and geometrical understanding. *International Journal of Education in Mathematics, Science and Technology*, 4(2), 72–83.
<http://doi.org/10.18404/ijemst.38054>

- Aydin, E. (2005). The use of computers in mathematics education: A paradigm shift from “computer assisted instruction” towards “student Programming.” *Turkish Online Journal of Educational Technology*, 4(2), 27–34. <https://files.eric.ed.gov/fulltext/EJ1102460.pdf>
- Bandura, A. (1997). *Self-efficacy: The exercise of control*. Freeman.
- Bandura, A. (2001). Social cognitive theory: An agentic perspective. *Annual Review of Psychology*, 52, 1–26. <https://doi.org/10.1146/annurev.psych.52.1.1>
- Barr, D., Harrison, J., & Conery, L. (2011). Computational thinking: A digital age skill for everyone. *Learning & Leading with Technology*, 38(6), 20–23. <https://files.eric.ed.gov/fulltext/EJ918910.pdf>
- Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads*, 2(1), 48–54. <http://doi.org/10.1145/1929887.1929905>
- Benton, L., Hoyles, C., Kalas, I., & Noss, R. (2017). Bridging primary programming and mathematics: Some findings of design research in England. *Digital Experiences in Mathematics Education*, 3(2), 115–138. <https://doi.org/10.1007/s40751-017-0028-x>
- Benton, L., Hoyles, C., Noss, R., & Kalas, I. (2016). Building mathematical knowledge with programming: Insights from the ScratchMaths project. In *Proceedings of Constructionism 2016* (pp. 25–32). <https://discovery.ucl.ac.uk/id/eprint/1475523/>
- Berland, M., & Wilensky, U. (2015). Comparing virtual and physical robotics environments for supporting complex systems and computational thinking. *Journal of Science Education and Technology*, 24(5), 628–647. <https://doi.org/10.1007/s10956-015-9552-x>
- Blikstein, P., Abrahamson, D., & Wilensky, U. (2006). *Agent-based modeling: Case studies in participatory simulations and modeling for learning*. Northwestern University. https://ccl.northwestern.edu/2008/Blikstein%20CetAI_ClassroomComplex.pdf

- Boaler, J. (2016). *Mathematical mindsets: Unleashing students' potential through creative math, inspiring messages, and innovative teaching*. Jossey-Bass.
- Bourdieu, P. (1977). *Outline of a theory of practice* [Esquisse d'une théorie de la pratique]. Transl. by Richard Nice. Cambridge, UK: Cambridge University Press.
- Bourdieu, P. (1990). *The logic of practice*. Stanford, CA: Stanford University Press.
- Bower, M., & Falkner, K. (2015, January). Computational thinking, the notional machine, pre-service teachers, and research opportunities. In *Proceedings of the 17th Australasian Computing Education Conference* (pp. 37–46). <https://crpit.scem.western-sydney.edu.au/confpapers/CRPITV160Bower.pdf>
- Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology, 3*(2), 77–101. <https://doi.org/10.1191/1478088706qp063oa>
- Braun, V., & Clarke, V. (2012). Thematic analysis. In H. Cooper, P. M. Camic, D. L. Long, A. T. Panter, D. Rindskopf, & K. J. Sher (Eds.), *APA handbook of research methods in psychology* (Vol. 2, pp. 57–71). APA. <https://doi.org/10.1037/13620-004>
- Braun, V., & Clarke, V. (2013). *Successful qualitative research: A practical guide for beginners*. SAGE.
- Braun, V., & Clarke, V. (2020). One size fits all? What counts as quality practice in (reflexive) thematic analysis? *Qualitative Research in Psychology, 18*(3), 328–352. <https://doi.org/10.1080/14780887.2020.1769238>
- Brennan, K., & Resnick, M. (2012, April). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American Educational Research Association*. <https://scratched.gse.harvard.edu/ct/files/AERA2012.pdf>

- Bruner, J. (1997). Celebrating divergence: Piaget and Vygotsky. *Human Development*, 40(2), 63–73. <https://doi.org/10.1159/000278705>
- Bundy, A. (2007). Computational thinking is pervasive. *Journal of Scientific and Practical Computing*, 1(2), 67–69. <http://www.spclab.com/publisher/journals/Vol1No2/N1.pdf>
- Burton, L. (1984). *Mathematical thinking: The struggle for meaning*. *Journal for Research in Mathematics Education*, 15(1), 35–49. <https://doi.org/10.2307/748986>
- Buteau, C., Gadanidis, G., Lovric, M., & Mueller, E. (2017). Computational thinking and mathematics curriculum. In S. Osterle, D. Allan, & J. Holm (Eds.), *Proceedings of the 2016 annual meeting of the Canadian Mathematics Education Study Group Conference* (pp. 119–136). Queen’s University.
- Byrne, D. (2022). A worked example of Braun and Clarke’s approach to reflexive thematic analysis. *Quality & Quantity*, 56(3), 1391–1412. <https://doi.org/10.1007/s11135-021-01182-y>
- Cabrera, L. (2019). Teacher preconceptions of computational thinking: A systematic literature review. *Journal of Technology and Teacher Education*, 27(3), 305–333. <http://doi.org/10.70725/676916nohbuf>
- Carman, T. (2020). *Merleau-Ponty* (2nd ed.). Routledge. <https://doi.org/10.4324/9781315537542>
- Catlin, D., & Blamires, M. (2018). Designing robots for special needs education. *Technology, Knowledge and Learning*, 24(2), 291–313. <https://doi.org/10.1007/s10758-018-9378-8>
- Cho, M.-H., & Heron, M. L. (2015). Self-regulated learning: the role of motivation, emotion, and use of learning strategies in students’ learning experiences in a self-paced online mathematics course. *Distance Education*, 36(1), 80–99. <https://doi.org/10.1080/01587919.2015.1019963>
- Clark, R. E. (2004). The classical origins of Pavlov’s conditioning. *Integrative Physiological & Behavioral Science*, 39(4), 279–294. <https://doi.org/10.1007/BF02734167>

- Clark, V. L. P., & Creswell, J. W. (2014). *Understanding research: A consumer's guide*. Pearson.
- Clements, D. H., & Sarama, J. (1997). Research on Logo: a decade of progress. *Computers in the Schools, 14*, 9–46. https://doi.org/10.1300/J025v14n01_02
- Cole, M., & Wertsch, J. V. (1996). Beyond the individual-social antinomy in discussions of Piaget and Vygotsky. *Human Development, 39*(5), 250–256.
<https://doi.org/10.1159/000278475>
- Confrey, J. (1999). Voice, perspective, bias and stance: Applying and modifying Piagetian theory in mathematics education. In L. Burton (Ed.), *Learning mathematics: From hierarchies to networks* (pp. 3–20). Falmer Press.
- Coronata, C., & Alsina, Á. (2014). Evaluation of the mathematical processes in the practices of teaching and learning in childhood education. *Procedia-Social and Behavioral Sciences, 141*, 1320–1323. <https://doi.org/10.1016/j.sbspro.2014.05.227>
- Creswell, J. W., & Poth, C. N. (2016). *Qualitative inquiry and research design: Choosing among five approaches*. SAGE.
- D'Amico, A., & Guastella, D. (2018). Robotics construction kits: From “objects to think with” to “objects to think and to emote with.” *Future Internet, 10*(2), Article 21.
<https://doi.org/10.3390/fi10020021>
- Davis, B., & Francis, K. (2023). Enactivism. In *Discourses on Learning in Education*.
<https://learningdiscourses.com/alphabetical-index/#starts-with-E>
- Davis, B., & Simmt, E. (2003). Understanding learning systems: Mathematics education and complexity science. *Journal for Research in Mathematics Education, 34*(2), 137–167.
<https://doi.org/10.2307/30034903>

