

**Teaching Computational Thinking with Block-Based
Programming Tools:
A Synthesis of Current Research**

Table of Contents

Chapter 1: Introduction	3
1.1 Introduction.....	3
1.2 Background	4
1.3 Problem Statement	5
1.4 Research Objectives.....	7
1.5 Scope of the Study	8
Chapter 2: Literature Review	9
2.1 Overview of Reviewed Studies.....	9
2.2 Critical Analysis.....	10
2.3 Connections.....	11
2.4 Conflicts Between the Findings	12
2.5 Synthesis Matrix for Literature Review.....	13
Chapter 3: Research Methodology	15
3.1 Research Design & Overview	15
3.2 Methodological Steps to Achieve Research Objectives	15
3.3 Data Collection & Analytical Tools	16
3.4 Expected Outcomes	17
Chapter 4: (Gantt Chart)	18
References	19

Chapter 1: Introduction

1.1 Introduction

In recent years, the presence of Computer Science education in primary and elementary school curricula has evolved from a specialized elective to a basic educational priority. At the core of this movement is the development of Computational Thinking (CT): a cognitive problem-solving framework that includes abstraction, algorithmic reasoning, problem decomposition, pattern recognition, and debugging. Visual and block programming environments, especially Scratch, have become the dominant vehicle for introducing such complex ideas to young students. Platforms like Scratch replace syntax-heavy text code with user-friendly, drag-and-drop visual blocks, reducing the entry barrier and allowing elementary students to work directly with computational logic, structural loops, and conditional reasoning. While block-based programming tools have seen broad acceptance in K-9 education, important open questions remain about their empirical effectiveness, the sub-skills they promote, and how they can be systematically integrated into core academic subjects such as mathematics and science. Some studies report substantial learning in algorithmic logic and structural programming concepts, while others argue that short-term visual coding interventions mainly improve creative expression, collaboration and self-efficacy, but not problem-solving skills. Assessment methods are also an ongoing challenge for educational researchers, as classic paper-and-pencil tests often do not do justice to the dynamic reasoning and process-based learning that occurs during interactive coding sessions. This research proposal examines the effects, pedagogical incorporation and evaluation of Scratch and visual programming environments for the development of computational thinking skills in elementary education.

1.2 Background

The conceptual basis of computational thinking at the early educational stage is grounded in the assumption that basic coding procedures develop habits of thinking transferable to problem-solving. Systematic literature reviews confirm the core educational principles (CEPs) of block-based environments, namely algorithmic design, data abstraction, automation, logic, and iterative refinement. Statistically significant gains in students' understanding of programming concepts such as sequences, loops and conditional execution have been demonstrated empirically in a variety of educational environments with a formal, long-term engagement with Scratch.

Visual programming can have a greater impact when it is integrated into existing subject matter curricula rather than being taught as an isolated technical skill. Indeed, recent educational frameworks suggest teaching Scratch in a subject matter curriculum. Some promising pedagogical models include:

Subject Integration (STEM & Math): Frameworks such as the CTMath model utilize Scratch to contextualize abstract mathematical topics—such as plane geometry and floor-tiling tasks—leading to high normalized learning gains and enhanced debugging capabilities.

Inquiry-Based Learning: Embedding Scratch activities within established instructional frameworks, such as the 5E Learning Model (Engage, Explore, Explain, Elaborate, Evaluate) in science units, has been shown to significantly elevate student self-efficacy and collaborative problem-solving.

Comparative Media Approaches: Scholars have evaluated the relative efficacy of screen-based visual tools (e.g., Scratch, App Inventor) against tangible coding environments (e.g., LEGO EV3 robotics), finding that while both foster core

computational competencies, screen-based visual platforms offer a highly accessible and scalable model for classroom-wide implementation.

However, a critical review of the existing literature reveals notable methodological trade-offs across current studies. Many empirical investigations rely on small sample sizes, quasi-experimental designs lacking random assignment, short intervention windows (typically 4 to 8 weeks), and narrow multiple-choice testing instruments. Therefore, there is a clear academic necessity to synthesize these findings, identify ongoing research gaps, and propose robust methodological frameworks to evaluate the long-term cognitive and practical outcomes of block-based programming in primary education.

