Available online at https://journal.rescollacomm.com/index.php/ijeer/index

International Journal of Ethno-Sciences and Education Research

Vol. 6, No. 1, pp. 17-26, 2026

From Concept Image to Computational Thinking: A Design-Based Research on Python-Integrated Mathematics Instruction

Damar Rais[1*]

[1]*Mathematics Department, State University of Makassar, Makassar, Indonesia*

**Corresponding author email: damar.rais@unm.ac.id*

Abstract

The growing demand for computational thinking (CT) in mathematics education calls for instructional designs that meaningfully connect abstract mathematical concepts with algorithmic reasoning. Grounded in the theory of concept image, this study investigates how Python-integrated instructional activities can support the transformation of students' mathematical understanding while simultaneously fostering computational thinking skills. Employing a Design-Based Research (DBR) methodology, the study was conducted through three iterative design cycles in a Calculus II (Integration) course involving twenty-six undergraduate mathematics education students at a university in Indonesia. Data were collected through concept image mapping tasks, computational thinking performance assessments, classroom observations, reflective interviews, and analyses of students' Python programming artifacts. The findings indicate that Python-assisted dynamic visualization facilitated a shift from static, procedural concept images toward more dynamic and relational cognitive representations. Furthermore, programming activities strengthened students' abstraction and algorithmic reasoning, as coding functioned as an externalization of students' evolving concept images. Based on iterative analysis across DBR cycles, this study formulates three instructional design principles: Concept-First Coding, Representational Fluidity, and Reflective Alignment. These principles position Python programming as a cognitive bridge that connects mathematical intuition with formal computational reasoning. The study contributes both theoretically, by integrating concept image theory with computational thinking, and practically, by offering empirically grounded design principles for mathematics instruction in higher education contexts.

Keywords: Concept Image, Computational Thinking, Python Programming, Design-Based Research, Mathematics Education.

1. Introduction

The rapid advancement of digital technologies in the third decade of the twenty-first century has fundamentally reshaped the landscape of mathematics education. Within the context of Industry 4.0 and the emerging paradigm of Industry 5.0, mathematics is increasingly viewed not merely as a discipline centered on numerical manipulation, but as a foundational component of digital literacy and complex problem solving. Alongside reading, writing, and numeracy, computational thinking (CT) has been widely recognized as a core competency for contemporary learners (Angeli & Giannakos, 2020).

Computational thinking encompasses cognitive practices such as decomposition, pattern recognition, abstraction, and algorithmic reasoning—processes that closely align with fundamental mathematical thinking. Consequently, integrating CT into mathematics curricula has become a strategic priority worldwide. However, despite its recognized importance, the implementation of computationally integrated mathematics instruction remains pedagogically challenging. In many instructional settings, programming activities are either reduced to syntax-oriented exercises or used merely as computational tools for executing predefined procedures, without fostering deeper mathematical reasoning.

This instructional dilemma reflects a broader epistemological tension between syntax and semantics in mathematical and computational representations. A syntax-driven approach risks reducing mathematical meaning to mechanistic code execution, thereby weakening conceptual understanding. Research in mathematics education and cognitive science has consistently demonstrated that well-formed symbolic expressions do not inherently guarantee meaningful understanding without semantic grounding and conceptual interpretation (El Turkey et al., 2024; Keller et al., 2021; O'Brien, 2024). In mathematics classrooms, this tension becomes particularly visible when students are able

to execute algorithms correctly but struggle to explain the underlying concepts or apply them flexibly in unfamiliar contexts.

One theoretical framework that offers insight into this phenomenon is the theory of concept image and concept definition. Concept image refers to the total cognitive structure associated with a mathematical concept, including mental pictures, properties, and processes, whereas concept definition denotes the formal and accepted verbal formulation of that concept (Tall & Vinner, 2022). Persistent learning difficulties often arise when students' concept images are incomplete, static, or misaligned with formal definitions. Traditional mathematics instruction frequently reinforces such static concept images, emphasizing symbolic manipulation over conceptual exploration.

The integration of text-based programming environments such as Python introduces new opportunities for reshaping students' concept images. By translating mathematical relationships into executable code, students are required to make their implicit reasoning explicit. Parameter manipulation and dynamic visualization allow learners to observe how changes in variables affect mathematical behavior in real time, potentially fostering richer and more coherent mental representations. Despite this potential, empirical research examining how Python-based instruction systematically transforms students' concept images—particularly through iterative instructional design approaches—remains limited, especially in higher education contexts in Indonesia (Hidayah & Sa'dijah, 2021).

At the same time, programming-based mathematics instruction holds promise for developing computational thinking. Writing code to model mathematical ideas compels students to decompose problems, abstract patterns, and construct step-by-step algorithms. Importantly, programming errors function as immediate cognitive feedback, signaling inconsistencies between students' conceptual understanding and formal logical structures. When deliberately designed, programming activities can therefore serve not merely as technical exercises, but as epistemic tools that mediate conceptual growth.

