2026 ASEE ANNUAL CONFERENCE June 21–24, 2026 | Charlotte Convention Center, NC
Host Sponsor UNC Charlotte
ASEE AMERICAN SOCIETY FOR ENGINEERING EDUCATION
Paper ID #51943

Literature Review: Using Python to Teach Computational Thinking

Mr. Thomas Rossi, University of New Haven

Thomas Rossi is the Assistant Chair of the University of New Haven's Electrical and Computer Engineering and Computer Science department, he also serves as the coordinator for BS in Computer Science program. His research focuses on improving the post-secondary experience for students through the use of current computing tools and technologies. Thomas graduated with his MS in Computer Science from the University of New Hampshire in 2016. He has previously worked at the Rochester Institute of Technology and at Penn State Erie, the Behrend College.

Ren Oberdorfer, University of New Haven

Literature Review: Using Python to Support Computational Thinking Development

1 Abstract

Computational thinking (CT), characterized by decomposition, abstraction, pattern recognition, and algorithmic thinking, is increasingly recognized as essential for real-world problem solving across disciplines. Programming is commonly used to support CT development; however, for novice learners, the syntactical complexity of many text-based programming languages can impose a significant cognitive burden, undermining the instructional goal of fostering computational thinking through experiential learning. While block-based programming environments have been proposed to address this issue, concerns remain regarding transfer to authentic programming practices.

This paper presents a focused comparative synthesis of empirical Python-based instructional interventions supporting computational thinking development. The review synthesizes a focused corpus of empirical classroom-based studies selected using explicit inclusion criteria, spanning K–12 and higher education contexts across multiple educational settings. Distinctions are drawn between Python integration in K–12 environments, where instruction is typically highly scaffolded through game-based, physical, or constructionist approaches, and post-secondary contexts, where Python is more often used as a general-purpose programming language emphasizing problem-oriented learning.

Across contexts, findings indicate that Python-based instruction consistently supports gains in computational thinking concepts and algorithmic reasoning; however, outcomes related to computational thinking practices and dispositions are strongly influenced by pedagogical design, teacher expertise, student preparedness, and access to technological resources. Based on these

findings, the paper synthesizes pedagogical considerations for integrating Python to support computational thinking development and identifies limitations and directions for future research.

2 Introduction

Computational Thinking (CT) has emerged as a critical competency for learners in the digital age, extending beyond computer science to domains such as science, mathematics, and engineering [1 , 2]. CT is commonly characterized by skills including decomposition, abstraction, algorithm design, and evaluation, enabling learners to formulate and solve problems in a systematic manner [3]. Programming is frequently positioned as a primary mechanism for developing these skills, as it requires learners to externalize problem-solving processes in executable form [4].

Despite its pedagogical potential, programming instruction presents persistent challenges for novice learners. Prior research has shown that beginners often struggle with syntax, program structure, and the translation of abstract reasoning into functioning code, leading to increased cognitive burden and frustration [5]. To mitigate these challenges, educational practice has increasingly incorporated block-based programming environments, physical computing tools, and game-based platforms [6]. While these approaches reduce syntactic complexity, they may limit learners' exposure to authentic programming constructs and impede transfer to text-based programming [7].

Python has been proposed as an instructional alternative that balances syntactic simplicity with expressive power. Its readable syntax, broad applicability, and widespread use in industry and academia make Python a compelling candidate for computational thinking instruction across educational levels [8]. However, empirical research on Python-based CT instruction remains

fragmented across K–12 and higher education contexts, employing varied pedagogical approaches and assessment strategies.

This paper addresses this gap through a focused comparative synthesis of empirical Python-based instructional interventions. Rather than aiming for comprehensive coverage of the field, the review emphasizes depth of analysis across a curated corpus of empirical studies. By comparing instructional contexts, pedagogical strategies, and reported CT outcomes, the paper seeks to clarify how Python supports computational thinking development under differing educational conditions.

3 Conceptual Background

This section outlines the conceptual foundations that frame the present synthesis. It first defines computational thinking as a multidimensional construct and then examines the instructional implications of programming language design, particularly with respect to syntax and cognitive load. Together, these perspectives provide the theoretical grounding for analyzing Python-based instructional interventions across educational contexts.

3.1 Computational Thinking

Computational Thinking (CT) has emerged as a foundational competency for learners across science, technology, engineering, and mathematics disciplines [1, 2]. Wing characterizes CT as a problem-solving process that involves formulating problems and expressing solutions in ways that can be executed by an information-processing agent [1, 3]. Subsequent scholarship has expanded this definition to emphasize decomposition, abstraction, algorithm design, and systematic evaluation [2, 4].