1.3 Problem Statement

While computational thinking has become an essential skill for today's students, our enthusiasm for bringing visual coding tools like Scratch into schools has outpaced our understanding of them. Simply put, these programs are spreading rapidly through elementary and middle classrooms, but we still lack enough real-world research to know their true impact on learning. Educational systems are often built on the assumption that these tools of software automatically promote sub-skills such as abstraction, decomposition and algorithmic thinking, but the current evidence in the field remains contradictory and fragmentary. For example, Djambong and Freiman (2016) report very limited cognition growth in sixth-grade students who used Scratch and note that there was no significant growth in algorithmic thinking skills.

Likewise, Koray and Bilgin (2023) revealed that programming activities improve collaboration and creativity but have no significant impact on problem solving and

critical thinking. This shows that we still don't fully understand which sub-skills Scratch actually builds, or what the best teaching environment is for using it.

This disparity points to a real gap in our knowledge: we do not know exactly which sub-skills Scratch develops, nor the most appropriate instructional environment for Scratch implementation.

This ambiguity is a problem for curriculum developers because the distinction between exposing students to programming and developing real computational competence is sometimes blurred. To address this issue, recent research has explored the possibility of connecting Scratch to structured pedagogical frameworks. For example, Fagerlund et al. (2021) synthesized the field into 12 fundamental concepts for education to standardize teaching and assessment. Other authors have used validated instructional design models to create a coherent educational structure, such as the ADDIE model (Yusrina et al., 2026) or the 5E learning cycle (Koray & Bilgin, 2023).

Several studies sought to assess differences through standardized tests such as the Computational Thinking Test for Beginners (Piedade & Dorotea, 2023; Martínez-Murciano & Pérez-Jorge, 2024) in terms of assessment. However, a number of reviews demonstrate that despite such efforts, primary schools still lack flexible assessment tools capable of measuring qualitative indicators such as the ability to process information and creativity, because assessment is often limited to a superficial examination of the final project, without tracking the cognitive processes of students during the work itself.

Also, there are clear methodological problems with current research. Most research is done on small, non-random samples and in single schools over short periods of time (as little as four to eight weeks). This makes it impossible to generalize findings. Also, post-testing alone hides basic individual differences among students present

before the experiment. Another gap that is largely ignored in the core body of research is teacher readiness. Even well-designed interventions depend on the confidence and preparedness of the teachers who deliver them, a factor considered only outside the six core studies (Kong, Lai, & Sun, 2020).

So, there's a real need to bring this scattered data together, look closely at the limits of the current research, and figure out which teaching practices actually work. This study thus provides a detailed review and critique of the literature on teaching computational thinking through Scratch and related block-based tools in schools.

1.4 Research Objectives

In this study, we aim to understand the existing contradictions in Scratch-based instruction, to identify the computational skills that can be reliably developed and the most suitable educational environments for them. Thus, the study is designed around the following five major objectives:

1. To assess the effectiveness of Scratch programs to develop computational thinking skills (e.g., abstraction, decomposition) through a review of the results of previous quasi-experimental studies and systematic reviews.
2. Evaluate the effectiveness of current assessment tools employed in primary schools (e.g., Computational Thinking Test for Beginners, Computational Thinking Scale) and determine their limitations, especially regarding the assessment of non-linear skills such as creativity and collaboration.
3. Evaluate the educational efficiency of Scratch incorporation into structured teaching frameworks (e.g., 5E and ADDIE models) as compared to its standalone programming subject teaching.
4. Consider the influence of age and other technological devices (e.g., robots) in the findings. Consider methodological limitations in the generalizability of current

research findings (e.g., length and size of the trials).

5. Extend the analysis beyond just primary-school student outcomes by using evidence beyond the six core classroom studies, including artifact-level tool comparisons, early-childhood interventions, and teacher-readiness research.

1.5 Scope of the Study

In an effort to keep the analysis of the development of the Computational Thinking (CT) in the early education focused, this synthesis and research proposal is limited by specific contextual, thematic, temporal and methodological dimensions: Thematic Scope: Core CT competencies (problem decomposition, abstraction, algorithmic design, pattern recognition and debugging) are explored through standalone coding curricula and subject-integrated models at the primary level, i.e. Mathematics (e.g. plane geometry) and Science (e.g. Force and Motion via the 5E model).

Temporal Scope: The synthesized body of research comprises peer-reviewed research, conference proceedings, and systematic reviews published between 2016 and 2026, with intervention windows ranging from short-term (4–8 weeks) to extended academic-year implementation (e.g., 2020–2021).

