

EFFECTIVENESS OF VISUAL PROGRAMMING AND TEXT-BASED PROGRAMMING IN DEVELOPING STUDENTS' COMPUTATIONAL THINKING SKILLS

Khatrin Juliani Taku Neno¹, Floriana Letni Banung²

Pendidikan Informatika, Citra Bangsa University, Kupang, Indonesia

khatrintakuneno96@gmail.com¹, florianaletnibanung21@gmail.com²

RESEARCH ARTICLE | DOI: <https://doi.org/10.37792/tgk61021>

Received: September 17, 2026 | Revised: September 30, 2026 | Accepted: September 30, 2026 | Published: September 2026

License: Creative Commons Attribution 4.0 International (CC BY 4.0)

ABSTRACT

Computational thinking is an essential skill in 21st-century education because it supports logical, systematic, and structured problem solving. This study compared visual programming using Scratch and text-based programming using Python in developing students' computational thinking skills. A quantitative quasi-experimental pretest-posttest comparison-group design was conducted at SMK N 4 Kupang with 25 students in Class X DPK and 24 students in Class X DKV 1. Each group received one instructional session scheduled for 2×45 minutes, with the same researcher serving as instructor. Both groups learned standard algorithms and the Stack data structure through equivalent learning objectives, materials, activities, tasks, instructional sequence, duration, and assessment procedures. Data were collected using an essay-format computational-thinking test and classroom observations. Descriptive statistics and N-Gain scores were used to describe learning improvement, while an independent-samples t-test was used to compare N-Gain scores. The visual programming group increased from a mean pretest score of 45.40 to 88.13, with an average N-Gain of 0.78 (High), whereas the text-based group increased from 48.00 to 72.50, with an average N-Gain of 0.49 (Medium). The between-group comparison yielded $t(47) = 8.12$, $p < .001$, with Cohen's $d = 2.37$. Classroom observations indicated more structured use of visual blocks for algorithm construction in the Scratch group and more syntax-focused trial-and-error behavior in the Python group. The findings are interpreted within the context of two intact classes at one school.

Keywords: *computational thinking, coding education, visual programming, text-based programming, Scratch, Python*

I. INTRODUCTION

The rapid advancement of digital technology in the 21st century demands that students not only possess basic technology literacy but also demonstrate logical, systematic, critical, and structured problem-solving abilities. Within Informatics education, computational thinking is a core competency. Computational thinking involves understanding complex problems, breaking them into manageable components, recognizing patterns, abstracting relevant details, and formulating solutions as sequential algorithms. Recent empirical studies

indicate that students' computational thinking skills can remain at moderate levels, with abstraction and algorithm design among the challenging dimensions.

Coding education can provide an effective vehicle for fostering computational thinking skills. Through programming activities, students translate problems into executable logic, test potential solutions, debug errors, and refine their reasoning. However, choosing an appropriate programming paradigm for beginners remains an important pedagogical consideration. Visual programming, such as Scratch, offers an intuitive block-based interface that reduces the need to manage programming syntax during initial learning. Conversely, text-based programming, such as Python, provides an authentic coding environment that requires precise syntax, variable declarations, and control structures.

Prior studies have often examined visual programming tools independently or developed Scratch- and Code.org-based learning activities. For example, Findayani et al. (2023) examined a Scratch-assisted project-based module, while Budiyanto et al. (2025) investigated computational thinking in Code.org activities. The present study differs by directly comparing visual and text-based programming in two intact secondary-school classes while keeping the learning content centered on equivalent standard algorithms and the Stack concept. This head-to-head comparison is intended to separate the programming environment from the core computational content as far as the quasi-experimental classroom setting permits.

Accordingly, this study aims to: (1) examine the improvement in computational thinking following visual programming instruction using Scratch; (2) examine the improvement following text-based programming instruction using Python; and (3) compare the learning gains of the two groups in a comparable novice learning context at SMK N 4 Kupang.

II. LITERATURE REVIEW

Computational thinking was popularized in educational and computing discourse by Wing (2006), who described it as a fundamental way of approaching problems through computational concepts and methods. In this study, the construct is operationalized through four dimensions: decomposition, pattern recognition, abstraction, and algorithm design. Mustahib et al. (2023) and Salim et al. (2024) reported challenges in students' computational thinking, particularly in higher-order dimensions such as abstraction and algorithmic formulation.

Coding activities provide a hands-on environment for reinforcing these dimensions. Sonjaya and Munir (2025) noted that block-based visual environments can support foundational programming comprehension and computational skills by reducing syntax-related obstacles. Findayani et al. (2023) and Budiyanto et al. (2025) also reported positive learning outcomes associated with Scratch- and Code.org-based environments. Maharani et al. (2023) further demonstrated the relevance of computational-thinking-based learning materials for developing students' mathematical literacy.