- Denning, P. J. (2009). The profession of IT: Beyond computational thinking. *Communications of the ACM*, 52(6), 28–30. <https://doi.org/10.1145/1516046.1516054>
- Denning, P., & Freeman, P. (2009). Computing's paradigm. *Communications of the ACM*, 52(12), 28–30. <https://doi.org/10.1145/1610252.1610265>
- Department of Education UK. (2015). *Statutory guidance: National curriculum in England: Computing programmes of study*. <https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study>
- diSessa, A. A. (2001). *Changing minds: Computers, learning, and literacy*. MIT Press.
- Dishon, G., & Kafai, Y. (2020). Making more of games: Cultivating perspective-taking through game design. *Computers & Education*, 148, Article 103810. <https://doi.org/10.1016/j.compedu.2020.103810>
- Drijvers, P. (2001). The concept of parameter in a computer algebra environment. In M. Van den Heuvel-Panhuizen (Ed.), *Proceedings of PME 25* (Vol. 2, pp. 385–392). Freudenthal Institute.
- Drijvers, P. (2015). Digital technology in mathematics education: Why it works (or doesn't). In S. Cho (Ed.), *Selected regular lectures from the 12th international congress on mathematical education* (pp. 135–151). Springer.
- Drijvers, P., Kodde-Buitenhuis, H., & Doorman, M. (2019). Assessing mathematical thinking as part of curriculum reform in the Netherlands. *Educational Studies in Mathematics*, 102, 435–456. <https://doi.org/10.1007/s10649-019-09905-7>
- Dweck, C. S. (2015). Carol Dweck revisits the “growth mindset.” *Education Week*, 35(5), 20–24.

- Dworkin, S. L. (2012). Sample size policy for qualitative studies using in-depth interviews. *Archives of Sexual Behavior, 41*(6), 1319–1320. <https://doi.org/10.1007/s10508-012-0016-6>
- Eberle, T. (2014). Phenomenology as a research method. In U. Flick (Ed.), *The SAGE handbook of qualitative data analysis* (pp. 184–202). SAGE.
<https://doi.org/10.4135/9781446282243.n13>
- English, L. D., & Gainsburg, J. (2015). Problem solving in a 21st-century mathematics curriculum. In L. D. English & D. Kirschner (Eds.), *Handbook of international research in mathematics education* (pp. 313–335). Routledge.
- Fadel, C., Bialik, M., & Trilling, B. (2015). *Four-dimensional education: The competencies learners need to succeed*. Center for Curriculum Redesign.
https://www.researchgate.net/publication/318430582_Four-Dimensional_Education
- Feurzeig, W., & Papert, S. A. (2011). Programming-languages as a conceptual framework for teaching mathematics. *Interactive Learning Environments, 19*(5), 487–501.
<https://doi.org/10.1080/10494820903520040>
- Findell, B., Swafford, J., & Kilpatrick, J. (Eds.). (2001). *Adding it up: Helping children learn mathematics*. National Academies Press.
- Forbes, W. A., Mgombelo, J., Gannon, S., & Buteau, C. (2019, February). Undergraduate students' mindsets in a computer programming mathematical learning environment. In *Eleventh Congress of the European Society for Research in Mathematics Education* (No. 6). Freudenthal Group; Freudenthal Institute; ERME.

- Forsström, S. E., & Kaufmann, O. T. (2018). A literature review exploring the use of programming in mathematics education. *International Journal of Learning, Teaching and Educational Research*, 17(12), 18–32. <https://doi.org/10.26803/ijlter.17.12.2>
- Fosnot, C. T. (2005). *Constructivism: Theory, perspectives, and practice*. Teachers College Press.
- Fox-Turnbull, W. (2009). Stimulated recall using autophotography-A method for investigating technology education. In A. Bekker, I. Mottier, & M. J. de Vries (Eds.), *Proceedings of the PATT-22 Conference: Strengthening the position of technology education in the curriculum* (pp. 204–217). International Technology and Engineering Educators Association.
- Freire, P. (1996). *Pedagogy of the oppressed* (Rev. ed.). Continuum.
- Gadanidis, G., Brodie, I., Minniti, L., & Silver, B. (2017). Computer coding in the K–8 mathematics curriculum. *What Works: Research Into Practice*. <https://www.onted.ca/monographs/what-works/computer-coding-in-the-k-8-mathematics-curriculum>
- Gadanidis, G., Cendros, R., Floyd, L., & Namukasa, I. (2017). Computational thinking in mathematics teacher education. *Contemporary Issues in Technology and Teacher Education*, 17(4), 458–477. <https://citejournal.org/volume-17/issue-4-17/mathematics/computational-thinking-in-mathematics-teacher-education>
- Garzon, J., & Bautista, J. (2018). Virtual Algebra Tiles: A pedagogical tool to teach and learn algebra through geometry. *Journal of Computer Assisted Learning*, 34(6), 876–883. <https://doi.org/10.1111/jcal.12296>

- Gibson, J. J. (1986). *The ecological approach to visual perception*. Lawrence Erlbaum Associates.
- Giddens, A. (1984). *The constitution of society: Outline of the theory of structuration*. Malden, MA: Polity Press.
- Girvan, C., & Savage, T. (2019). Virtual worlds: A new environment for constructionist learning. *Computers in Human Behavior, 99*, 396–414.
<https://doi.org/10.1016/j.chb.2019.03.017>
- Goos, M., & Kaya, S. (2020). Understanding and promoting students' mathematical thinking: A review of research published in ESM. *Educational Studies in Mathematics, 103*(1), 7–25.
<https://doi.org/10.1007/s10649-019-09921-7>
- Greiff, S., Wüstenberg, S., & Funke, J. (2012). Dynamic Problem Solving: A New Assessment Perspective. *Applied Psychological Measurement, 36*(3), 189–213.
<https://doi.org/10.1177/0146621612439620>
- Groenewald, T. (2004). A phenomenological research design illustrated. *International Journal of Qualitative Methods, 3*(1), 42–55. <https://doi.org/10.1177/160940690400300104>
- Grootenboer, P. (2009). Rich mathematical tasks in the Maths in the Kimberley (MITK) project. In R. H., B. Bicknell & T. Burgess (Eds.), *Crossing divides* (pp. 696–699). MERGA.
- Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher, 42*(1), 38–43. <https://doi.org/10.3102/0013189X12463051>
- Guest, G., MacQueen, K. M., & Namey, E. E. (2011). *Applied thematic analysis*. SAGE.
- Gueudet, G., Buteau, C., Mesa, V., & Misfeldt, M. (2014). Instrumental and documentational approaches: From technology use to documentation systems in university mathematics education. *Research in Mathematics Education, 16*(2), 139–155.
<https://doi.org/10.1080/14794802.2014.918349>