This study argues that a productive synergy exists between concept image development and computational thinking, with programming functioning as a mediating representational system. To investigate this synergy empirically, the study adopts a Design-Based Research (DBR) methodology, which enables the iterative development, implementation, and refinement of instructional interventions within authentic classroom settings (Maharani & Nusantara, 2020; Prahmana, 2022). DBR is particularly suitable for examining learning processes that unfold dynamically over time and for generating instructional design principles grounded in both theory and practice.

Accordingly, the central research question guiding this study is:

What characteristics of Python-integrated mathematics instructional designs effectively transform students' concept images while simultaneously fostering computational thinking skills?

The novelty of this research lies in its explicit integration of a classical cognitive theory—concept image—with a key twenty-first-century competency—computational thinking—through an iterative DBR framework. Rather than focusing solely on learning outcomes or technological affordances, this study examines how students' mental representations evolve as they engage with Python-based mathematical tasks. Through this approach, the study aims to contribute theoretically to mathematics education research and practically to the design of computationally enriched mathematics instruction.

2. Literature Review

2.1. Concept Image and Mathematical Understanding

One of the most influential cognitive frameworks for analyzing students' mathematical understanding is the theory of concept image and concept definition. According to Tall and Vinner (2022), a concept image refers to the entire cognitive structure associated with a mathematical concept, including mental pictures, properties, intuitive beliefs, and associated processes. In contrast, a concept definition represents the formal and accepted verbal or symbolic description of that concept. Learning difficulties often arise when students' concept images are fragmented, static, or inconsistent with formal definitions, leading to procedural performance without conceptual understanding.

Empirical studies have shown that traditional mathematics instruction tends to reinforce static concept images, particularly when learning is dominated by symbolic manipulation and formula memorization (Sari & Kusuma, 2023). For instance, students may associate the concept of a function exclusively with its algebraic formula or a single graphical representation, without recognizing its dynamic input–output relationships. Such limited concept images restrict students' ability to transfer knowledge across contexts or to reason flexibly when confronted with non-routine problems.

Digital tools offer new opportunities to enrich concept images by enabling dynamic representations. Visualization environments allow students to observe how mathematical objects change in response to parameter variation, thereby supporting conceptual coherence. However, many visualization-based tools position students primarily as observers rather than constructors of mathematical meaning. As noted by Ziatdinov and Valles Jr. (2022), dynamic visualizations alone do not necessarily compel learners to engage with the underlying logical structures that generate

these representations. Consequently, the challenge lies not merely in providing visual experiences, but in designing learning environments that require students to actively construct and manipulate mathematical ideas.

2.2. Computational Thinking in Mathematics Education

Computational thinking (CT) has been widely conceptualized as a problem-solving framework involving decomposition, pattern recognition, abstraction, and algorithmic reasoning (Angeli & Giannakos, 2020; Caderheiro & Gadanidis, 2023). In mathematics education, CT extends beyond learning to code; it represents a way of expressing mathematical reasoning in computational forms. When students translate mathematical ideas into algorithms, they engage in abstraction by identifying invariant structures and generalizing patterns across cases (Hickmott & Prieto-Rodriguez, 2022).

Python has emerged as a particularly suitable programming language for mathematics instruction due to its readable syntax, extensive mathematical libraries, and strong visualization capabilities. Prior research indicates that Python-based activities can enhance students' engagement and support higher-order reasoning when appropriately scaffolded (Rais & Xuezhi, 2024; Zhang et al., 2024). Nevertheless, without careful instructional design, programming tasks may impose excessive cognitive load, shifting students' focus toward debugging syntax rather than understanding mathematical concepts (Li & Schoenfeld, 2021; Sweller, 2020).

Recent studies integrating computational tools such as GeoGebra, Scratch, and Python demonstrate positive effects on CT development, particularly in abstraction and algorithmic thinking (Fang et al., 2024; Falloon, 2024). However, many of these studies emphasize learning outcomes rather than examining how students' internal cognitive structures evolve during computational engagement. As a result, the mechanisms through which programming influences conceptual understanding remain underexplored.

2.3. Programming as a Cognitive Bridge: Research Gap

An emerging perspective in mathematics education conceptualizes programming as a cognitive bridge between intuitive understanding and formal mathematical reasoning. Writing code requires students to externalize their mental representations, making implicit assumptions explicit and subject to logical verification. Errors in code execution function as immediate feedback, signaling misalignments between students' concept images and formal structures (Keller et al., 2021).

Despite growing interest in this perspective, empirical studies that explicitly connect concept image theory with computational thinking development remain scarce. Existing research often addresses these constructs in isolation, either focusing on conceptual understanding without computational processes or examining CT skills without grounding them in cognitive theory (dos Santos & de Menezes, 2025; Wulandari & Surya, 2024). Moreover, few studies employ iterative methodologies capable of capturing learning processes over time.