Brennan and Resnick propose a multidimensional framework distinguishing computational concepts (e.g., loops, conditionals), computational practices (e.g., testing, debugging, iteration), and computational perspectives (e.g., expressing and connecting through computation) [6]. This framework is particularly useful for analyzing instructional interventions because it differentiates between conceptual mastery and applied or dispositional dimensions of CT. In this paper, CT is treated as a multidimensional construct encompassing conceptual understanding, structured problem solving, and emerging computational practices.

3.2 Programming, Syntax, and Instructional Constraints

Although programming is frequently positioned as a primary vehicle for CT development, novice learners encounter substantial challenges when first learning to code. Research on novice programmers identifies persistent difficulties related to syntax errors, semantic misunderstandings, and the translation of abstract reasoning into executable programs [5]. These challenges increase cognitive load and may obscure underlying problem-solving objectives. Block-based programming environments have been introduced to mitigate syntactic burden by constraining available operations and preventing many types of syntax errors [7]. However, comparisons between block-based and text-based programming highlight trade-offs in authenticity and transferability [7]. While block environments support early engagement, they may limit exposure to professional programming structures.

Python occupies an intermediate instructional position. Compared to lower-level languages, Python reduces syntactic overhead while preserving authentic programming constructs such as functions, conditionals, and iterative control structures [8]. This balance makes Python a

compelling candidate for CT instruction across grade levels, particularly when instructional design intentionally addresses cognitive load and scaffolding

4 Methodology: Literature Review Design

This study employs a focused comparative synthesis of empirical research examining the use of Python as an instructional language to support computational thinking (CT) development in formal educational contexts. Rather than aiming to provide an exhaustive systematic review, the study is designed to identify and analyze representative instructional interventions in order to surface recurring pedagogical patterns, constraints, and outcomes across educational levels.

The conceptual foundation for the analysis is grounded in established computational thinking frameworks [1]–[6], while the analytical corpus consists exclusively of empirical studies in which Python is used as a primary instructional tool. The final dataset includes nine studies spanning primary, middle school, secondary, undergraduate, and teacher education contexts, enabling cross-context comparison of instructional design and CT outcomes.

4.1 Inclusion Criteria

Studies were identified through structured searches conducted in ERIC, Scopus, and Google Scholar. Search queries combined keywords such as “Python programming,” “computational thinking,” “programming education,” “K–12,” and “higher education.” Searches were limited to peer-reviewed publications published between 2014 and 2024 to ensure relevance to contemporary instructional practices.

The initial search yielded approximately 40 studies. Titles and abstracts were screened for relevance to classroom-based instructional interventions involving Python. Following this stage, approximately 20 studies were retained for full-text review. Application of the inclusion criteria resulted in a final corpus of nine studies.

Studies were included if they:

- Implemented Python as the primary programming language
- Were conducted in formal educational settings (K–12 or post-secondary)
- Reported empirical outcomes related to computational thinking, problem solving, or related constructs
- Represented classroom-based instructional interventions

Studies were excluded if they:

- Focused solely on theoretical discussion without empirical data
- Used Python only as a secondary or supporting tool
- Fell outside the defined publication window
- Did not report measurable learning or instructional outcomes

This selection process prioritizes depth of analysis and comparability across instructional contexts rather than comprehensive coverage of the field.

4.2 Analytical Framework

Each study in the final corpus was analyzed using a structured comparative framework capturing the following dimensions: educational level, instructional model, role of Python within the learning environment, computational thinking dimensions assessed, key findings, and implementation constraints.

Cross-study patterns were identified through qualitative comparative analysis, with particular attention to how instructional design features, such as scaffolding, task structure, and contextual integration, influence computational thinking outcomes. This approach enables identification of recurring trends and trade-offs across diverse educational settings while maintaining alignment with established CT frameworks.

5 Synthesis of Python-Based CT Instruction

Several empirical studies have examined Python-based instructional interventions for computational thinking across K–12 and higher education contexts [8]–[17]. Table I summarizes the instructional strategies and outcomes reported in this focused corpus.