- Gueudet, G., Buteau, C., Muller, E., Mgombelo, J., & Sacristán, A. (2020, September). Programming as an artefact: What do we learn about university students' activity? In T. Hausberger, M. Bosch, & F. Chellougui (Eds.), *Proceedings of the Third Conference of the International Network for Didactic Research in University Mathematics (INDRUM 2020)* (pp. 443–452). University of Carthage and INDRUM. <https://hal.archives-ouvertes.fr/hal-03113851/document>
- Gueudet, G., Buteau, C., Muller, E., Mgombelo, J., Sacristán, A. I., & Santacruz Rodriguez, M. (2022). Development and evolution of instrumented schemes: A case study of learning programming for mathematical investigations. *Educational Studies in Mathematics, 110*, 353–377. <https://doi.org/10.1007/s10649-021-10133-1>
- Gutstein, E. (2012). *Reading and writing the world with mathematics: Toward a pedagogy for social justice*. Routledge.
- Guzdial, M. (2008). Education: Paving the way for computational thinking. *Communications of the ACM, 51*(8), 25–27. <https://doi.org/10.1145/1378704.1378713>
- Guzdial, M. (2011, March 22). A definition of computational thinking from Jeannette Wing. *Computing Ed Research*. <http://computinged.wordpress.com/2011/03/22/a-definition-of-computational-thinking-from-jeanette-wing/>
- Hamilton, E. (2019). Computer-assisted instruction. *Salem Press encyclopedia*. Salem Press.
- Healy, L., & Kynigos, C. (2010). Charting the microworld territory over time: Design and construction in mathematics education. *ZDM, 42*(1), 63–76. <https://doi.org/10.1007/s11858-009-0193-5>
- Hegedus, S. J., & Moreno-Armella, L. (2010). Accommodating the instrumental genesis framework within dynamic technological environments. *For the Learning of Mathematics, 30*(1), 26–31. <http://doi.org/10.2307/20749435>

- Herbert, S. (2021). Overcoming challenges in assessing mathematical reasoning. *The Australian Journal of Teacher Education*, 46(8), 17–30. <https://doi.org/10.14221/ajte.2021v46n8.2>
- Heylighen, F. (2015). *Stigmergy as a universal coordination mechanism: Components, varieties and applications*. <http://doi.org/10.13140/RG.2.1.4479.5044>
- Heylighen, F. (2016). Stigmergy as a universal coordination mechanism I: Definition and components. *Cognitive Systems Research*, 38, 4–13. <https://doi.org/10.1016/j.cogsys.2015.12.002>
- 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(1), 48–69.
- Holbert, N., & Wilensky, U. (2019). Designing educational video games to be objects-to-think-with. *Journal of the Learning Sciences*, 28(1), 32–72. <https://doi.org/10.1080/10508406.2018.1487302>
- Hsu, T. C., Chang, S. C., & Hung, Y. T. (2018). How to learn and how to teach computational thinking: Suggestions based on a review of the literature. *Computers & Education*, 126, 296–310. <https://doi.org/10.1016/j.compedu.2018.07.004>
- Hutto, D. D., & Sánchez-García, R. (2015). Choking RECTified: Embodied expertise beyond Dreyfus. *Phenomenology and the Cognitive Sciences*, 14, 309–331. <https://doi.org/10.1007/s11097-014-9380-0>
- Hycner, R. H. (1985). Some guidelines for the phenomenological analysis of interview data. *Human studies*, 8(3), 279–303. <https://doi.org/10.1007/bf00142995>
- Inouye, K., Lee, S., & Oldac, Y. I. (2023). A systematic review of student agency in international higher education. *Higher Education*, 86(4), 891–911.

- Joshi, A., Darasawang, P., & Tepsuriwong, S. (2019). Exploring the role of computers in knowledge construction of young learners in a constructionist classroom. *Journal of Language Teaching and Research*, 10(6), 1197–1208.
<http://doi.org/10.17507/jltr.1006.07>
- Kafai, Y. B. (2012). *Minds in play: Computer game design as a context for children's learning*. Routledge.
- Kallio, H., Pietilä, A. M., Johnson, M., & Kangasniemi, M. (2016). Systematic methodological review: Developing a framework for a qualitative semi-structured interview guide. *Journal of Advanced Nursing*, 72(12), 2954–2965. <https://doi.org/10.1111/jan.13031>
- Kensit, D. A. (2000). Rogerian theory: A critique of the effectiveness of pure client-centred therapy. *Counselling Psychology Quarterly*, 13(4), 345–351.
<https://doi.org/10.1080/713658499>
- Kieren, T., Simmt, E., & Mgombelo, J. (1997). Occasioning understanding: Understanding occasioning. In *Proceedings of the 26th International Conference for Psychology of Mathematics Education—North American Chapter* (Vol. 2, pp. 627–634).
<https://files.eric.ed.gov/fulltext/ED383537.pdf>
- Knuth, D. E. (1985). Algorithmic thinking and mathematical thinking. *The American Mathematical Monthly*, 92(3), 170–181.
<https://doi.org/10.1080/00029890.1985.11971572>
- Kretchmar, J. (2019). Constructivism. *Salem Press encyclopedia*. Salem Press.
- Kurland, D. M., & Kurland, L. C. (1987). Computer applications in education: A historical overview. *Annual Review of Computer Science*, 2(1), 317–358.
<http://doi.org/10.1146/annurev.cs.02.060187.001533>

- Kus, M., & Cakiroglu, E. (2022). Mathematics in the informal setting of an art studio: Students' visuospatial thinking processes in a studio thinking-based environment. *Educational Studies in Mathematics, 110*, 545–571. <https://doi.org/10.1007/s10649-022-10142-81-27>.
- Kynigos, C. (2015). Constructionism: Theory of learning or theory of design? In S. Cho (Ed.), *Selected regular lectures from the 12th International Congress on Mathematical Education* (pp. 417–438). Springer.
- Lee, I., Grover, S., Martin, F., Pillai, S., & Malyn-Smith, J. (2020). Computational thinking from a disciplinary perspective: Integrating computational thinking in K–12 science, technology, engineering, and mathematics education. *Journal of Science Education and Technology, 29*(1), 1–8. <https://doi.org/10.1007/s10956-019-09803-w>
- Li, Q., & Ma, X. (2010). A meta-analysis of the effects of computer technology on school students' mathematics learning. *Educational Psychology Review, 22*(3), 215–243. <https://doi.org/10.1007/s10648-010-9125-8>
- Liao, Y. K. C., & Bright, G. W. (1991). Effects of computer programming on cognitive outcomes: A meta-analysis. *Journal of Educational Computing Research, 7*(3), 251–268. <http://doi.org/10.2190/E53G-HH8K-AJRR-K69M>
- Ling, A. N. B., & Mahmud, M. S. (2023). Challenges of teachers when teaching sentence-based mathematics problem-solving skills. *Frontiers in Psychology, 13*. <https://doi.org/10.3389/fpsyg.2022.1074202>
- Lippi, S. (2016). Hallucination as theorized by Merleau-Ponty and Lacan: How perception, reality and the real are interconnected. *Chiasmi International, 18*, 91–103. <https://doi.org/10.5840/chiasmi20161811>

- Lourenço, O. (2012). Piaget and Vygotsky: Many resemblances, and a crucial difference. *New Ideas in Psychology*, 30(3), 281–295. <https://doi.org/10.1016/j.newideapsych.2011.12.006>
- Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*, 41, 51–61. <https://doi.org/10.1016/j.chb.2014.09.012>
- Lyle, J. (2003). Stimulated recall: A report on its use in naturalistic research. *British Educational Research Journal*, 29(6), 861–878. <https://doi.org/10.1080/0141192032000137349>
- Lyon, J. A., & J. Magana, A. (2020). Computational thinking in higher education: A review of the literature. *Computer Applications in Engineering Education*, 28(5), 1174–1189. <https://doi.org/10.1002/cae.22295>
- Martin, L. C., & Towers, J. (2009). Improvisational coactions and the growth of collective mathematical understanding. *Research in Mathematics Education*, 11(1), 1–19. <http://doi.org/10.1080/14794800902732191>
- Mason, J., Burton, L., and Stacey, K. (1982). *Thinking mathematically*. Addison-Wesley.
- Mason, J., Burton, L., and Stacey, K. (2010). *Thinking mathematically*. Pearson.
- Mason, S. L., & Rich, P. J. (2019). Preparing elementary school teachers to teach computing, coding, and computational thinking. *Contemporary Issues in Technology and Teacher Education*, 19(4), 790–824.
- Maturana, H. R. (2002) Autopoiesis, structural coupling and cognition: A history of these and other notions in the biology of cognition. *Cybernetics & Human Knowing*, 9(3–4), 5–34.
- Maturana, H. R., & Varela, F. J. (1987). *The tree of knowledge: The biological roots of human understanding*. New Science Library/Shambhala.

