

DESIGNING A SCRATCH-TO-PYTHON LEARNING PROGRESSION FOR PRIMARY SCHOOL LEARNERS: A DESIGN AND DEVELOPMENT RESEARCH APPROACH

Purushothman Munusamy¹
Mageswaran Sanmugam²

¹ Centre for Instructional Technology and Multimedia, Universiti Sains Malaysia, Gelugor 11800, Penang, Malaysia. (Email: zpuru2306@gmail.com)

² Centre for Instructional Technology and Multimedia, Universiti Sains Malaysia, Gelugor 11800, Penang, Malaysia. (Email: mages@usm.my)

Article history

Received date : 15-8-2026
Revised date : 16-8-2026
Accepted date : 15-10-2026
Published date : 20-10-2026

To cite this document:

Munusamy, P., & Sanmugam, M. (2026). Designing a Scratch-to-Python learning progression for primary school learners: A design and development research approach. *Journal of Islamic, Social, Economics and Development (JISED)*, 11 (87), 152 – 159.

Abstract: *This work considers the block-to-text programming transition, stressing representational and syntactic challenges that require careful instructional design, particularly for primary school students. The design and development of a carefully crafted Scratch-to-Python learning progression for Malaysian Standard 4 students is presented. The research follows a product-oriented approach to design and development. The design of the instructional structure takes into account Cognitive Load Theory, learning transfer, and computational thinking and states four design principles, namely representation alignment, progressive scaffolding, focus on code analysis, and cross-representation mapping. Two instructional paths are proposed: a Sequential model with eight Scratch lessons and eight Python lessons and a Concurrent model with eight Scratch and eight Python lessons altogether. Both paths have two lessons a week for 8 weeks. The paper describes design reasoning, concept mapping, instructional sequencing, assessment matching, and formative improvements gained through pilot implementation. Particularly, the final design for the main study places Scratch and Python code-analysis pre-test prior to the instruction, an intermediate engagement assessment at the end of Week 6, another engagement assessment at the end of Week 8, and code-analysis post-test after instruction. This paper adds to the body of research a clear instructional design specification but not the results of its effectiveness testing. Comparative testing will be done in future research.*

Keywords: *block-based programming; text-based programming; Scratch; Python; design and development research; primary education; code analysis skills*

Introduction

In this paper, the use of programming in computational thinking education will be considered along with problems associated with shifting from a visually organized block code to text. The scratch program presents the structure of the program graphically, while the Python language requires the learner to take into account the text, punctuation, indentation, and execution of the program (Lin & Weintrop, 2021; Strong et al., 2025).

For elementary school children, a successful shift should ensure both conceptual continuity and an incremental increase in representation complexity at a reasonable pace. In case of the substitution of Python for Scratch without ensuring that the learners are aware of common elements between the two programs, the transition would not be successful. Contemporary literature proves the need to establish connections between block-based and text-based representations with scaffolding helping beginners to reason across modalities (Lin et al., 2025; Strong et al., 2025).

The paper will describe the design process of two complementary learning paths for the learners of Standard 4 in Malaysia. The value of this research is in providing design logic of the learning path which includes lessons' organization, mappings of constructs, assessments, and adjustments due to pilot studies. The paper does not provide results of comparing the relative effectiveness of the two approaches.

Problem Statement

The literature on programming education often stresses the importance of computational thinking and programming success, but instructional descriptions give considerably less information on how students are coached to transit between different representations. The process of moving from blocks to code requires more than syntax acquisition; students need an opportunity to see the equivalent operations, predict the operation of the program, and use analysis strategies in the new representation system. Transition support is identified by Strong et al. (2025) as a separate issue; Lin et al. (2025) show how a transition between representations could be facilitated.

The second design issue is the timing of evaluation. Tests including questions related either to Python language or to connections between Scratch and Python language cannot be interpreted in the same way if a certain group of participants knows only about Scratch. Thus, the current project sees instructional experience and alignment of assessment instruments as two sides of the same design problem. The idea is to develop an appropriate and age-appropriate sequence before testing any causal hypotheses.

Research Objectives and Questions

Three goals for the development process are outlined in the paper: (1) identification of the design requirements for the progression from Scratch to Python; (2) conversion of these design requirements into Sequential and Concurrent lesson pathways; and (3) description of the formative changes made to the lesson and assessment structure. Three research questions that follow from these are related to the design requirements, their implementation in lesson pathways, and procedural changes made.