To address this gap, the present study integrates concept image theory and computational thinking within a Design-Based Research (DBR) framework. By iteratively designing, implementing, and refining Python-integrated instructional activities, this study seeks to uncover how students' concept images transform through computational engagement and how these transformations support the development of CT. This approach advances prior research by providing process-oriented evidence of cognitive change rather than solely outcome-based evaluation.

3. Materials and Methods

3.1. Research Design

This study adopts Design-Based Research (DBR) as its methodological framework. DBR is characterized by iterative cycles of design, implementation, analysis, and redesign conducted within authentic educational contexts (McKenney & Reeves, 2018). Unlike experimental approaches that isolate variables under controlled conditions, DBR aims to generate instructional designs that are both theoretically grounded and practically viable. In the context of this study, DBR enables the systematic alignment of concept image theory with computational thinking through Python-integrated mathematics instruction (Maharani & Nusantara, 2020; Prahmana, 2022).

The research was conducted across three DBR cycles, each consisting of (1) instructional design development, (2) classroom implementation, (3) data collection and analysis, and (4) design refinement. Insights gained from each cycle informed modifications to subsequent instructional interventions, allowing the design to evolve responsively to students' learning needs. The overall research procedure followed three iterative Design-Based Research cycles, as illustrated in Figure 1.

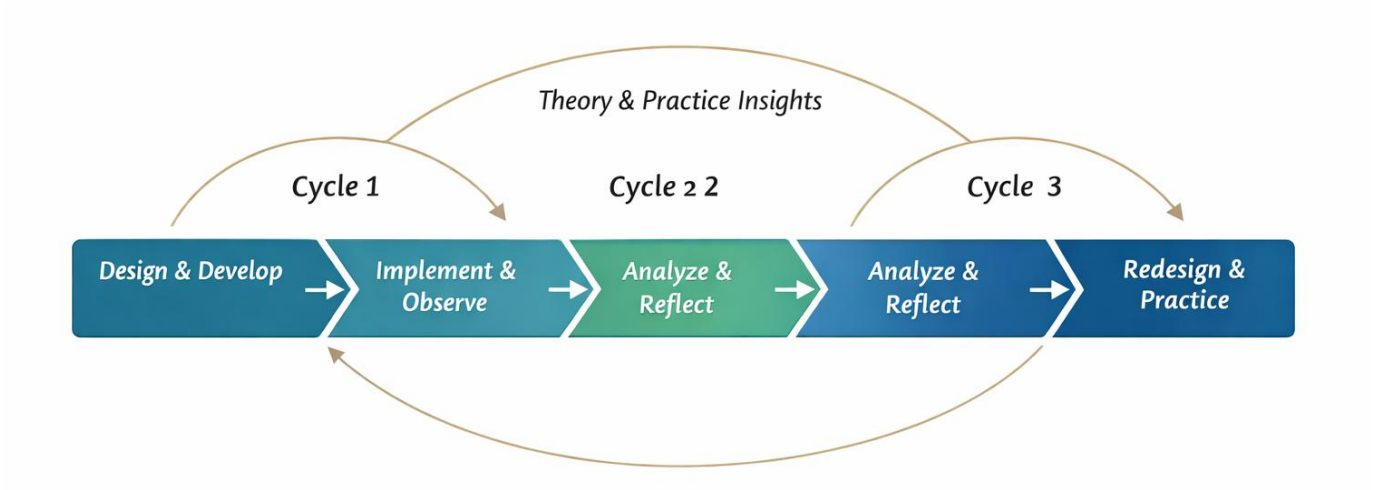


Figure 1: Design-Based Research workflow illustrating iterative cycles of instructional design, implementation, analysis, and refinement.

3.2. Participants and Context

The participants were twenty-six undergraduate students enrolled in a Calculus II (Integration) course in the fourth semester of a mathematics education program at a PGRI university in Yogyakarta, Indonesia. The selection of pre-service mathematics teachers was intentional, as these students are expected to integrate computational thinking into their future teaching practices.

The intervention was implemented over one academic semester within a regular course setting. Python programming was systematically embedded into the calculus curriculum, particularly in topics related to integrals, functions, and symbolic computation. This context provided a meaningful environment for examining how computational tools mediate conceptual understanding in higher mathematics education.

3.3. Instructional Intervention

The instructional intervention consisted of a structured sequence of Python-integrated learning tasks designed to enrich students' concept images while fostering computational thinking. Learning activities were organized around three core components: dynamic visualization, algorithmic reconstruction, and computational reflection.

Initially, students engaged in conceptual exploration through visual representations generated from Python code. Subsequently, they were guided to reconstruct mathematical concepts algorithmically by writing their own code rather than relying solely on built-in functions. Reflective prompts were incorporated to encourage students to explicitly connect code behavior with formal mathematical definitions. This progression was intended to reduce extraneous cognitive load while supporting meaningful abstraction (Sweller, 2020).