Population and Spatial Scope: The scope centers on primary and lower-secondary school environments, spanning Kindergarten through Grade 9 (ages 5–15), with a strong emphasis on Grades 4–6, and synthesizes empirical data collected internationally, including Canada, Portugal, Turkey, Indonesia, and cross-national literature reviews.

Technical Scope: The primary technical focus is on block-based visual programming environments—specifically Scratch (web-based and desktop

platforms)—with secondary comparative evaluation of App Inventor and tangible educational robotics (LEGO EV3).

Chapter 2: Literature Review

2.1 Overview of Reviewed Studies

Before turning to a critical synthesis of the findings, this section briefly situates each of the ten studies reviewed. Djambong and Freiman (2016) conducted an exploratory pilot comparing Scratch and LEGO EV3 across two grade levels, finding limited algorithmic gains. Fagerlund et al. (2021) systematically reviewed 30 studies to define 12 Core Educational Principles for teaching CT with Scratch. Piedade and Dorotea (2023) ran a quasi-experimental study with Portuguese fourth-graders, showing significant gains in loop and conditional understanding. Martínez-Murciano and Pérez-Jorge (2024) conducted a PRISMA review comparing Scratch and App Inventor across ten studies. Yusrina et al. (2026) introduced CTMath, an ADDIE-based Scratch model for elementary mathematics that produced large effect sizes.

Koray and Bilgin (2023) embedded Scratch within the 5E science model with Turkish sixth-graders, finding gains in creativity and collaboration but not in algorithmic or critical thinking. Park and Shin (2019) used automated Dr. Scratch rubric analysis to compare Scratch and App Inventor projects on structural sub-skills. Yang, Ng, and Gao (2022) and Montuori et al. (2023) extended the debate into early-childhood settings, comparing robotics, block play, and unplugged activities. Finally, Kong, Lai, and Sun (2020) examined a year-long teacher professional-development program, the only study in this set to address educator

readiness rather than student outcomes directly. Together, these ten sources form the empirical and methodological basis for the critical analysis below.

2.2 Critical Analysis

Collectively, these studies show that visual programming languages (primarily Scratch) are a powerful and promising tool for teaching computational thinking to children and young people. Still, a conclusive scientific conclusion must be reached on account of the methodological limitations identified in this body of work.

Studies such as Piedade & Dorotea (2023) and Yusrina et al. (2026) demonstrate a quasi-experimental methodology with control groups and standardized tests, showing clear statistical differences, which is consistent in its strengths. Additionally, the comprehensive reviews by Fagerlund et al. (2021) and Martínez-Murciano & Pérez-Jorge (2024) successfully provide an explicit map of the current literature.

Studies by Koray & Bilgin and Yusrina suggest that programming has transformed from a mere technical activity to facilitated instruction through the use of explicit instructional models such as 5E and ADDIE.

On the other hand, there are some methodological deficiencies in this literature; the majority of samples are very small (such as the 22-student samples in the Djambong and Koray studies), and genuine random sampling is almost non-existent. Some studies do not conduct pre-tests, which undermines the precision of post-tested outcomes. In addition, conducting tests in a single school for one to two months raises questions about students' ability to retain these skills over time.

Present-day assessment systems are incapable of evaluating the cognitive processes and creative thinking involved in programming for children through conventional

multiple-choice tests, or by solely relying on the final outcome. Future research should focus on conducting long-term studies and utilizing dynamic assessment tools to capture the student's thought process at every step.

Beyond these six core studies, related work adds useful nuance. Park and Shin (2019) show that Scratch and App Inventor are not simply better or worse than each other but cultivate different sub-skills, which helps explain why Martínez-Murciano and Pérez-Jorge (2024) found only a partial, incomplete comparison between the two tools. Yang, Ng, and Gao (2022) and Montuori et al. (2023) push the plugged-versus-unplugged and standalone-versus-integrated questions down into early childhood, suggesting these are not just primary-school debates. And Kong, Lai, and Sun (2020) point to a factor missing from all six core studies: how ready teachers actually are to deliver CT instruction, which may explain some of the inconsistent findings across the literature, including the uneven skill-specific gains reported by Koray and Bilgin (2023).

2.3 Connections

1. The shared 'Scratch' platform as the primary teaching tool across most of the research, with App Inventor, LEGO EV3 robotics, and other tools appearing as points of comparison.
2. Collectively, the papers aimed to gauge computational thinking by its most well-known skills, including abstraction and decomposition, algorithms, debugging, and pattern recognition.
3. Research focused on a nearly equal grade range (4th, 5th) and early-secondary years (6th–9th), suggesting that these years are commonly seen as conducive to learning computational algorithms.