Literature Review

Block-Based and Text-Based Programming

Block-based environments help novice learners since they make the components of programs and valid combinations thereof more apparent. Conversely, textual programming requires learners to find the same constructs in an entirely different system. Hence, the learning problem represents a representational shift as opposed to just a change in software (Lin & Weintrop, 2021). Studies on shifts in representation stress explicit mapping, guided practice, and well-chosen bridging activities (Strong et al., 2025).

Scaffolding and Learning Transfer

Scaffolding is a way of providing temporary help to students until they gain the requisite skills needed to act independently. Scaffolding is defined in this research as the provision of matched examples, guided tracing, side-by-side comparisons, and Python problems with gradually fading help. Transfer is defined conservatively as the intended ability to transfer coding analysis ideas to both Scratch and Python formats; it is still an open question whether the transfer actually occurs (Lin et al., 2025; Chen et al., 2026).

Code Analysis and Engagement

Code analysis includes the prediction of outcomes, tracking of the control flow, detection of bugs, and analysis of changes in variables. This kind of task allows students to prove their understanding of concepts without the need to have each task as a coding activity. Engagement is a separate construct, which includes affective, behavioral, and cognitive aspects of learning (Fredricks et al., 2004). Thus, the designed program distinguishes between code-analysis tests and student engagement survey.

Theoretical and Conceptual Foundations

This design is based on three different but complimentary approaches. Cognitive Load Theory serves as a guide in introducing syntax and using worked examples. Learning transfer suggests establishing explicit relations between identical elements of programming. Finally, computational thinking influences choice of sequence, conditionals, repetition, decomposition, tracing, and debugging as the main elements of the course content. As a result, principles of representation alignment, scaffolding, analysis of code, and cross-representation mapping become the design principles of this course. These principles operate as design propositions rather than results achieved during the pilot study.

Table 1: Theoretical Foundations and Operational Design Principles

Foundation	Design implication	Instructional example
Cognitive Load Theory	Manage new notation and syntax demands	Matched worked examples and short code segments
Learning transfer	Make shared concepts across representations explicit	Scratch block-to-Python statement mapping
Computational thinking	Focus on reasoning about code behaviour	Tracing, debugging, prediction and explanation

Methodology

Research Design

The DDR Type I approach that focuses on products was used in developing the instructional product. The process involved the following steps: requirement analysis, design specification, development of instructional materials, evaluation by experts, piloting and then revision. The development goal was to create a teaching progression involving lesson plans, presentation material, exercise activities, and related assessment schemes. This report covers the design specification and procedures involved in improving the model. It does not involve experiment to assess its effectiveness.

Learners and Implementation Context

The intervention involves Malaysian Standard 4 children, aged about ten years. The pilot lasted for four weeks and involved 50 children from one urban primary school in northern Malaysia, with 25 children placed in each condition. One teacher delivered the intervention in both conditions. The main study will involve eight weeks and 16 lessons. The pilot will only inform design issues; statistical results from the pilot cannot be used again.

Instructional Design Procedure

The Sequential Pathway entails that the Weeks 1-4 are devoted to eight Scratch lessons, and the Weeks 5-8 are devoted to eight Python lessons. In contrast, the Concurrent Pathway entwines both the Scratch and the Python lessons in the entirety of the 16 lessons. Both the pathways entail two lessons of thirty minutes each in a week. Even though both the pathways are associated with similar computational concepts, both the pathways vary in the timing of the learners' introduction to the textual representations.

Table 2: Definitive Main-Study Instructional Architecture

Period	Sequential pathway	Concurrent pathway
Weeks 1–4 (Lessons 1–8)	Scratch instruction	Integrated Scratch and Python
Weeks 5–8 (Lessons 9–16)	Python instruction	Integrated Scratch and Python
Dose	16 × 30-minute lessons	16 × 30-minute lessons

Design Review and Formative Refinement

Expert review of the instruments associated with the related research provided guidance for the appropriate use of language for each age group and construct validity; however, this is not meant to imply independent verification of the whole instructional system or its efficacy as a result of this work. The specific expert reviews of the instructional design and validation results for the platform are not included in this article due to lack of independently verified data set.

Design and Development of the Learning Progression

Cross-Representation Concept Mapping

Concept maps serve as the link between design principles on paper and teachable class activities. Equivalent constructs were found in both Scratch and Python, and then converted into short cases that ask the students to predict, trace, compare, or debug the behavior of the program. This is a conceptual map, not a one-to-one mapping, because some elements in Scratch and Python do not map to each other precisely, and we do not want to suggest that there is an exact equivalence.