- Mayo, P. (1999). *Gramsci, Freire (1996) and adult education: Possibilities for transformative action*. Palgrave Macmillan.
- McCarthy-Curvin, A., Buddo, C., & George, L. (2020). TEST problem solving within the mathematics classroom: Challenges and recommendations. *Journal of Education & Development in the Caribbean, 19*(2). <http://doi.org/10.46425/j319021548>
- Merleau-Ponty, M. (1962). *Phenomenology of perception* (D. A. Landes, Trans.). Routledge and Kegan Paul.
- Meyer, D. (2015). Missing the promise of mathematical modeling. *The Mathematics Teacher, 108*(8), 578–583.
- Mgombelo, J. (2017). Collective learning: Re-thinking the environment, artifacts. *Topic Group Session*. CMESG, McGill, Montreal, QC.
- Miles, M. B., & Huberman, A. M. (1994). *Qualitative data analysis: An expanded sourcebook*. SAGE.
- Misfeldt, M., & Ejsing-Duun, S. (2015). Learning mathematics through programming: An instrumental approach to potentials and pitfalls. In K. Krainer & N. Vondrová (Eds.), *Proceedings of the Ninth Congress of the European Society for Research in Mathematics Education* (pp. 2524–2530). Charles University in Prague, Faculty of Education and ERME.
- Montuori, C., Gambarota, F., Altoe, G., & Arfe, B. (2024). The cognitive effects of computational thinking: A systematic review and meta-analytic study. *Computers and Education, 210*. <https://doi.org/10.1016/j.compedu.2023.104961>
- Moreland, J., & Cowie, B. (2016). Exploring the methods of auto-photography and photo-interviews: Children taking pictures of science and technology. *Waikato Journal of Education, 11*(1). <https://doi.org/10.15663/wje.v11i1.320>

- Moses, L., Rylak, D., Reader, T., Hertz, C., & Ogden, M. (2020). Educators' perspectives on supporting student agency. *Theory Into Practice*, 59(2), 213–222.
<https://doi.org/10.1080/00405841.2019.1705106>
- Muller, E., Buteau, C., Ralph, B., & Mgombelo, J. (2009). Learning mathematics through the design and implementation of exploratory and learning objects. *International Journal for Technology in Mathematics Education*, 16(2), 63–74.
- Nardo, A. (2021). Exploring a Vygotskian Theory of Education and Its Evolutionary Foundations. *Educational Theory*, 71(3), 331–352. <https://doi.org/10.1111/edth.12485>
- National Council of Teachers of Mathematics. (2000). *Principles and standards for school mathematics*. <https://www.nctm.org/Standards-and-Positions/Principles-and-Standards/>
- Nemirovsky, R., Kelton, M. L., & Rhodehamel, B. (2013). Playing mathematical instruments: Emerging perceptuomotor integration with an interactive mathematics exhibit. *Journal for Research in Mathematics Education*, 44(2), 372–415. <https://doi.org/10.5951/jresematheduc.44.2.0372>
- Noss, R. & Hoyles, C. (Eds.). (1992). *Learning mathematics and Logo*. MIT Press.
- Nova Scotia Department of Education and Early Childhood Development. (2020). *Mathematics primary guide*. <https://curriculum.novascotia.ca/english-programs/course/mathematics-primary>
- OECD. (2020). *Curriculum overload: A way forward*. https://www.oecd.org/en/publications/curriculum-overload_3081ceca-en.html
- OECD. (2022). *PISA 2022 Mathematics framework*. <https://pisa2022-maths.oecd.org/ca/index.html>
- Ontario Ministry of Education. (2020). *The Ontario curriculum Grades 1-8: Mathematics*. <https://www.dcp.edu.gov.on.ca/en/curriculum/elementary-mathematics>

- Ontario releases new math curriculum, includes computer coding. (2020, June 23). *Global News*.
<https://globalnews.ca/video/7097765/ontario-releases-new-math-curriculum-includes-computer-coding/>
- Palinkas, L. A., Horwitz, S. M., Green, C. A., Wisdom, J. P., Duan, N., & Hoagwood, K. (2015). Purposeful sampling for qualitative data collection and analysis in mixed method implementation research. *Administration and Policy in Mental Health and Mental Health Services Research*, 42(5), 533–544. <https://doi.org/10.1007/s10488-013-0528-y>
- Papadakis, S., & Kalogiannakis, M. (2019). Evaluating a course for teaching introductory programming with Scratch to pre-service kindergarten teachers. *International Journal of Technology Enhanced Learning*, 11(3), 231–246.
<http://doi.org/10.1504/IJTEL.2019.10020447>
- Papavlasopoulou, S., Giannakos, M. N., & Jaccheri, L. (2019). Exploring children’s learning experience in constructionism-based coding activities through design-based research. *Computers in Human Behavior*, 99, 415–427. <https://doi.org/10.1016/j.chb.2019.01.008>
- Papert, S. (1972). Teaching children to be mathematicians versus teaching about mathematics. *International Journal of Mathematical Education in Science and Technology*, 3(3), 249–262. <https://doi.org/10.1080/0020739700030306>
- Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books.
- Papert, S. (1993). *The children’s machine: Rethinking school in the age of the computer*. Basic Books.
- Papert, S., & Harel, I. (1991). Situating constructionism. *Constructionism*, 36(2), 1–11.
- Parunak, H. V. D. (2006). A survey of environments and mechanisms for human-human stigmergy. In D. Weyns, H. V. D. Parunak, & F. Michel (Eds.), *Environments for multi-agent systems II* (pp. 163–186). Springer. https://doi.org/10.1007/11678809_10

- Patton, M. Q. (2014). *Qualitative research & evaluation methods: Integrating theory and practice*. SAGE.
- Pavlov, I. P. (1927). *Conditioned reflexes*. Oxford University Press.
- Pea, R. D. (1987). Cognitive technologies for mathematics education. In A. Schoenfeld (Ed.), *Cognitive science and mathematics education* (pp. 89–122). Erlbaum.
- Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New Ideas in Psychology*, 2(2), 137–168. [https://doi.org/10.1016/0732-118X\(84\)90018-7](https://doi.org/10.1016/0732-118X(84)90018-7)
- Pedaste, M., Mäeots, M., Siiman, L. A., de Jong, T., van Riesen, S. A. N., Kamp, E. T., Manoli, C. C., Zacharia, Z. C., & Tsourlidaki, E. (2015). Phases of inquiry-based learning: Definitions and the inquiry cycle. *Educational Research Review*, 14, 47–61. <https://doi.org/10.1016/j.edurev.2015.02.003>
- Piaget, J. (1957). *Construction of reality in the child*. Routledge & Kegan Paul.
- Piggott, J. (2011). Rich tasks and contexts. *nrich*. <https://nrich.maths.org/articles/rich-tasks-and-contexts>
- Pickering, A. (2013). Being in an environment: A performative perspective. *Natures Sciences Sociétés (Montrouge)*, 21(1), 77–83. <https://doi.org/10.1051/nss/2013067>
- Prime Minister Justin Trudeau talks importance of coding, algorithms. (2016, January 14). *GlobalNews.ca*. <https://globalnews.ca/video/2453986/prime-minister-justin-trudeau-talks-importance-of-coding-algorithms>
- Proulx, J. (2006). Constructivism: A re-equilibration and clarification of the concepts, and some potential implications for teaching and pedagogy. *Radical Pedagogy*, 8(1). https://radicalpedagogy.icaap.org/content/issue8_1/proulx.html