4. Adhering to topics: various studies have begun to connect programming with science and mathematics, rather than teaching it alone.
5. Extending beyond primary and lower-secondary grades: Yang, Ng, and Gao (2022) and Montuori et al. (2023) show that the same tensions around tool choice and instructional design appear even earlier, in preschool and early-childhood settings.

2.4 Conflicts Between the Findings

1. Various sub-skills were identified as improving: the work of Piedade & Dorotea and Yusrina demonstrated significant improvements in understanding conditional statements and iteration, while the research conducted by Koray and Bilgin showed no improvement in algorithmic thinking or problem-solving.
2. Djambong & Freiman suggest that Scratch is not as effective as physical tools, and that it can be used to enhance the performance of older students when paired with tangible tools like LEGO kits and robots. However, other studies indicate that Scratch alone can produce satisfactory results.
3. Some argue that it is better to use Scratch within core subject lessons like math and science, while others see it as a standalone programming lesson. The issue remains unresolved, as both approaches have produced successful outcomes.
4. A clear discrepancy exists between empirical studies that exclusively use standardized multiple-choice tests and systematic reviews that describe these tools as insufficient for measuring computational thinking.

2.5 Synthesis Matrix for Literature Review

Study Title	Authors	Year	Research Design	Key Findings	Methodology & Limitations
Task-Based Assessment of Students' CT Skills Through Visual Programming or Tangible Coding Environments	Djambong & Freiman	2016	Exploratory pilot case study (pre/post-test)	Grade 9 outperformed Grade 6 overall (44.9%→48.2% vs 29.6%→33.0%); isolated algorithmic items showed limited gains.	24 students, 5-week intervention, New Brunswick, Canada. No random assignment or control group; very small sample.
Computational Thinking in Programming with Scratch in Primary Schools: A Systematic Review	Fagerlund, Häkkinen, Vesisenaho & Viiri	2021	Systematic literature review (K–9)	Defined 12 Core Educational Principles (CEPs) for CT in Scratch; wide variation in assessment methods.	30 studies selected from 432 initial results. Possible selection bias; efficiency/creativity rarely assessed.
Effects of Scratch-Based Activities on 4th-Grade Students' CT Skills	Piedade & Dorotea	2023	Quasi-experimental, post-test only with control group	Experimental M=20.91 vs control M=16.25 ($\Delta M=4.66$, $p<0.001$); simple loops 85% vs 67%.	189 Portuguese students. No pre-test; non-probability sampling; single national context.
Computational Thinking with Scratch or App Inventor in Primary Education	Martínez-Murciano & Pérez-Jorge	2024	PRISMA systematic review	Both tools develop CT, motivation, and collaboration; Scratch favors math competence, App Inventor favors creativity.	PRISMA review of 425 records; 10 studies retained. Only one used App Inventor; long-term durability unclear.
CTMath: Enhancing Elementary Mathematics and CT Through a Structured Scratch-Based Model	Yusrina, Sutama, Sutopo & Muhibbin	2026	Design-based research + quasi-experimental	CTMath N-gain=0.487 vs control 0.224; Hedges' $g=0.97$ (large effect).	ADDIE five-phase model, $n=43$. Single region/topic; low inter-rater reliability.

Study Title	Authors	Year	Research Design	Key Findings	Methodology & Limitations
The Effect of Block Coding (Scratch) Activities Integrated into the 5E Learning Model in Science Teaching on CT Skills and Programming Self-Efficacy	Koray & Bilgin	2023	Quasi-experimental single-group pre-/post-test	Total CT Scale gain +3.81 ($p<0.05$); creativity and collaboration significant, algorithmic/critical thinking not significant.	22 Turkish students, 5-week intervention. Small sample; no control group; novelty effect possible.
Comparing the Effectiveness of Scratch and App Inventor with Regard to Learning CT Concepts	Park & Shin	2019	Automated rubric-based content analysis (Dr. Scratch)	Scratch scored higher on parallelism/flow control; App Inventor scored higher on interactivity/data representation.	Analysis of shared projects, not a classroom study.
Robot Programming Versus Block Play in Early Childhood Education	Yang, Ng & Gao	2022	Comparative early-childhood intervention	Extends the plugged-vs-unplugged debate to early childhood; compares CT, sequencing, and self-regulation.	Early-childhood sample.
Combined Unplugged and Educational Robotics Training to Promote CT and Cognitive Abilities in Preschoolers	Montuori, Pozzan, Padova, Ronconi, Vardanega & Arfé	2023	Quasi-experimental preschool intervention	Tests whether combining unplugged activities with robotics adds benefit for CT and cognitive abilities.	Relevant to the curriculum-integration-vs-standalone conflict.
Teacher Development in Computational Thinking: Design and Learning Outcomes of Programming Concepts, Practices and Pedagogy	Kong, Lai & Sun	2020	Year-long professional-development program	127 teachers grew confident with sequences, algorithms, loops; less secure with conditionals, variables, functions.	Only study examining teacher readiness.