Table 3: Illustrative Scratch–Python Concept Mapping

Concept	Scratch example	Python example	Target analysis skill
Sequence	Ordered connected blocks	Ordered statements	Trace execution order
Variables	Set/change variable blocks	Variable assignment	Track state changes
Selection	If / if-else blocks	if / else statements	Predict branch execution
Repetition	Repeat / forever blocks	for / while loops	Trace iterations
Input and output	Ask / say blocks	input() / print()	Relate interaction patterns
Debugging	Incorrect block arrangement	Logical or syntax error	Locate and explain an error

The mapping table represents a design specification. The accurate lessons must be checked against the teacher manual before submitting the journal. Abstractness through the use of blocks and Python functions must be implemented only when such topics have been covered in the 16 lessons in the curriculum.

Sequential and Concurrent Learning Pathways

In the Sequential path, Scratch is used first for creating representations that are visual in nature. The subsequent teaching of Python takes up the same concepts but now using representations that are textual in nature and including prompts that encourage the learners to compare the execution, structure, and output. On the other hand, in the concurrent pathway, the comparison is made between representations of the two languages within one instructional period. Both paths include structured prompts to encourage analysis.

Lesson-Level Teaching Routine

A 30-minute procedure for increasing consistency is suggested. It comprises introduction of and access to the previous concept, a sample solution, code tracing or debugging, a task for the learner, and an explanation/reflection. Each of the components might take different times depending on the complexity of the concept, but the overall lesson will take 30 minutes no matter what. This procedure provides a framework without assuming that all students move at the same pace.

Table 4: Illustrative 30-Minute Lesson Routine

Phase	Approximate time	Purpose
Concept introduction and recall	5 minutes	Connect to prior representation
Worked example and mapping	7 minutes	Make program behaviour visible
Guided code analysis	8 minutes	Trace, predict or debug
Independent/paired task	7 minutes	Apply concept
Explanation and closure	3 minutes	Articulate reasoning

Assessment Alignment and Pilot-Informed Refinement

Two sets of outcomes families are clearly outlined by this experiment. Two code analysis measures (Scratch CASA and Python CASA) are performed before and after the final experiment. The Student Engagement and Attitude Survey (SEAS) is performed twice after exposure to Scratch and Python for each set. The first SEAS test was conducted after week 2 when the Sequential group was exposed only to Scratch and the Concurrent group was exposed

to both. It limits the use of this test as an equivalent to pre-exposure. The schedule resolves this problem without transforming the pilot test into a normal pre-test.

Table 5: Definitive Assessment Schedule

Time	Assessment	Measurement interpretation
Before Week 1	Scratch CASA pre-test and Python CASA pre-test	Pre-intervention baseline
End of Week 6, after Lesson 12	SEAS T1	Intermediate/mid-late engagement measurement
End of Week 8, after Lesson 16	SEAS T2	Post-intervention engagement measurement
After Week 8	Scratch CASA post-test and Python CASA post-test	Post-intervention code-analysis performance

SEAS T1 is not a typical pretest since it is conducted after both groups have interacted with the programming representation contained in the survey. The CASA posttests will be conducted after the last instructional period has ended to ensure that evaluation does not interfere with the instructional process. This helps establish how the measurement aligns with the construct being tested and illustrates the formative nature of pilot studies (Thabane et al., 2010).

Discussion

The primary advantage of the developed progression is the clear definition of how Scratch and Python could be taught together to children. It makes representational alignment a design choice and offers two exposure sequences to be compared in future research. It satisfies calls for direct transition assistance as it cannot be assumed that skill in blocks implies skill in text (Strong et al., 2025; Lin et al., 2025).

The design also separates the instructional design process and evaluation of its results. The concept map, lesson sequence, and assessment schedule are development artifacts and do not provide a proof of increased coding analysis skills or engagement. The pilot has uncovered the timing of the assessment problem and helped to refine the procedure, yet a controlled evaluation is necessary to assess which instructional path is more effective and whether there is representation transfer.

Another consideration relates to the quality of measurement. While content review helps to ensure the relevancy and clarity of items, it cannot replace evidence of internal consistency and construct validity. Item-level psychometric analysis is necessary before the aggregate scores of the SEAS pilot could be considered validated measures in the main experiment. This is why the design paper does not use engagement-score changes as the proof of success of the developed model.