3.4. Data Collection

To ensure methodological rigor, data triangulation was employed using multiple sources. Data were collected through: 1) Concept image mapping tasks administered before and after the intervention to capture changes in students' mental representations. 2) Computational thinking performance assessments evaluating students' abilities in decomposition, abstraction, and algorithmic reasoning within Python contexts. 3) Classroom observations documenting instructional interactions and students' engagement during coding activities. 4) Semi-structured interviews conducted with selected participants to explore their reasoning processes and learning experiences. 5) Student-generated Python artifacts, including scripts and notebook outputs, which served as concrete evidence of the externalization of concept images.

3.5. Data Analysis

Data analysis followed a qualitative thematic approach. Observation notes, interview transcripts, and programming artifacts were coded iteratively to identify patterns in concept image transformation and computational reasoning (Tall & Vinner, 2022). Cross-cycle analysis was conducted to trace changes across DBR iterations and to extract recurring instructional features associated with successful learning outcomes.

The analytical process aimed not only to evaluate the effectiveness of the intervention but also to generate empirically grounded instructional design principles that can inform future computationally integrated mathematics instruction.

4. Results and Discussion

4.1. Result

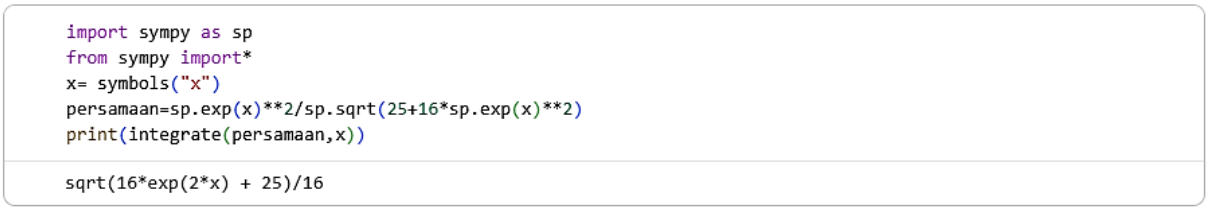
This section presents the empirical findings derived from three Design-Based Research (DBR) cycles. The results are organized around observable changes in students' concept images, the development of computational thinking skills, and the emergence of instructional design principles. To ensure analytic rigor, findings are supported by triangulated data from concept image mapping, classroom observations, computational artifacts, and student reflections.

4.1.1. Development of Computational Thinking Skills

In parallel with changes in concept images, substantial growth was observed in students' computational thinking (CT) skills, particularly in abstraction and algorithmic reasoning. During the first DBR cycle, many students experienced difficulty decomposing mathematical problems into computational steps. Initial coding attempts revealed logical inconsistencies, such as undefined variables or incorrect iteration structures, indicating gaps in procedural reasoning rather than syntactic knowledge.

As DBR cycles progressed, students increasingly demonstrated the ability to decompose complex mathematical tasks into manageable algorithmic components. Programming activities required learners to define variables explicitly, structure control flows, and generalize solutions through parameterization. These requirements compelled students to confront the internal logic of mathematical procedures, making their reasoning processes explicit and testable.

Analysis of students' Python artifacts revealed a clear progression in abstraction. Early code samples focused on solving specific numerical instances, whereas later artifacts featured generalized functions capable of handling multiple cases. This shift reflects an emerging capacity for pattern recognition and abstraction—core pillars of computational thinking. Debugging processes further functioned as cognitive feedback mechanisms, prompting students to revise both their code and their underlying mathematical reasoning.



```
import sympy as sp
from sympy import*
x= symbols("x")
persamaan=sp.exp(x)**2/sp.sqrt(25+16*sp.exp(x)**2)
print(integrate(persamaan,x))
```

```
sqrt(16*exp(2*x) + 25)/16
```

Figure 2: Student-generated Python code artifact illustrating the symbolic construction and evaluation of an integral expression as evidence of algorithmic reasoning and abstraction.

4.1.2. Evolution of Students' Concept Images Across DBR Cycles

Analysis of students' concept image mapping tasks and classroom observations revealed a progressive transformation in students' mathematical understanding across the three DBR cycles. During the initial cycle, students' concept images were predominantly static and procedural. Most participants relied heavily on memorized formulas and algorithmic templates, demonstrating limited awareness of the structural relationships underlying integral concepts. This limitation became apparent when students encountered non-routine problems, where symbolic manipulation alone proved insufficient.

In the second DBR cycle, the introduction of Python-based dynamic visualization prompted noticeable changes in students' cognitive representations. By modifying parameters directly within Python code and observing immediate graphical feedback, students began to perceive mathematical expressions as dynamic systems rather than fixed symbolic entities. This interaction supported a predict–observe–explain pattern of reasoning, where students formed hypotheses about graphical behavior before executing code and then reconciled discrepancies through discussion and reflection.