Chapter 3: Research Methodology

3.1 Research Design & Overview

To address the contradictions in the literature regarding visual programming and to achieve the five research objectives, this study adopts a mixed-methods literature synthesis and proposal framework. The research combines quantitative meta-synthesis of empirical intervention data with qualitative thematic analysis of pedagogical structures, assessment tools, and contextual variables.

3.2 Methodological Steps to Achieve Research Objectives

Phase 1 — Empirical Efficacy Analysis: Systematic extraction and synthesis of quantitative effect sizes (e.g., Hedges' g , mean gains, t-test values) across quasi-experimental studies, measuring specific skill acquisition by isolating procedural logic (loops, conditionals) versus cognitive sub-skills (abstraction, decomposition, debugging).

Phase 2 — Evaluation of Assessment Instruments: Comparative analysis of existing measurement tools (e.g., BCTt, CTS, Dr. Scratch rubric analysis), identifying limitations in psychometric validity, baseline measurement control, and their ability to capture non-linear, soft competencies like creativity and collaboration.

Phase 3 — Pedagogical Integration Comparative Framework: Synthesis of instructional models, comparing standalone coding interventions against curriculum-embedded approaches, and evaluating normalized learning gains when

Scratch is integrated into structured instructional models (e.g., ADDIE in Mathematics, 5E in science) versus isolated weekly modules.

Phase 4 — Confounding & Contextual Variable Mapping : Cross-study comparative analysis examining sample age ranges (Grades K–9), intervention durations (4 weeks to a full academic year), and tool modalities, disentangling age-maturity effects from tool characteristics.

Phase 5 — Extended Multidimensional Synthesis : Incorporating secondary empirical domains beyond standard K–9 classroom outcomes, integrating code-level project artifact analysis, early-childhood unplugged/robotics interventions, and teacher Pedagogical Content Knowledge (PCK) data to explain outcome variances.

3.3 Data Collection & Analytical Tools

- **Synthesis Matrix & Categorization:** a standardized data-extraction matrix capturing study design, sample size, intervention length, target competencies, pedagogical models, and methodological trade-offs.
- **Automated & Psychometric Metrics:** Dr. Scratch automated rubric data alongside validated testing scales (BCTt, CTS) to evaluate project structure versus student self-efficacy.
- **Comparative Cross-Tabulation:** mapping tool capabilities (Scratch vs. App Inventor vs. LEGO EV3) across specific CT sub-domains (e.g., parallelism, synchronization, interactivity).