- Rabardel, P. (2002). *People and technology: A cognitive approach to contemporary instruments* (H. Wood, Trans.). Université Paris.
- Rabardel, P., & Beguin, P. (2005). Instrument mediated activity: from subject development to anthropocentric design. *Theoretical Issues in Ergonomics Science*, 6(5), 429–461.
<https://doi.org/10.1080/14639220500078179>
- Reid, D. A., & Mgombelo, J. (2015). Survey of key concepts in enactivist theory and methodology. *ZDM*, 47(2), 171–183. <http://doi.org/10.1007/s11858-014-0634-7>
- Rich, K. M., Yadav, A., & Schwarz, C. V. (2019). Computational thinking, mathematics, and science: Elementary teachers' perspectives on integration. *Journal of Technology and Teacher Education*, 27(2), 165–205. <http://doi.org/10.70725/303733vqlleui>
- The Royal Society. (2012). *Shut down or restart? The way forward for computing in UK schools*.
<https://royalsociety.org/~media/education/computing-in-schools/2012-01-12-computing-in-schools.pdf>
- Saad, A., & Zainudin, S. (2022). A review of Project-Based Learning (PBL) and Computational Thinking (CT) in teaching and learning. *Learning and Motivation*, 78, Article 101802.
<https://doi.org/10.1016/j.lmot.2022.101802>
- Scherer, R., Siddiq, F., & Sánchez Viveros, B. (2019). The cognitive benefits of learning computer programming: A meta-analysis of transfer effects. *Journal of Educational Psychology*, 111(5), 764–792. <https://doi.org/10.1037/edu0000314>
- Scibner, C. (1986). Thinking in action: Some characteristics of practical thought. In R. J. Sternberg & R. K. Wagner (Eds.), *Practical intelligence: Nature and origins of competence in the everyday world* (pp. 13–30). Cambridge University Press.

- Selby, C., & Woollard, J. (2013). *Computational thinking: The developing definition*. University of Southampton. <https://eprints.soton.ac.uk/356481>
- Shulman, L. S. (1986). Those who understand: Knowledge growth in teaching. *Educational Researcher*, 15(2), 4–14. <https://doi.org/10.3102/0013189X015002004>
- Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, 22, 142–158. <https://doi.org/10.1016/j.edurev.2017.09.003>
- Shvarts, A., Alberto, R., Bakker, A., Doorman, M., & Drijvers, P. (2021). Embodied instrumentation in learning mathematics as the genesis of a body-artifact functional system. *Educational Studies in Mathematics*, 107, 447–469. <https://doi.org/10.1007/s10649-021-10053-0>
- Skinner, B. F. (1957). *Verbal behavior*. Appleton-Century-Crofts.
- Smith, T. (2019). Paulo Freire (1996). *Salem Press encyclopedia*. Salem Press.
- Stager, G. (2016). Seymour Papert (1928-2016). *Nature*, 537, Article 308. <https://doi.org/10.1038/537308a>
- Staples, M. E. (2008). Supporting whole-class collaborative inquiry in a secondary mathematics classroom. *Cognition and Instruction*, 26(4), 379–426. <https://doi.org/10.1080/07370000802394861>
- Stenalt, M. H. (2021). Researching student agency in digital education as if the social aspects matter: Students' experience of participatory dimensions of online peer assessment. *Assessment & Evaluation in Higher Education*, 46(4), 644–658. <https://doi.org/10.1080/02602938.2020.1798355>
- Stenalt, M. H., & Lassesen, B. (2022). Does student agency benefit student learning? A systematic review of higher education research. *Assessment & Evaluation in Higher Education*, 47(5), 653–669. <https://doi.org/10.1080/02602938.2021.1967874>

- Strauss, A., & Corbin, J. (1998). *Basics of qualitative research: Techniques and procedures for developing grounded theory* (2nd ed.). SAGE.
- Sung, W., Ahn, J. H., Kai, S. M., & Black, J. (2017, March). Effective planning strategy in robotics education: An embodied approach. In *Society for Information Technology & Teacher Education international conference* (pp. 1065–1071). AACE.
- Tall, D. (2002). *Advanced mathematical thinking*. Kluwer.
- Thagard, P. (2005). *Mind: Introduction to cognitive science*. MIT Press.
- Thorndike, E. L. (1927). The law of effect. *The American Journal of Psychology*, *39*(1/4), 212–222. <https://doi.org/10.2307/1415413>
- Tooke, D. J. (2001). Introduction. *Computers in the Schools*, *17*, 1–7. https://doi.org/10.1300/J025v17n01_01
- Toom, A. (2017). Teachers' professional and pedagogical competencies: A complex divide between teacher work, teacher knowledge, and teacher education. In D. J. Clandinin & J. Husu (Eds.). *The SAGE handbook of research on teacher education* (Vol. 2, pp. 803–819). SAGE.
- Towers, J., & Davis, B. (2002). Structuring occasions. *Educational Studies in Mathematics*, *49*(3), 313–340. <https://doi.org/10.1023/A:1020245320040>
- Towers, J., & Martin, L. C. (2015). Enactivism and the study of collectivity. *ZDM*, *47*(2), 247–256. <https://doi.org/10.1007/s11858-014-0643-6>
- Tsarava, K., Moeller, K., Román-González, M., Golle, J., Leifheit, L., Butz, M. V., & Ninaus, M. (2022). A cognitive definition of computational thinking in primary education. *Computers and Education*, *179*. <https://doi.org/10.1016/j.compedu.2021.104425>

- Turkle, S., & Papert, S. (1990). Epistemological pluralism: Styles and voices within the computer culture. *Signs: Journal of Women in Culture and Society*, 16(1), 128–157.
<https://doi.org/10.1086/494648>
- Vale, I., & Barbosa, A. (2023). Active learning strategies for an effective mathematics teaching and learning. *European Journal of Science and Mathematics Education*, 11(3), 573–588.
<https://doi.org/10.30935/scimath/13135>
- Varela, F. J. (1992). Autopoiesis and a biology of intentionality. In *Proceedings of the workshop “Autopoiesis and Perception”* (pp. 4–14). Dublin City University.
- Varela, F. J., & Goguen, J. A. (1978). The arithmetic of closure. *Journal of Cybernetics*, 8(3–4), 291–324. <https://doi.org/10.1080/01969727808927587>
- Varela, F. J., Thompson, E., & Rosch, E. (2016). *The embodied mind, revised edition: Cognitive science and human experience*. MIT Press.
- Vaughn, M., Jang, B. G., Sotirovska, V., & Cooper-Novack, G. (2020). Student agency in literacy: A systematic review of the literature. *Reading Psychology*, 41(7), 712–734.
<http://doi.org/10.1080/02702711.2020.1783142>
- Vergnaud, G. (2009). The theory of conceptual fields. *Human Development*, 52(2), 83–94.
<https://doi.org/10.1159/000202727>
- Vergnaud, G. (2013). Conceptual development and learning. *QURRICULUM - Revista De Teoría, Investigación Y Práctica Educativa*, 26, 39–59.
<https://www.ull.es/revistas/index.php/qurriculum/article/view/65>
- Verillon, P., & Rabardel P. (1995). Cognition and artifacts: A contribution to the study of thought in relation to instrument activity. *European Journal of Psychology in Education*, 9(3), 77–101. <https://doi.org/10.1007/BF03172796>