By the third DBR cycle, students demonstrated relational concept images, characterized by the ability to explain causal relationships among variables and to articulate how algebraic structures influenced graphical outcomes. Qualitative analysis of student explanations indicated increased conceptual coherence, as learners were able to move fluidly between symbolic expressions, computational representations, and visual interpretations. These findings suggest that Python-assisted visualization served as a catalyst for enriching and stabilizing students' concept images.

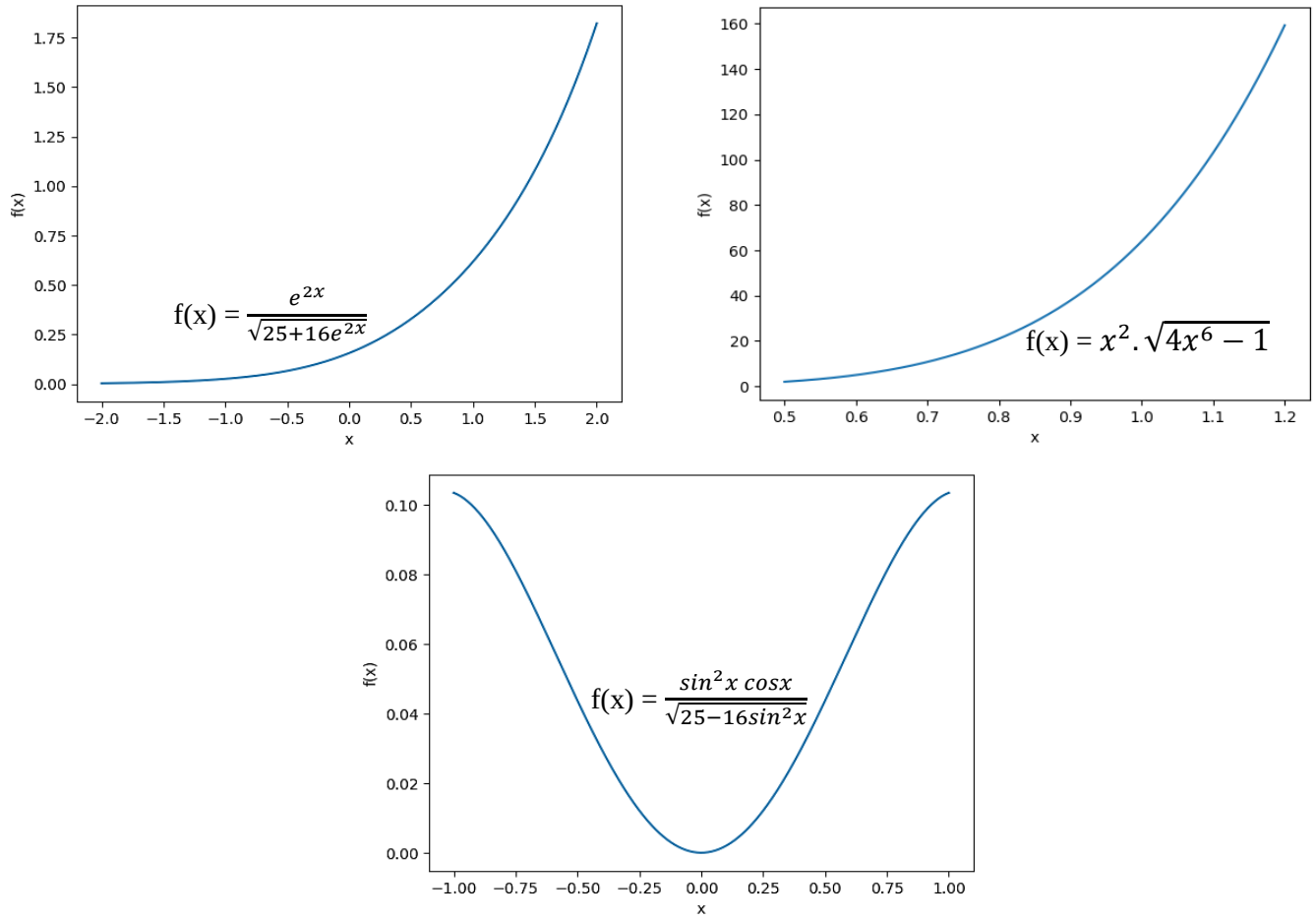


Figure 3: Python-generated visualizations of mathematical functions demonstrating dynamic relationships between symbolic expressions and graphical representations, supporting representational fluidity and concept image development.

4.1.3. Emergence of Instructional Design Principles

A cross-cycle analysis was conducted to identify recurring instructional features associated with successful learning outcomes. Through iterative refinement and theoretical reflection, three instructional design principles consistently emerged across DBR cycles. These principles represent empirically grounded patterns rather than predetermined instructional assumptions.