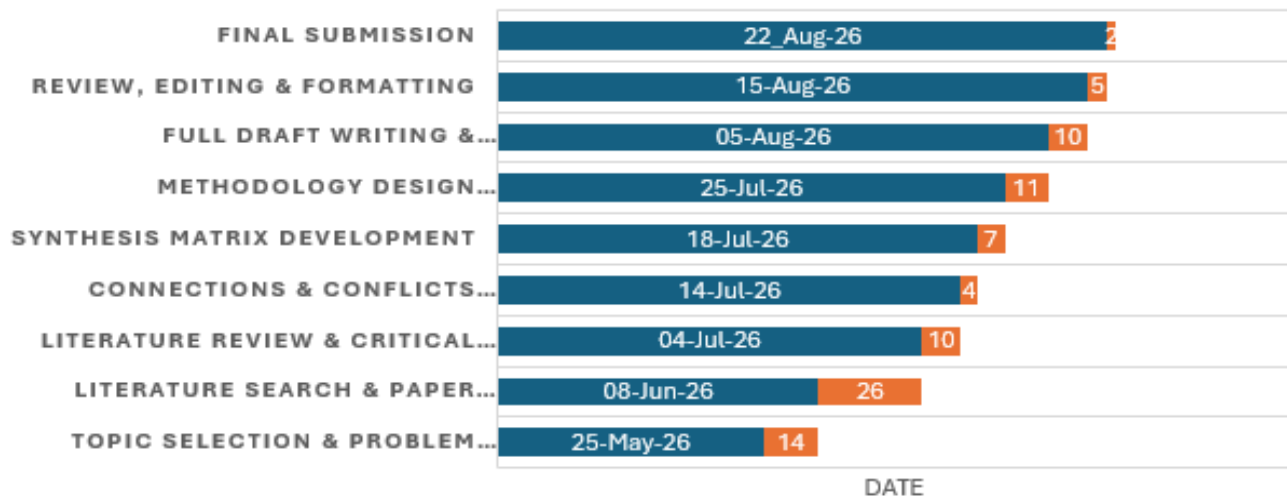
3.4 Expected Outcomes

This research synthesis and proposed framework aim to deliver clear scientific and practical contributions to the field of Computer Science Education and primary pedagogy, structured across two dimensions:

Scientific Contributions — a harmonized conceptual taxonomy mapping specific block-based tools to the CT sub-skills they most effectively foster; a standardized evaluation matrix identifying measurement gaps in existing instruments and proposing a dynamic evaluation framework combining artifact-level metrics with process-based cognitive assessment; and methodological recommendations addressing current trade-offs (small samples, post-test-only designs, short interventions) to enable more reliable longitudinal studies.

Practical Deliverables — evidence-based instructional guidelines for curriculum designers on integrating block-based coding into core subjects rather than teaching it in isolation, and a teacher-preparedness roadmap highlighting the prerequisite Pedagogical Content Knowledge required for sustainable CT instruction.

Chapter 4: (Gantt Chart)



References

- Djambong, T., & Freiman, V. (2016). Task-based assessment of students' computational thinking skills developed through visual programming or tangible coding environments. In 13th International Conference on Cognition and Exploratory Learning in Digital Age (CELDA 2016) (pp. 41–51).
- Fagerlund, J., Häkkinen, P., Vesisenaho, M., & Viiri, J. (2021). Computational thinking in programming with Scratch in primary schools: A systematic review. *Computer Applications in Engineering Education*, 29(1), 12–28. <https://doi.org/10.1002/cae.22255>
- Kong, S. C., Lai, M., & Sun, D. (2020). Teacher development in computational thinking: Design and learning outcomes of programming concepts, practices and pedagogy. *Computers & Education*, 151, 103872. <https://doi.org/10.1016/j.compedu.2020.103872>
- Koray, A., & Bilgin, E. (2023). The effect of block coding (Scratch) activities integrated into the 5E learning model in science teaching on students' computational thinking skills and programming self-efficacy. *Science Insights Education Frontiers*, 18(1), 2825–2845. <https://doi.org/10.15354/sief.23.or410>
- Martínez-Murciano, M. C., & Pérez-Jorge, D. (2024). Computational thinking with Scratch or App Inventor in primary education. *Campus Virtuales*, 13(2), 107–122. <https://doi.org/10.54988/cv.2024.2.1364>

- Montuori, C., Pozzan, G., Padova, C., Ronconi, L., Vardanega, T., & Arfé, B. (2023). Combined unplugged and educational robotics training to promote computational thinking and cognitive abilities in preschoolers. *Education Sciences*, 13(9), 858. <https://doi.org/10.3390/educsci13090858>
- Park, Y., & Shin, Y. (2019). Comparing the effectiveness of Scratch and App Inventor with regard to learning computational thinking concepts. *Electronics*, 8(11), 1269. <https://doi.org/10.3390/electronics8111269>
- Piedade, J., & Dorotea, N. (2023). Effects of Scratch-based activities on 4th-grade students' computational thinking skills. *Informatics in Education*, 22(3), 499–523. <https://doi.org/10.15388/infedu.2023.19>
- Yang, W., Ng, D. T. K., & Gao, H. (2022). Robot programming versus block play in early childhood education: Effects on computational thinking, sequencing ability, and self-regulation. *British Journal of Educational Technology*, 53(6), 1817–1841. <https://doi.org/10.1111/bjet.13215>
- Yusrina, H.,utama, Sutopo, A., & Muhibbin, A. (2026). CTMath: Enhancing elementary mathematics and computational thinking skills through a structured Scratch-based model. *Frontiers in Education*, 11, 1767445. <https://doi.org/10.3389/feduc.2026.176744>