- Vygotsky, L. (1978a). Interaction between learning and development. *Readings on the Development of Children*, 23(3), 34–41.
- Vygotsky, L. S. (1978b). *Mind in society: Development of higher psychological processes* (M. Cole, V. Jolm-Steiner, S. Scribner, & E. Souberman, Eds.). Harvard University Press.
<https://doi.org/10.2307/j.ctvjf9vz4>
- Wang, V. C. X. (2013). Traditional teaching or innovative teaching via technology? *Learning & Performance Quarterly*, 2(1), 1–13.
- Watson, J. B. (1913). Psychology as the behaviorist views it. *Psychological Review*, 20(2), 158–177. <https://doi.org/10.1037/h0074428>
- Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology*, 25(1), 127–147. <https://doi.org/10.1007/s10956-015-9581-5>
- Weintrop, D., & Wilensky, U. (2015, June). To block or not to block, that is the question: Students' perceptions of blocks-based programming. In *Proceedings of the 14th international conference on interaction design and children* (pp. 199–208).
<https://doi.org/10.1145/2771839.2771860>
- Wells, G. (2007). Semiotic mediation, dialogue and the construction of knowledge. *Human Development*, 50(5), 244–274. <http://doi.org/10.1159/000106414>
- Wheeler, M. (2011). Martin Heidegger. In E. N. Zalta (Ed.), *The Stanford encyclopedia of philosophy*. <https://plato.stanford.edu/archives/fall2020/entries/heidegger/>
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33–35.
<https://doi.org/10.1145/1118178.1118215>

- Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A*, 366(1881), 3717–3725.
<https://doi.org/10.1098/rsta.2008.0118>
- Wing, J. M. (2010). *Computational thinking: What and why?*
<http://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf>
- Wing, J. M., & Stanzione, D. (2016). Progress in computational thinking, and expanding the HPC community. *Communications of the ACM*, 59(7), 10–11.
<https://doi.org/10.1145/2933410>
- Yadav, A., Gretter, S., Good, J., & McLean, T. (2017). Computational thinking in teacher education. In P. Rich & C. Hodges (Eds.), *Emerging research, practice, and policy on computational thinking* (pp. 205–220). Springer.
- Yadav, A., Krist, C., Good, J., & Caeli, E. N. (2018). Computational thinking in elementary classrooms: Measuring teacher understanding of computational ideas for teaching science. *Computer Science Education*, 28(4), 371–400.
<http://doi.org/10.1080/08993408.2018.1560550>
- Yushau, B., Bokhari, M. A., & Wessels, D. C. J. (2004). Computer aided learning of mathematics: Software evaluation. *Mathematics and Computer Education*, 38(2), 165–182.
- Zhang, L., & Nouri, J. (2019). A systematic review of learning computational thinking through Scratch in K-9. *Computers & Education*, 141, Article 103607.
<https://doi.org/10.1016/j.compedu.2019.103607>

Appendix A

Invitation Letter (Email)

Title of Study: Mathematics Learning in a Computer Programming Environment: Co-actional Phenomenon, Mathematical Significance and Computational Thinking

Principal Investigator: Wendy Ann Forbes, Ph.D. Candidate, Department of Education, Brock University. wfl7jp@brocku.ca

Faculty Supervisor: Dr. Joyce Mgombelo, Associate Professor, Department of Educational Studies, Brock University.

Dear (MICA student),

I am inviting you to participate in a research study entitled Mathematics Learning in a Computer Programming Environment: Co-actional Phenomenon, Mathematical Significance, and Computational Thinking. The purpose of this study is to explore the nature of interactions between learners and their programming environment as they use programming for mathematics learning. Specifically, I am interested in understanding what guides your actions as you program for mathematics learning.

As a participant, you will be asked to participate in a semi-structured one-on-one audio-recorded interview that will last approximately 45 minutes. The interview will be conducted virtually and will consist of questions about your final project to understand how you interact with the learning environment to design your exploratory object (EO) or learning object (LO). Your EO or LO and associated reflection will be collected and reviewed beforehand so that the interviewer can ask questions directly related to the corresponding assignment pieces. Your submitted EO or LO will also be used in the interview for clarity when referring to said documents. Submitting your EO is a requirement for participating in the study since investigating participants' actions from the codes in their EO is a critical part of the research study. The researcher will collect field notes to record her experiences, thinking, and things seen or heard during the interview. The audio-recorded interviews will later be transcribed.

If interested, you will need to complete by <date> the online form “Participation Form” on Google Forms at <link> to show your interest in being a participant in the research project. The project requires 8 students, at least 1 in each of the MICA courses, and the researchers will strive to choose a diverse group with respect to gender, program of study, and year of study.

If you agree to participate, you will receive financial compensation of \$50

There are no direct benefits to you as a participant. However, due to the nature of the interview, you may benefit indirectly through meaningful reflection that may increase your understanding of the course requirements and the potential of programming for mathematics in your future endeavor. Furthermore, the research may help to develop technological integration in

classrooms in general and, specifically, in STEM education. K-12 students and post-secondary students alike are expected to use computational thinking skills to design and create technology. Thus, computer coding or computer programming has become an essential tool for education at all levels around the globe. Therefore, educators need to understand how students interact with interactive learning tools to orchestrate a learning environment for meaningful learning in mathematics. As a student in a MICA course, you are ideally situated to give valuable firsthand accounts that will contribute to understanding how students develop mathematical and computational skills in such an environment.

There is no significant risk to participants. However, there may be some psychological risk in recounting negative experiences you may have had with the learning environment. Also, you may be worried that participation in the study will contribute to your final grade since you will be asked to submit a copy of an EO or LO with any corresponding reflection that you have done for the course. You may also have confidentiality concerns. To minimize the risks, you will be informed before the interview, that you can withdraw from the research at any time without giving a reason and without penalty. Also, before each interview, you will be notified that you can stop the interview at any time or choose not to answer a question, without withdrawing from the study if you feel uncomfortable.

Furthermore, you will be informed before each interview that your participation is not for evaluation but to understand the nature of your interaction with the learning environment in designing your EO or LO. Your involvement in the research is independent of participation in the course in the sense that data from the research does not contribute in any way to the course; I do not give feedback to the course instructor about participation or non-participation participants. All data will be kept confidential.

Data for the study will be kept confidential. Each interview will be stored using a unique code. Only pseudonyms, gender, program of study, and year of study maybe used in reporting and storing data. The name of participants and their e-mail contact will only be known to the Principal Investigator and the Principal Student Investigator, and only the said persons will have access to data. Each data, including transcript and audio-recorded interview, will be stored using pseudonyms. Furthermore, only pseudonyms will be used in reporting. All data about the research will be stored in the principal student investigator's password-protected computer in password-protected files. Only the gender of participants, year in the MICA course, and program of study will be identified.

Shortly after the interview, you will be given a copy of the transcribed interview to verify the accuracy of the conversation with the interviewer. At this point, you may add comments or clarify any within three days.