Transitioning directly to text-based programming may require beginners to coordinate problem-solving logic with syntax, punctuation, data types, and language semantics. This study therefore compares the two environments under the same broad content focus: standard algorithms and a Stack data structure applied to a book collection and distribution scenario. The novelty lies in the direct comparison of the two programming paradigms using equivalent learning content and a common computational-thinking assessment rather than evaluating each platform in isolation.

III. METHODOLOGY

This study employed a quantitative quasi-experimental pretest-posttest comparison-group design. Because the study used intact classes, the groups were not randomly assigned at the individual-student level.

Participants and Group Selection

The research was conducted at SMK N 4 Kupang. The sample consisted of Class X DPK (N = 25), assigned as the visual programming group and taught using Scratch, and Class X DKV 1 (N = 24), assigned as the text-based programming group and taught using Python. The two classes were selected as intact classes within the school's existing class organization and research access. No individual random assignment was conducted. A pretest was administered to assess baseline computational-thinking performance. Based on the reported pretest means and standard deviations, the independent-samples comparison was approximately $t(46.93) = -0.80$, $p = .430$, indicating no statistically significant difference in baseline scores between the two classes.

The baseline comparison indicated that the two groups did not differ significantly on the pretest based on the reported summary statistics (approximately $t(46.93) = -0.80$, $p = .430$). Thus, the groups had comparable baseline computational-thinking performance at the beginning of the intervention, although the use of intact classes means that other pre-existing class-level differences cannot be completely ruled out.

Intervention Procedure and Instructional Equivalence

The intervention was conducted in one instructional session for each group, scheduled for 2×45 minutes. The researcher served as the instructor for both Scratch and Python groups. The intervention itself was reported as 60 minutes within the scheduled 2×45 -minute lesson period. Both groups addressed the same learning topic: standard algorithms incorporating the concept of a Stack data structure, applied to the same book collection and distribution problem context. The pretest was administered before instruction and the same posttest procedure was administered after instruction. Instructional equivalence was maintained through the same learning objectives, instructional materials, problem context, learning activities, tasks, instructional sequence, duration, and assessment procedures. The learning sequence consisted of concept explanation, problem presentation, guided practice, task implementation, debugging or refinement of the solution, and evaluation. The visual group implemented the equivalent tasks using Scratch block-based programming, whereas the text-based group implemented them using Python. Thus, the programming environment was the principal instructional difference between the two groups.

The same instructor, learning objectives, instructional materials, problem context, learning activities, tasks, instructional sequence, and assessment procedures were used for both groups. The groups differed primarily in the programming environment used to implement the equivalent tasks: Scratch for visual programming and Python for text-based programming.

Instrument and Data Collection

The computational-thinking instrument consisted of 20 essay-format items designed to measure decomposition, pattern recognition, abstraction, and algorithm design. Item validity was examined separately for the pretest and posttest data in both groups. In the control/text-based group, all 20 items were classified as valid, with pretest item correlations ranging from 0.485 to 0.838 and posttest item correlations ranging from 0.418 to 0.792. In the experimental/visual group, all 20 items were also classified as valid, with pretest item

correlations ranging from 0.487 to 0.816 and posttest item correlations ranging from 0.434 to 0.916. Reliability was very high in both groups: for the control/text-based group, $r_{11} = 0.940$ for the pretest and 0.923 for the posttest; for the experimental/visual group, $r_{11} = 0.933$ for the pretest and 0.946 for the posttest. In each case, the coefficient exceeded the reported critical value of 0.700 and was classified as reliable with a very high category. The instrument therefore met the reported validity and reliability criteria for use in the study.

Observation Procedure

Classroom observations were conducted by the researcher during the instructional session. Observations were organized into three stages. In the pre-coding/planning phase, the focus was on how students decomposed and abstracted the problem before constructing a program, including whether they prepared an algorithm sketch, flowchart, or pseudocode and whether they understood the Push and Pop operations in the book Stack context. In the implementation phase, the focus was on students' interaction with the programming platform: Scratch students' fluency in selecting and arranging logic blocks and their discussion of logical concepts with the instructor, and Python students' attention to logical structure versus syntax-related difficulties such as punctuation, indentation, and data types. In the debugging/problem-solving phase, the focus was on students' strategies when errors occurred, including trial-and-error behavior and structured analysis of the location and cause of errors. The observation findings were summarized descriptively and used to contextualize the quantitative results rather than to establish causal mechanisms.