Table 1: Instructional design principles emerging from Design-Based Research cycles and their associated cognitive functions.

Design Principle	Core Instructional Feature	Cognitive Function
Concept-First Coding	Conceptual exploration precedes programming	Reduces cognitive load; supports schema formation
Representational Fluidity	Continuous transition among symbolic, visual, and computational forms	Strengthens conceptual coherence
Reflective Alignment	Structured reflection linking code output to mathematical meaning	Prevents trial-and-error learning

These principles guided the refinement of instructional tasks in subsequent DBR cycles and form the foundation for the discussion that follows.

4.2. Discussion

The findings of this study demonstrate that Python-integrated mathematics instruction, when deliberately designed through a DBR framework, can systematically transform students' concept images while fostering computational thinking skills. This section interprets the results in relation to existing theory and research, with particular emphasis on the three identified design principles.

4.2.1. Concept-First Coding and Cognitive Load Regulation

The principle of Concept-First Coding emphasizes the importance of introducing programming activities only after students have engaged in initial conceptual exploration. The results indicate that when students encountered mathematical ideas visually and intuitively before writing code, they experienced fewer difficulties with syntactic errors and demonstrated greater persistence in debugging tasks. This finding aligns with Cognitive Load Theory, which suggests that premature exposure to technical complexity can overwhelm working memory and hinder learning (Sweller, 2020).

Unlike syntax-driven programming instruction, Concept-First Coding positions programming as a response to epistemological needs arising from mathematical inquiry. Students understood why code was written, not merely how to write it. This approach supports the construction of meaningful schemas that integrate mathematical reasoning with computational processes.

4.2.2. Representational Fluidity and Concept Image Transformation

The second design principle, Representational Fluidity, underscores the importance of coordinating multiple representational modes. The observed transformation of students' concept images supports the claim that deep mathematical understanding emerges when learners can move flexibly among symbolic, visual, and computational representations without losing meaning (Tall & Vinner, 2022).

Python-based activities required students to continuously translate mathematical ideas into code and graphical outputs. This translation process prevented compartmentalized learning, in which students might excel in symbolic manipulation but struggle with interpretation or application. Instead, representational fluidity fostered conceptual coherence by reinforcing the structural equivalence of different representations.

4.2.3. Reflective Alignment and Metacognitive Development

The principle of Reflective Alignment addresses the risk of superficial learning through unreflective trial-and-error coding. Although students were often able to produce correct outputs, meaningful learning occurred only when they were prompted to compare computational results with formal mathematical definitions and visual intuition. Reflection tasks encouraged students to justify outcomes, identify inconsistencies, and revise their understanding accordingly.

This finding resonates with theories of reflective abstraction, which emphasize that cognitive development occurs through the reorganization of existing knowledge structures rather than mere accumulation of procedures. Reflective Alignment ensured that computational success was accompanied by conceptual validation, thereby strengthening the durability and transferability of learning.

4.2.4. Implications for Mathematics Education and DBR

From a methodological perspective, the DBR framework proved instrumental in uncovering the mechanisms through which computational tools influence mathematical understanding. The iterative nature of DBR enabled the refinement of instructional strategies in response to observed learning challenges, resulting in design principles that are both theoretically grounded and practically applicable.

Pedagogically, the study demonstrates that programming should not be treated as an auxiliary skill but as a representational medium that reshapes how students conceptualize mathematics. By embedding computational thinking within conceptual exploration, educators can create learning environments that promote deeper understanding, abstraction, and problem-solving competence.

5. Conclusion

This study demonstrates that the integration of Python-based programming into undergraduate mathematics instruction can meaningfully transform students' conceptual understanding when it is guided by a Design-Based Research (DBR) framework. Rather than positioning programming as a supplementary technical skill, the findings confirm that computational tools can function as epistemic mediators that reshape how students construct and negotiate mathematical meaning. Across three iterative DBR cycles, students exhibited a clear progression from procedural and static concept images toward more relational and coherent representations. This transformation was supported by the deliberate sequencing of instructional activities, in which computational thinking development preceded and enabled deeper conceptual interpretation through visualization. The empirical evidence indicates that Python programming fostered abstraction and algorithmic reasoning, which subsequently facilitated representational fluency between symbolic expressions and graphical outputs.