Participation in this study is voluntary. It is up to you to decide whether or not you participate in this research. Other than the investigators, your participation or non-participation will remain confidential to everyone, including instructors and teaching assistants. There will be no consequences to your future career or academic endeavors. You may withdraw from this study at any time without giving a reason by e-mailing me at wfl7jp@brocku.ca. There is no penalty or loss of any benefits due to you if you should withdraw. In addition, you may decline to answer any questions in the interview session without withdrawing from the study.

The results of this study will be included in my dissertation for my PH. D in Educational Studies and may be published in professional journals and presented at conferences. The study results will be reported in ways to maintain confidentiality. Feedback about this study will be provided to you through a one-page summary report that will be sent upon completion of this project.

To compensate you for your time you will receive \$50 via e-transfer after the interview. If a participant withdraws from the study before an interview, no payment will be made.

This study has been reviewed and received ethics clearance through Brock University's Research Ethics Board [No.xxxx]. You may contact me at the email address above if you need further information about this study. This study has been reviewed and received ethics clearance through the Research Ethics Board at Brock University [REB-21-276]. If you have any comments or concerns about your rights as a research participant, don't hesitate to get in touch with the Office of Research Ethics at (905) 688-5550 Ext. 3035, reb@brocku.ca.

Kind regards
Wendy Ann

Appendix B

Information-Consent Letter for Students

Date: Insert Date

Title of Study: Mathematics Learning in a Computer Programming Environment: Co-actional Phenomenon, Mathematical Significance and Computational Thinking

Principal Investigator:

Wendy Ann Forbes, Ph.D. Candidate
 Department of Educational Studies
 Brock University
 wf17jp@brocku.ca

Faculty Supervisor:

Dr. Joyce Mgombelo
 Department of Educational Studies
 Brock University
 (905) 688-5550 Ext. 5117
 jmgombelo@brocku.ca

INVITATION

You are invited to participate in a study that involves research. The purpose of this study is to explore the nature of interactions(co-action) between learners and their programming environment as they use programming for mathematics learning. Specifically, I am interested in understanding what guides your actions as you program for mathematics learning.

NATURE OF INVOLVEMENT

As a participant, you will be asked to participate in a semi-structured one-on-one audio-recorded interview that will last approximately 45 minutes. The interview will be conducted virtually and will consists of questions about your final learning object (LO) or exploratory object (EO) to understand how you interact with the learning environment to design your object. Your EO or LO and associated reflection will be collected and reviewed one to two weeks beforehand so that the interviewer can ask questions directly related to the corresponding assignment pieces. Your submitted EO or LO will also be used in the interview for clarity when referring to said documents. Submitting your EO is a requirement for participating in the study since investigating participants' actions from the codes in their EO is a critical part of the research study. The researcher will collect field notes to record her experiences, thinking, and things seen or heard during the interview. The audio-recorded interviews will later be transcribed. Furthermore, within two weeks after an interview, each participant will be asked to review an exact transcript of their interview for verification purposes (member check). All participants will be given a

corresponding two weeks to respond with any inconsistencies. After two weeks it will be assumed that transcribed data is correct.

COMPENSATION

All 8 participants from the three MICA courses will be asked to participate in an interview for 45 – 60 min and do a member check of the transcribed data. Therefore, to compensate participants for their time, each student will receive \$50 via e-transfer after their interview. If a participant withdraws from the study before interview, no payment will be made. All participants will receive the \$50 compensation as long as they participate in the interview process, even if they do not complete their interview or withdraw from the study during their interview. Participants do not need to give a reason for withdrawal to receive payment.

POTENTIAL BENEFITS AND RISK

There are no direct benefits to participants. However, due to the nature of the interview, participants may benefit indirectly through meaningful reflection that may increase their understanding of the course requirements and potential for their future endeavors. Furthermore, the research may help develop technological integration in classrooms in general and, specifically, in STEM education. K-12 students and post-secondary students alike are expected to use computational thinking skills to design and create technology. Thus, computer coding or computer programming has become an essential tool for education at all levels around the globe. Therefore, educators need to understand how students interact with interactive learning tools to orchestrate learning environments for meaningful learning in mathematics. As a student in the MICA course, you are ideally situated to give valuable firsthand accounts that will contribute to understanding how students develop mathematical and computational skills in such an environment.

There is no significant risk to participants. However, there may be some psychological risk in recounting negative experiences you may have had with the programming learning environment. Also, participants may wonder if participation in the study may contribute to their final grade since they will be asked to submit an EO or LO and the corresponding reflection. Participants may also have confidentiality concerns. To minimize the risks, participants will be informed before the interview that they can withdraw from the research at any time without giving a reason and without penalty. Also, before each interview, participants will be notified that they can stop the conversation or choose not to answer a question without withdrawing from the study if they feel uncomfortable.

Furthermore, participants will be informed before each interview that their participation is not for evaluation but to understand the nature of their interaction with the learning environment in designing their EO or LO. Participants' involvement in the research is independent of participation in the course. Data from the study does not contribute to the course; I do not give feedback to the course instructor about participation or non-participation participants. All data will be kept confidential.

CONFIDENTIALITY

Each interview will be stored using a unique code for confidentiality. Your identity will not be associated with the data collected nor be revealed at any time before or after the study. Only pseudonyms, gender, and program of study will be used in reporting and storing data. Anonymous quotations may be used for reporting. The name of participants will only be known to the Principal Student Investigator and the supervisor, and only the said persons will have access to data. All data about the research will be stored in my password-protected computer in password-protected files.

Shortly after the interview, participants will be shown a copy of the transcribed interview to verify the accuracy of the conversation with the interviewer. At this point, participants may add comments or clarify any within three days.

VOLUNTARY PARTICIPATION

Participation in this study is voluntary. It is up to you to decide whether or not you participate in this research. Other than the investigators, your participation or non-participation will remain confidential to everyone, including instructors and teaching assistants, and will have no consequences on your future career or academic endeavors. You may withdraw from this study at any time without giving a reason by e-mailing me at wfl7jp@brocku.ca. However, you cannot withdraw at advanced stages of data analysis or after information about the data is published in any form, i.e., via conference, journal, or dissertation. There is no penalty or loss of any benefits due to you if you should withdraw. In addition, you may decline to answer any questions in the interview session without withdrawing from the study. A one-page summary of the research results will be made and e-mailed to participants.

PUBLICATION OF RESULTS

The results of this study will be included in my dissertation for my PH. D in Educational Studies and may be published in professional journals and presented at conferences. The study results will be reported in ways to maintain confidentiality. Anonymous excerpts from data (transcription, reflection, or EO/LO) may be used in reports. That is, while gender, year of study, and program of study may be reported, only pseudonyms will be used in reporting. Feedback about this study will be provided to you through a one-page summary report that will be sent upon completion of this project.

CONTACT INFORMATION AND ETHICS CLEARANCE

You may contact me at the e-mail address above if you need further information about this study. This study has been reviewed and received ethics clearance through the Research Ethics Board at Brock University [REB-21-276]. If you have any comments or concerns about your rights as a research participant, don't hesitate to get in touch with the Office of Research Ethics at (905) 688-5550 Ext. 3035, reb@brocku.ca.

Thank you for your assistance in this project. Please print a copy of this form for your records.

By clicking on the 'I AGREE' button below, you have indicated that you agree to participate in the study described above. This means that you have read and understood the information provided above. You have agreed that you have had the opportunity to ask questions about this study and still have the chance to ask for details about the study in the future. You understand that your participation is voluntary, and you are free to withdraw at any time without any explanation or penalty.