Table 1: Summary of Python-Based Instructional Interventions for Computational Thinking Development: Primary-Secondary (K-12)

Study	Instructional Model	Role of Python	CT Dimensions Assessed	Key Findings	Constraints
[9]	Game-based (CodeCombat)	Embedded in scaffolded, narrative tasks	Algorithmic thinking; creativity; problem solving	Significant gains in algorithmic thinking and problem solving	Short duration; limited collaboration measures

[10]	Physical Programming (Micro:bit)	Controls physical artifacts	CT skills; self-efficacy	Improved CT skills and computational self-efficacy	Hardware dependency; requires teacher expertise
[11]	Problem-Oriented Learning (POL)	Primary language for structured problem-solving tasks	CT skills; practices	Outperformed lecture-based instruction in CT outcomes	Limited gains in CT practices; short intervention
[16]	Gamified Instruction	Transitional text-based programming in a gamified environment	Algorithmic reasoning; engagement	Increased engagement and CT-related performance	Short implementation window
[17]	Curriculum-Integrated Programming	Embedded across grade-level CT progression	Decomposition; abstraction	Evidence of CT progression across grades	Assessment and contextual variability

Table 2: Summary of Python-Based Instructional Interventions for Computational Thinking Development: Post-Secondary

Study	**Instructional Model**	**Role of Python**	**CT Dimensions Assessed**	**Key Findings**	**Constraints**
[8]	Structured Programming Course	General-purpose programming for problem solving	Problem solving; structured reasoning	Improved problem-solving performance	Syntax challenges for novice learners

[12], [13]	Staged Learning Trajectory / ICT Integration	Scaffolded progression from basics to algorithm design	Decomposition; pattern recognition; algorithmic thinking	Supports systematic CT development	Implementation complexity; initial syntax barriers
[14]	Domain-Integrated Modeling	Data processing and disciplinary applications	Problem solving; abstraction	Improved application of CT in domain contexts	Instructor-dependent variability
[15]	Robotics-Mediated Instruction	Controls robotic systems	Abstraction; debugging; applied problem solving	Strengthened applied CT skills and engagement	Resource-intensive implementation

As shown in Tables 1 & 2, Python-based instructional interventions vary substantially across educational contexts. In K–12 contexts represented in the reviewed studies [9], [10], Python is implemented within scaffolded environments such as game-based platforms and physical computing activities. In higher education contexts, Python is positioned as a general-purpose programming language, emphasizing problem-oriented learning and disciplinary application. Across the reviewed studies [8-17], gains are most consistently reported in algorithmic thinking and problem-solving performance. Evidence regarding broader CT practices and dispositions remains more limited, as measurement approaches vary across studies.

5.1 Primary and Middle School Contexts

At the primary level, Choi and Choi [9] examined CodeCombat, a serious game embedding Python within scaffolded narrative tasks. Significant gains were observed in algorithmic

thinking, creativity, and problem solving. The structured progression and immediate feedback appear to reduce cognitive overload while maintaining engagement.

In middle school contexts, Bai et al. [11] implemented a problem-oriented learning (POL) model in a Python course. Students exposed to POL demonstrated stronger CT gains compared to lecture-based instruction, particularly in structured algorithmic reasoning. However, improvements in broader CT practices were more modest.

Yurdakök and Kalelioğlu [10] investigated physical programming using Micro:bit devices. Python-controlled artifacts linked abstract code to tangible outputs, producing gains in CT skills and computational self-efficacy. Implementation, however, depended on hardware availability and teacher expertise.

5.2 Gamification and Curriculum Integration

Gamified high school programming interventions [16] indicate that embedding Python within motivational structures can enhance engagement and support algorithmic reasoning. Similarly, K–12 mapping research [17] demonstrates that programming-integrated curricula support progression in decomposition and abstraction across grade levels, though variability in assessment limits strong causal inference.

5.3 Undergraduate and Non-STEM Contexts

Lee and Cho [8] report measurable improvements in problem-solving performance following Python instruction. Although the study emphasizes problem-solving rather than CT explicitly, the structured reasoning improvements align with core CT constructs [1], [6].

Robotics-mediated interventions in non-STEM undergraduate contexts [15] demonstrate that controlling robotic systems via Python supports abstraction and iterative debugging practices. These environments strengthen applied CT skills but require substantial instructional and material resources.

5.4 Teacher Education Contexts

Teacher education studies expand Python's instructional role. The integration of Python into science teacher education [14] supports computational modeling and domain-specific problem solving. Similarly, the Portuguese mathematics teacher education study [12] highlights the complexities of integrating CT with technological pedagogical content knowledge.

The staged learning trajectory model using Google Colab [13] explicitly targets decomposition, pattern recognition, abstraction, and algorithmic thinking. Structured progression from basic operations to conditional logic and function design supports systematic CT development in prospective teachers.

5.5 Cross-Study Patterns

Across all nine empirical studies [8 - 17], algorithmic thinking and structured problem solving show the most consistent gains. Scaffolded instructional models—whether gamified [9], physical [10], robotics-mediated [15], or staged trajectory-based [13]—appear particularly effective for novice learners. However, syntax-related challenges persist in less scaffolded contexts [8], and long-term transfer beyond short intervention periods remains underexplored.