A key contribution of this research lies in the articulation of three empirically grounded instructional design principles: Concept-First Coding, Representational Fluidity, and Reflective Alignment. These principles emerged through iterative refinement rather than a priori assumptions, addressing reviewer concerns regarding theoretical grounding and methodological rigor. Collectively, they offer a transferable framework for designing technology-enhanced mathematics instruction that balances conceptual exploration, computational practice, and reflective validation. From a methodological perspective, this study illustrates the value of DBR in bridging theory and

classroom practice. The iterative cycles enabled systematic identification of instructional features that were responsive to observed learning challenges, strengthening the internal coherence between research design, data analysis, and pedagogical outcomes. This alignment responds directly to calls for design research that yields both local instructional improvements and broader theoretical insights. Despite these contributions, the study is not without limitations. The scope of the intervention was confined to a specific mathematical topic and institutional context, which may limit generalizability. Additionally, the qualitative emphasis of the analysis, while suitable for uncovering cognitive processes, suggests the need for complementary quantitative measures in future research. Subsequent studies may extend this work by examining longitudinal impacts, comparing different programming environments, or integrating assessment instruments that capture conceptual change at scale.

In conclusion, this research provides evidence that thoughtfully designed computational integration—anchored in DBR and articulated through clear design principles—can support deeper mathematical understanding and computational thinking development. The findings contribute to the growing discourse on computationally enriched mathematics education and offer practical guidance for educators seeking to align programming activities with conceptual learning goals.

References

Angeli, C., & Giannakos, M. (2020). Computational thinking education: Issues and challenges. Computers in Human Behavior, 105, Article 106185. https://doi.org/10.1016/j.chb.2019.106185

Azra, H. (2025). The effect of digital visualization tools on understanding of non-linear functions in advanced algebra: A mixed methods study in Georgia schools, USA. European Journal of Education and Pedagogy, 6(5), 13–18. https://doi.org/10.24018/ejedu.2025.6.5.991

Bakker, A. (2018). Design research in education. In Design research in education: A practical guide for early career researchers (pp. 3–22). Routledge. https://doi.org/10.4324/9780203701010

Caderheiro, A. C., & Gadanidis, G. (2023). Integrating computational thinking in mathematics education: A conceptual framework. The Journal of Mathematical Behavior, 70, Article 101034. https://doi.org/10.1016/j.jmathb.2023.101034

Cerone, A., Roggenbach, M., Davenport, J., Denner, C., Farrell, M., Haveraaen, M., Moller, F., Koerner, P., Krings, S., Olveczky, P., Schlingloff, B.-H., Shilov, N., & Zhumagambetov, R. (2020). Rooting formal methods within higher education curricula for computer science and software engineering (Version 1). arXiv. https://doi.org/10.48550/arXiv.2010.05708

Chang, L.-C., Lin, H.-R., & Lin, J.-W. (2023). Learning motivation, outcomes, and anxiety in programming courses—A computational thinking–centered method. Education and Information Technologies, 29(1), 545–569. https://doi.org/10.1007/s10639-023-12313-3

dos Santos, J. F., & de Menezes, I. F. (2025). Between facilities and learning: Connecting computational thinking and mathematical skills through Python programming. Revista Internacional de Pesquisa em Educação Matemática, 15(3), 1–19. https://doi.org/10.37001/ripem.v15i3.4528

El Turkey, H., Karakok, G., Cilli-Turner, E., Satyam, V. R., Savić, M., & Tang, G. (2024). A framework to design creativity-fostering mathematical tasks. International Journal of Science and Mathematics Education, 22(8), 1761–1782. https://doi.org/10.1007/s10763-024-10449-3

Falloon, G. (2024). Advancing young students' computational thinking: An investigation of structured curriculum in early years primary schooling. Computers & Education, 216, Article 105045. https://doi.org/10.1016/j.compedu.2024.105045

Fang, X., Ng, D. T. K., & Yuen, M. (2024). Effects of GeoGebra-enhanced Scratch computational thinking instruction on fifth-grade students' motivation, anxiety, and cognitive load. Education and Information Technologies, 30(1), 377–402. https://doi.org/10.1007/s10639-024-13052-9

Hickmott, D., & Prieto-Rodriguez, E. (2022). A scoping review of studies on the integration of computational thinking in undergraduate mathematics. International Journal of Research in Undergraduate Mathematics Education, 8(3), 540–565. https://doi.org/10.1007/s40753-022-00171-4

Hidayah, I., & Sa'dijah, C. (2021). Analisis kemampuan berpikir komputasional siswa dalam menyelesaikan masalah matematika. Jurnal Pendidikan: Teori, Penelitian, dan Pengembangan, 6(3), 450–460. https://doi.org/10.17977/jptpp.v6i3.14610

Johnson, H. L., McClintock, E., & Gardner, A. (2021). Opening possibilities: An approach for investigating students' transfer of

mathematical reasoning. In Research in mathematics education (pp. 59–79). Springer International Publishing. https://doi.org/10.1007/978-3-030-65632-4_3

Kamalia, D., Lesmono, A. D., Handayani, R. D., & Choirrudin, C. (2024). The analysis of students' computational thinking skills through the implementation of GeoGebra integrated student worksheets on motion. Berkala Ilmiah Pendidikan Fisika, 12(2), 245–258. https://doi.org/10.20527/bipf.v12i2.18458