Data Analysis

Descriptive statistics included the mean, standard deviation, minimum, and maximum scores. Learning improvement was quantified using the normalized gain (N-Gain): $g = (\text{posttest} - \text{pretest}) / (\text{maximum score} - \text{pretest})$. N-Gain values were interpreted as High ($g \geq 0.70$), Medium ($0.30 \leq g < 0.70$), and Low ($g < 0.30$), following Hake (1998).

An independent-samples t-test was used to compare the N-Gain scores of the two groups. The reported normality results were mixed: for the control/text-based group, the Shapiro-Wilk test yielded $W = 0.908$, $p = .028$ for the pretest and $W = 0.928$, $p = .078$ for the posttest; for the experimental/visual group, $W = 0.926$, $p = .080$ for the pretest and $W = 0.899$, $p = .021$ for the posttest. Thus, normality was not uniformly satisfied across the pretest and posttest scores. The reported homogeneity test yielded $F = 0.290$, $p = .593$, indicating homogeneous variance under the stated test criterion. The between-group comparison of N-Gain scores yielded $t(47) = 8.12$, $p < .001$, with Cohen's $d = 2.37$. The p-value is reported as $p < .001$ based on the reported t statistic and 47 degrees of freedom. Because the normality results supplied by the researcher concern pretest and posttest scores rather than the N-Gain distribution itself, the inferential result is interpreted cautiously. Qualitative observation data were summarized descriptively to identify recurring patterns in students' planning, coding, and debugging behaviors.

Ethics and Consent

This study was conducted in accordance with ethical standards for educational research involving human participants. Institutional permission and administrative approval were granted by the Principal of SMK N 4 Kupang. Prior to data collection, informed consent was obtained from the participating students and their legal guardians, detailing the study objectives, the voluntary nature of participation, and assurances regarding data confidentiality and anonymity. All test performance data and observational records were anonymized and used strictly for academic research purposes.

IV. RESULTS

The descriptive findings for pretest, posttest, and N-Gain scores across the two instructional groups are summarized in Table 1.

Table 1. Summary of Pretest, Posttest, and N-Gain Results

Instructional Group	N	Pretest Mean $\pm$ SD	Posttest Mean $\pm$ SD	Average N-Gain	Category
Visual Programming (Scratch)	25	45.40 $\pm$ 11.89	88.13 $\pm$ 6.94	0.78	High
Text-Based Programming (Python)	24	48.00 $\pm$ 10.98	72.50 $\pm$ 6.00	0.49	Medium

The visual programming group increased from a mean pretest score of 45.40 to a mean posttest score of 88.13, with an average N-Gain of 0.78 (High). The text-based programming group increased from 48.00 to 72.50, with an average N-Gain of 0.49 (Medium).

The inferential analysis showed a statistically significant difference in N-Gain scores between the two groups, $t(47) = 8.12$, $p < .001$, with Cohen's $d = 2.37$. The reported homogeneity test yielded $F = 0.290$, $p = .593$. The normality results for the pretest and posttest scores were mixed, with $p < .05$ for the control-group pretest and experimental-group posttest. Because the supplied normality results do not specifically report the N-Gain distributions, the between-group t-test result is interpreted with appropriate caution.

The pretest comparison based on the reported descriptive statistics yielded approximately $t(46.93) = -0.80$, $p = .430$, indicating no statistically significant baseline difference. This supports baseline comparability between the two intact classes in terms of their reported pretest performance.

Instructional Observation Findings

- Visual Programming Group: Students used visual blocks to construct program logic and interacted with the instructor while refining algorithm sequences and debugging errors. The observations suggested that the block-based representation made the sequence of the Stack procedure easier to inspect during practice.
- Text-Based Programming Group: Students were frequently observed editing code directly until execution succeeded, with substantial attention directed to syntax-related errors. This pattern was observed during classroom activities and should be interpreted as an observational description rather than as proof that syntax alone caused the slower learning gain.

V. DISCUSSION

The quantitative results show a larger mean learning gain for the Scratch group than for the Python group in the participating classes, and the between-group comparison of N-Gain scores was statistically significant ($t(47) = 8.12$, $p < .001$, Cohen's $d = 2.37$). One plausible interpretation is that the block-based interface reduced the immediate burden of syntax management, allowing students to focus more directly on problem decomposition, logical sequencing, and algorithm construction. This interpretation is consistent with the classroom observations and with prior literature on block-based programming (Sonjaya & Munir, 2025;

Budiyanto et al., 2025), but the quasi-experimental design does not establish syntax reduction as the sole causal mechanism.