I AGREE

Appendix C

Semi- Structured Interview Questions (Final Project)

My name is Wendy. Thank you for agreeing to participate in my research. In this interview, I will ask you questions about your final EO to understand how you interact with the learning environment to design and implement your final product. In other words, the questions are designed for you to walk me through your decisions and actions in designing your EO or LO. If at any point during this interview you feel uncomfortable, you may stop without any penalty or without giving a reason. You may also skip a question if you feel the need to do so. There are no penalties. This interview is independent of your participation in the course in that data from this interview does not contribute in any way to the course. I do not give feedback to your instructor about your contribution or non-contribution to this research. Therefore, please feel free to speak freely about your engagement with the learning environment. While I will be asking you specific questions to understand the nature of your interaction with the programming environment, feel free to add any details that you think will help me know what guide your actions in this learning environment. Also, please do not hesitate to ask me for clarifications about any questions as we progress through this interview?

My first set of questions is about the overview of you design?

1. Explain how you design your LO/EO?
 - a. How did you get the idea for your project?
 - b. What is the purpose of your exploratory object? What did you hope to accomplish by designing your EO or LO?
 - c. How did you get started designing your project?
 - d. What computational concepts did you initially use in your design (e.g., sequencing loops, events, parallelism, conditional, etc.)
 - e. Why did you choose to design your LO/EO that way? [interviewer will refer to specific concepts mentioned by the participants. For example, why did you choose to use loops in your design?]
 - f. I noticed that you used [specifics computational concepts not mentioned by participants] in you design?
 - a. Please explain why you used those [chosen computational concepts]. What prompted you to use [chosen computational concepts] in your design?
 - b. Did you choose [chosen computational concepts] based on anything from the task or your understanding of programming? Explain
 - g. What mathematical concepts, including formulas, did you initially use in your design?
 - h. Why did you choose to use [specifics concepts mentioned by participants] in your design?

- i. I noticed that you used [specifics mathematical concepts not mentioned by participants] in you design]?
 - a. Please explain why you used those [chosen mathematical concepts] in you design. What prompted you to use [chosen mathematical concepts] in your design?
 - b. Did you choose [chosen computational concepts] based on anything from the task or your understanding of the mathematics? Explain
2. What changes did you make to your LO/EO?
 - a. Did you add anything new (computational concepts or mathematical concepts) to your design as you code your LO/EO?
 - b. What prompts those changes? Why did you decide to change from your original design?
 - a. Were the changes due to the mathematics, anything you noticed in the code or task?
 - c. Did the purpose of your design change (i.e., in terms of what you want your learning or exploratory object to accomplish)?
 - a. Was this due to the mathematics, anything you noticed in the code or task?
3. How did you implement those changes?
4. How difficult was it to make those changes? Explain?
5. Did you have any other challenges or difficulties implementing your design (computational concepts or mathematical concepts)? Explain?
6. Did you think of any other alternate design at any point during your coding? That is, any other mathematical or programming concepts you could have used?
 - a. Why did you decide against using those mathematical concepts of programming concepts?
 - b. Was this due to the mathematics, anything you noticed in the code or task?
7. Did you seek help from your peers or instructor or did any research for your LO/EO? Explain when and why?
8. Did you do any debugging? Explain?
 - a. What prompted you to do debugging?
 - b. How did debugging affect your final design of your LO/EO and your overall understanding? Explain the changes you made and how they affected your understanding of programming for mathematics learning.
9. How does debugging contribute to your overall understanding of programming for mathematics learning?

10. Did you learn anything new about the computational concepts or practice of [whatever the learner used to design LO/EO]? Explain how you came to that understanding?
 - a. Do you think you would have learned those same concepts or practices outside of programming?
11. Did you learn anything new about the mathematical concepts of [whatever the learner used to design LO/EO]? Explain how you came to that understanding?
 - a. Do you think you would have learned those same concepts outside of programming?
12. What is your overall perspective of programming for doing mathematics?
 - a. What was your perception of programming for mathematics learning before the course?
 - b. Did you have any experience with programming for mathematics learning before entering the course? Describe
13. How did your prior experience with programming for mathematics learning impact your engagement in this course?
 - a. Has your perception of programming for mathematics learning changed due to your engagement with this course? Explain?
 - b. Do you think you will use the knowledge from this course in your career of choice? Explain
14. On a scale of 0(null)-10 (excellent), how would you rate your proficiency in using programming for math learning (computational skills to design mathematical learning objects) before the course? Explain?
15. On a scale of 0(null)-10 (excellent), how would you rate your proficiency in using programming for math learning (computational skills to design mathematical learning objects] after the course? Explain
 - a. What do you think are the contributing factors to that change (before and after)? Explain?
16. Is there anything you wish to comment on that I have not mentioned?
17. Do you have any other questions?

Thank you very much for your time and sharing your experience.

Appendix D

Email to Instructor

Hello (Name MICA Instructor)

I am seeking your permission to invite your MICA students to participate in a study that involves research. The purpose of this study is to explore the nature of interactions (co-action) between learners and their programming environment as they use programming for mathematics learning. Specifically, I am interested in understanding what guides students' actions as they program for mathematics learning.

As participants, your student will be asked to participate in a semi-structured one-on-one audio-recorded interview that will last approximately 45 minutes. The interview will be conducted virtually and will consist of questions about their final learning object (LO) or exploratory object (EO) so that I can understand how they interact with the learning environment to design their object. I am hoping that I can collect their EO or LO and associated reflection one to two weeks before the interviews so that I can ask questions directly related to corresponding assignment pieces.

As all participants will be asked to participate in an interview and do a member check of their transcribed interview, they will receive \$50 compensation.

There are no direct benefits to participants nor any significant risk to participants. However, there may be some psychological risk if they had a negative experience with the programming learning environment. Also, participants may wonder if their involvement in the study may contribute to their final grade since they will be asked to submit an EO or LO with the corresponding reflection. Therefore, participants will be informed before interviews that their participation is not for evaluation. That is, participants will be told that data from the study will not contribute to the course, and I do not give feedback to course instructors about participation or non-participation participants.

Participants may also have confidentiality concerns. To minimize the risks, participants will be informed before interviews that participation in this study is voluntary and that they can withdraw from the research at any time without giving a reason and without penalty before dissemination of information. Also, before each interview, participants will be notified that they can stop the conversation or choose not to answer a question without withdrawing from the study if they feel uncomfortable.

Furthermore, participants will be informed about how their data will be kept confidential. Each interview will be stored using a unique code for confidentiality. Participants' identities will not be associated with the data collected nor be revealed at any time before or after the study. Only pseudonyms, gender, year, and program of study may be used in reporting and storing data. Anonymous quotations may be used for reporting. Furthermore, the name of participants and course instructors will only be known to the Principal Student Investigator and the supervisor,

and only the said persons will have access to data. All data about the research will be stored in my password-protected computer in password-protected files.

The results of this study will be included in my dissertation for my PH. D in Educational Studies and may be published in professional journals and presented at conferences. The study results will be reported in ways to maintain confidentiality. Anonymous excerpts from data (transcription, reflection, or EO/LO) may be used in reports. That is, while gender and program of study may be reported, only pseudonyms will be used in reporting.

You may contact me at wfl7jp@brocku.ca if you need further information about this study. This study has been reviewed and received ethics clearance through the Research Ethics Board at Brock University [REB-21-276]. If you have any comments or concerns about your students' rights as research participants, don't hesitate to get in touch with the Office of Research Ethics Research Ethics Office at (905) 688-5550 Ext. 3035, reb@brocku.ca.

Thank you for your assistance in this project.

Kind regards
Wendy