Keller, P., Kaboré, A. K., Plein, L., Klein, J., Le Traon, Y., & Bissyandé, T. F. (2021). What you see is what it means! Semantic representation learning of code based on visualization and transfer learning. ACM Transactions on Software Engineering and Methodology, 31(2), 1–34. https://doi.org/10.1145/3485135

Li, Y., & Schoenfeld, A. H. (2021). Characterizing and developing mathematical thinking: Directions for theory, practice, and policy. Journal on Mathematics Education, 14(1), 1–15. https://doi.org/10.22342/jme.14.1.13456.1-15

Maharani, S., & Nusantara, T. (2020). Pengembangan desain pembelajaran berbasis design-based research untuk meningkatkan kemampuan abstraksi matematis. Jurnal Elemen, 6(2), 215–230. https://doi.org/10.29408/jel.v6i2.2127

McKenney, S., & Reeves, T. C. (2018). Conducting educational design research (2nd ed.). Routledge.

Munns, A. (2016). A study of competence in mathematics and mechanics in an engineering curriculum. European Journal of Engineering Education, 42(6), 1062–1075. https://doi.org/10.1080/03043797.2016.1259292

O'Brien, T. (2024). Grammatical structures in mathematics: A personal view (Version 3). arXiv. https://doi.org/10.48550/arXiv.2410.07569

Park, K., Johnson, J., Peterson, C. S., Yedla, N., Baysinger, I., Aponte, J., & Sharif, B. (2024). An eye tracking study assessing source code readability rules for program comprehension. Empirical Software Engineering, 29(6). https://doi.org/10.1007/s10664-024-10532-x

Prahmana, R. C. I. (2022). Design based research dalam pendidikan matematika. Kanisius.

Prince, T. (2023). Enhancing linear algebra learning through computational thinking: A project-based approach. HETS Online Journal, 13(2), 132–141. https://doi.org/10.55420/2693.9193.v13.n2.129

Rais, D. (2025). Integration of computational thinking through calculus learning using Python programming language: A systematic review and curriculum implications. Journal Research Education Mamminasata, 36–49.

Rais, D., & Xuezhi, Z. (2023). Human cognitive: Learning mathematics through Python programming to support students' problem-solving skills. Anatolian Journal of Education, 8(2), 85–98. https://doi.org/10.29333/aje.2023.826a

Rais, D., & Xuezhi, Z. (2024). Elevating student engagement and academic performance: A quantitative analysis of Python programming integration in the "Merdeka Belajar" curriculum. Journal on Mathematics Education, 15(2), 495–516. https://doi.org/10.22342/jme.v15i2.pp495-516

Sari, M. K., & Kusuma, J. W. (2023). Integrasi Python dalam pembelajaran kalkulus: Dampaknya terhadap concept image mahasiswa. Jurnal Tadris Matematika, 6(1), 89–104. https://doi.org/10.21274/jtm.2023.6.1.89-104

Sung, W., & Black, J. B. (2020). Factors to consider when designing effective learning: Infusing computational thinking in mathematics to support thinking-doing. Journal of Research on Technology in Education, 53(4), 404–426. https://doi.org/10.1080/15391523.2020.1784066

Sweller, J. (2020). Cognitive load theory and educational technology. Educational Technology Research and Development, 68, 1–16. https://doi.org/10.1007/s11423-019-09701-3

Tall, D., & Vinner, S. (2022). Concept image and concept definition: 40 years on. Educational Studies in Mathematics, 109(2), 273–291. https://doi.org/10.1007/s10649-021-10113-w

Wan, H., Zhang, X., Yang, X., & Li, S. (2024). Which approach is effective: Comparing problematization-oriented and structuring-oriented scaffolding in instructional videos for programming education. Education and Information Technologies, 29(14), 17807–17823. https://doi.org/10.1007/s10639-024-12550-0

Wulandari, S., & Surya, E. (2024). Analisis computational thinking pada siswa SMP melalui pemrograman visual dan tekstual. Jurnal Didaktik Matematika, 11(1), 12–28. https://doi.org/10.24815/jdm.v11i1.35241

Ye, H., Ng, O.-L., & Leung, A. (2024). Examining mathematics teachers' creative actions in programming-based mathematical activities. ZDM – Mathematics Education, 56(4), 695–711. https://doi.org/10.1007/s11858-024-01579-7

Zhang, W., Zeng, X., & Song, L. (2024). Towards an assessment model of college students' computational thinking with text-based programming. Education and Information Technologies, 30(2), 1363–1385. https://doi.org/10.1007/s10639-024-12872-z

Ziatdinov, R., & Valles Jr., J. R. (2022). Synthesis of modeling, visualization, and programming in GeoGebra as an effective approach for teaching and learning STEM topics. Mathematics, 10(3), 398. https://doi.org/10.3390/math10030398