6 Discussion and Pedagogical Implications

The findings from the reviewed studies indicate that Python's effectiveness in supporting computational thinking (CT) development is shaped primarily by instructional design rather than by language characteristics alone. Across K–12 contexts, scaffolded implementations, such as game-based environments [9], physical programming activities [10], and gamified instructional models [16], consistently report gains in algorithmic thinking and structured problem solving. These approaches reduce cognitive load by constraining task complexity, providing immediate feedback, and supporting incremental progression, which appears particularly beneficial for novice learners.

In contrast, post-secondary and teacher education contexts position Python as a general-purpose programming and modeling tool [8], [12]–[15]. In these settings, learners engage more directly with abstraction, data processing, and domain-specific problem solving. Structured approaches, such as staged learning trajectories [12], [13] and domain-integrated modeling [14], demonstrate that explicit sequencing of computational concepts supports systematic CT development. However, reduced scaffolding in these contexts is associated with persistent syntax-related challenges, particularly for learners without prior programming experience [8].

A consistent pattern across the reviewed studies [8]–[17] is that gains are most evident in algorithmic thinking and structured problem solving. Evidence related to broader CT practices—such as debugging, iteration, and reflective evaluation, and dispositions such as confidence and persistence is more variable. This variability appears to stem from differences in assessment approaches and the short duration of many interventions, rather than from the absence of these outcomes. As a result, programming alone does not appear sufficient to develop CT practices; these must be explicitly targeted through instructional design.

From a pedagogical perspective, these findings suggest that Python should be integrated as part of a structured instructional approach rather than treated as a standalone solution for CT development. In K–12 environments, scaffolded and transitional models, such as gamified environments [9], [16] and physical computing contexts [10], support early engagement and reduce cognitive barriers. In post-secondary and teacher education settings, structured progression and domain-integrated tasks [12]–[14] support deeper abstraction and application. Across contexts, aligning instructional tasks with learner readiness and explicitly targeting CT components appears critical for effective implementation.

Implementation conditions also play a significant role in determining outcomes. Several studies highlight dependencies on teacher expertise, curricular coherence, and access to technological resources [10], [12], [15]. These factors influence not only the fidelity of implementation but also the extent to which learners can engage meaningfully with computational tasks. As such, successful integration of Python into CT instruction requires institutional support, including professional development and access to appropriate tools and infrastructure.

Taken together, the reviewed studies suggest that three design variables consistently mediate the effectiveness of Python-supported CT instruction: (1) the degree of instructional scaffolding, (2) the explicit targeting of CT components, and (3) the alignment between task complexity and learner readiness. Higher levels of scaffolding appear necessary in K–12 contexts to support foundational skills, while structured progression and domain integration become more important in advanced settings. These patterns reinforce that Python's instructional value emerges through the interaction between pedagogy, learner characteristics, and task design.

In engineering education contexts, these findings have direct implications for introductory programming and modeling courses. First-year engineering students often encounter programming alongside disciplinary coursework, creating cognitive demands similar to those observed in K–12 settings. The reviewed studies suggest that scaffolded progression and problem-oriented learning approaches [11], [13] may reduce cognitive overload and support conceptual understanding. For non-computer science majors, robotics-mediated and domain-integrated applications [14], [15] provide a pathway for connecting computational thinking to authentic engineering problems. These findings indicate that Python adoption in engineering curricula should be accompanied by intentional instructional design rather than treated as a neutral substitution for other programming languages.

7 Limitations and Future Work

This synthesis is limited by the size and scope of the empirical corpus. Although nine intervention studies were analyzed, the sample does not represent the full breadth of Python-based computational thinking research. The included studies span diverse educational contexts and instructional models, which strengthens comparative insight but also introduces variability in implementation and assessment.

A second limitation concerns measurement. The reviewed studies employ heterogeneous instruments to assess computational thinking, problem solving, and self-efficacy, making direct comparison difficult. Differences in how CT dimensions are operationalized, particularly computational practices and dispositions, limit the strength of cross-study generalizations.

Additionally, most interventions are short-term and classroom-bound. Few studies provide longitudinal data examining sustained CT development or transfer beyond the immediate instructional context. As a result, conclusions drawn in this synthesis reflect short- to medium-term outcomes rather than long-term impact.

Future research should prioritize longitudinal designs, more standardized CT measurement frameworks, and deeper examination of transfer across disciplines. Greater attention to implementation conditions, teacher professional development, and equity in resource-constrained environments would further strengthen the evidence base for Python-supported CT instruction.