The Python group frequently displayed syntax-focused trial-and-error behavior during classroom activities. This may have competed with time available for planning and abstraction, particularly while students were learning Stack operations. However, statements concerning working memory or dual cognitive load should be treated as interpretations suggested by the observed behavior unless supported by direct cognitive-load measures. The present study did not measure cognitive load directly.

The findings therefore support the pedagogical possibility that visual programming can provide useful scaffolding for novice learners when the instructional objective is to develop foundational computational-thinking processes. The results should not be interpreted as evidence that visual programming is universally superior to text-based programming; rather, they describe the outcomes observed when two intact classes learned comparable algorithmic content through Scratch and Python.

Limitations

This study has several limitations. First, the sample was small (49 students), which limits statistical precision. Second, the participants came from only two intact classes at one school, so class-level characteristics and school-specific conditions may have influenced the results. Third, the quasi-experimental design did not randomly assign individual students to programming environments. Fourth, the intervention consisted of one instructional session, which limits conclusions about longer-term learning and retention. Finally, the supplied normality results were mixed and did not specifically report the N-Gain distributions, so the inferential comparison should be interpreted cautiously. These limitations constrain the generalizability of the findings beyond comparable novice learning contexts.

VI. CONCLUSION

Within the context of the two participating classes at SMK N 4 Kupang, both programming approaches were associated with improved computational-thinking scores. The Scratch group achieved an average N-Gain of 0.78 (High), while the Python group achieved 0.49 (Medium). The N-Gain comparison was statistically significant, $t(47) = 8.12$, $p < .001$, with Cohen's $d = 2.37$. These results describe the outcomes observed in the participating intact classes and should not be generalized beyond comparable novice learning contexts without further evidence.

For educators working with comparable novice secondary students, visual programming may be considered as an introductory scaffold before learners move to text-based programming, particularly when the immediate goal is to establish algorithmic planning and computational-thinking foundations.

For future research, hybrid or transition models that explicitly connect block-based representations with equivalent text-based code could be examined across multiple schools and larger samples. Future studies should also report intervention fidelity and directly measure possible mediating factors such as cognitive load, debugging strategies, and prior programming experience.

REFERENCES

- Budiyanto, C. W., Fenyvesi, K., Maharani, Y. I., Yuana, R. A., Nashiroh, P. K., & Latifah, R. (2025). Investigating computational thinking in K-12 visual programming activities on Code.org: A Brennan-Resnick framework approach. *Elinvo (Electronics, Informatics, and Vocational Education)*, 10(1), 54–62. <https://doi.org/10.21831/elinvo.v10i1.78486>
- Findayani, N. K., Suparta, I. N., & Sariyasa. (2023). Development of project based statistics e-module with Scratch assistance to improve students' computational thinking skills. *Jurnal Pendidikan MIPA*, 24(3), 516–527. <https://doi.org/10.23960/jpmipa/v24i3.pp516-527>
- Hake, R. R. (1998). Interactive-engagement versus traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses. *American Journal of Physics*, 66(1), 64–74. <https://doi.org/10.1119/1.18809>
- Maharani, S., Susanti, V. D., Andari, T., Krisdiana, I., & Astuti, I. P. (2023). Computational Thinking (CT)-based Student Worksheet to Improve the Mathematical Literacy of Mathematics Prospective Teacher. *AL-ISHLAH: Jurnal Pendidikan*, 15(3), 2690–2702. <https://doi.org/10.35445/alishlah.v15i3.4412>
- Mustahib, M., Roshayanti, F., & Dewi, E. R. S. (2023). Profil computational thinking siswa kelas X SMA Negeri 2 Mranggen tahun 2023. *JP3 (Jurnal Pendidikan dan Profesi Pendidik)*, 9(1), 18–25. <https://doi.org/10.26877/jp3.v9i1.17044>
- Salim, F., Ika, K., Tira, T., Karnaen, A. F., Sari, D. R., & Rahmadita, Y. (2024). Analisis kemampuan berpikir komputasional siswa pada mata pelajaran Informatika di SMP Kota Pontianak. *Jurnal Pendidikan Informatika dan Sains*, 13(2), 257–263. <https://doi.org/10.31571/saintek.v13i2.10610>
- Sonjaya, R. P., & Munir, M. (2025). Examining the impact of block-based visual programming in programming education: A systematic review. *Indonesian Journal of Teaching in Science*, 5(1). <https://doi.org/10.17509/ijotis.v5i1.78732>
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33–35. <https://doi.org/10.1145/1118178.1118215>