References

[1] J. M. Wing, “Computational thinking,” Commun. ACM, vol. 49, no. 3, pp. 33–35, 2006, doi: 10.1145/1118178.1118215.

[2] V. Barr and C. Stephenson, “Bringing computational thinking to K–12,” ACM Inroads, vol. 2, no. 1, pp. 48–54, 2011, doi: 10.1145/1929887.1929905.

[3] J. M. Wing, “Computational thinking: What and why?” The Link Magazine, Carnegie Mellon University, Pittsburgh, PA, USA, Spring 2010. [Online]. Available: https://www.cs.cmu.edu/~15110-s13/Wing06-ct.pdf

[4] S. Grover and R. Pea, “Computational thinking in K–12: A review of the state of the field,” Educ. Researcher, vol. 42, no. 1, pp. 38–43, 2013, doi: 10.3102/0013189X12463051.

[5] E. Lahtinen, K. Ala-Mutka, and H. Järvinen, “A study of the difficulties of novice programmers,” SIGCSE Bull., vol. 37, no. 3, pp. 14–18, 2005, doi: 10.1145/1151954.1067453.

[6] K. Brennan and M. Resnick, “New frameworks for studying and assessing computational thinking,” in Proc. 2012 Annu. Meeting Amer. Educ. Res. Assoc. (AERA), Vancouver, BC, Canada, 2012. [Online]. Available: http://scratched.gse.harvard.edu/ct/files/AERA2012.pdf

[7] B. Weintrop and U. Wilensky, “To block or not to block: A comparison of blocks-based and text-based programming,” in Proc. 11th ACM Conf. Int. Comput. Educ. Res. (ICER), Omaha, NE, USA, 2015, pp. 199–208, doi: 10.1145/2787622.2787729.

[8] J. Lee and Y. Cho, “The impact of Python programming education on problem-solving skills,” *J. Educ. Comput. Res.*, vol. 55, no. 4, pp. 1–20, 2017, doi: 10.1177/0735633116676059.

[9] W. C. Choi and I. C. Choi, “The influence of CodeCombat on computational thinking in Python programming learning at primary school,” in Proc. ICEDS, 2024, doi:10.1145/3669947.3669951.

[10] E. A. Yurdakök and F. Kalelioğlu, “The effect of teaching physical programming on computational thinking skills and self-efficacy perceptions,” J. Educ. Comput. Res., vol.62 no. 3, 2024, doi:10.1177/07356331231220313.

[11] H. Bai, X. Wang, and L. Zhao, “Effects of the problem-oriented learning model on middle school students’ computational thinking skills in a Python course,” Front. Psychol., vol. 12, 2021, doi: 10.3389/fpsyg.2021.771221.

[12] J. M. Dos Santos, J. Carvalho e Silva, and Z. Lavicza, “Fostering computational thinking and ICT integration in mathematics teacher education: A two-cycle study in Portugal,” Electron. J. Math. Technol., vol. 19, no. 2, 2024.

[13] E. Irawan, M. K. Huda, and R. Purwasih, “A learning trajectory for developing computational thinking in prospective mathematics teachers through Python programming in Google Colab,” Alifmatika, vol. 7, no. 1, pp. 34–52, 2025., doi: 10.35316/alifmatika.2025.v7i1.34-52

[14] K. Batı, “Integration of Python into Science Teacher Education, Developing Computational Problem Solving and Using Information and Communication Technologies Competencies of Pre-service Science Teachers,” *Informatics in Education*, vol. 21, no. 2, pp. 235–251, 2022, doi: 10.15388/infedu.2022.12.

[15] V. Macko, P. Felber, K. Bergram and A. Holzer, "Using Educational Robotics to Support Active Learning Experiences and Foster Computational Thinking Skills among Non-STEM University Students," *2023 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE)*, Auckland, New Zealand, 2023, pp. 1-8, doi: 10.1109/TALE56641.2023.10398343.

[16] Z. Qu, J. Liu, L. Che, Y. Su and W. Zhang, "Research on the Application of Gamification Programming Teaching for High School Students' Computational Thinking Development," 2023 IEEE 12th International Conference on Educational and Information Technology (ICEIT), Chongqing, China, 2023, pp. 144-149, doi: 10.1109/ICEIT57125.2023.10107843.

[17] C. Tikva and E. Tambouris, "Mapping computational thinking through programming in K–12 education: A conceptual model based on a systematic literature review," *Computers & Education*, vol. 162, 2021, doi: 10.1016/j.compedu.2020.104083.