Educational Implications

The progression provides a concrete basis for educators to develop matching Scratch-Python pairs and code analysis tasks. It serves as a specification of criteria in terms of representational continuity and scaffolding for curriculum developers, rather than desirable features to incorporate in the design of a programming language intervention curriculum. For researchers, it highlights the importance of differentiating sequencing of instruction from timing of measurements.

Limitations and Future Research

This paper is a design and development report rather than an effectiveness study. It does not include independently verified findings of instructional platform validity or a completed primary study comparison. The lesson routine and concept map are design examples that should be independently validated against the resulting instructional platforms. It will be the job of future research to examine the issues of implementation fidelity, item functioning analysis, code analysis findings, and effectiveness of the Sequential and Concurrent paths.

Conclusion

Scratch to Python progression, which is described in the following, is a theoretically based and replicable instruction design for Malaysian elementary schools. Two possible routes of progression, the common focus on code interpretation, and the assessment plan corresponding to children's representational experiences were specified. The main value of the research consists in the design process of the instructional material; issues of transfer and comparative effectiveness still have to be empirically clarified.

Acknowledgements

The authors acknowledge the support and academic guidance of Universiti Sains Malaysia and the participating school community.

References

- Chen, D., Zhang, Y., Luo, H., Gao, Z., Yu, L., & Lin, Y. (2026). Exploring the impact of scaffolding in programming on students' computational thinking: Evidence from a three-level meta-analysis. *Journal of Educational Computing Research*, 64(1), 208–237. <https://doi.org/10.1177/07356331251386618>
- Giannakoulas, A., & Xinogalos, S. (2024). Studying the effects of educational games on cultivating computational thinking skills to primary school students: A systematic literature review. *Journal of Computers in Education*, 11, 1283–1325. <https://doi.org/10.1007/s40692-023-00300-z>
- Kunz, K., Moeller, K., Ninaus, M., Trautwein, U., & Tsarava, K. (2023). Making the transition to text-based programming: The pilot evaluation of a computational thinking intervention for primary school students. In *Proceedings of the 18th WiPSCE Conference on Primary and Secondary Computing Education Research*. Association for Computing Machinery. <https://doi.org/10.1145/3605468.3609770>
- Lin, Y., & Weintrop, D. (2021). The landscape of block-based programming: Characteristics of block-based environments and how they support the transition to text-based programming. *Journal of Computer Languages*, 67, 101075. <https://doi.org/10.1016/j.cola.2021.101075>
- Lin, Y., Weintrop, D., & McKenna, J. (2025). Switch mode: How one environment supports multiple strategies to transition from block-based to text-based programming. *International Journal of Child-Computer Interaction*, 44, 100737. <https://doi.org/10.1016/j.ijcci.2025.100737>
- Mills, K. A., Cope, J., Scholes, L., & Rowe, L. (2025). Coding and computational thinking across the curriculum: A review of educational outcomes. *Review of Educational Research*, 95(3), 581–618. <https://doi.org/10.3102/00346543241241327>
- Mladenović, M., Žanko, Ž., & Zaharija, G. (2024). From blocks to text: Bridging programming misconceptions. *Journal of Educational Computing Research*, 62(5), 1082–1106. <https://doi.org/10.1177/07356331241240047>

- Montuori, C., Gambarota, F., Altoè, G., & Arfé, B. (2024). The cognitive effects of computational thinking: A systematic review and meta-analytic study. *Computers & Education*, 210, 104961. <https://doi.org/10.1016/j.compedu.2023.104961>
- Strong, G., Bresnihan, N., & Tangney, B. (2025). Supporting learners in the transition from block-based to text-based programming: A systematic review. *Journal of Computer Languages*, 84, 101342. <https://doi.org/10.1016/j.cola.2025.101342>
- Thabane, L., Ma, J., Chu, R., Cheng, J., Ismaila, A., Rios, L. P., Robson, R., Thabane, M., Giangregorio, L., & Goldsmith, C. H. (2010). A tutorial on pilot studies: The what, why and how. *BMC Medical Research Methodology*, 10, Article 1. <https://doi.org/10.1186/1471-2288-10-1>
- Tsai, M.-J., Chien, F. P., Sun, W.-T., & Jha, N. K. (2025). How block-based programming supports novice learners' coding comprehension: Evidence from eye-tracking lag-sequential analysis. *Computers & Education*, 239, 105430. <https://doi.org/10.1016/j.compedu.2025.105430>
- Wei, Y., Wang, L., Tang, Y., Su, J., Lei, Y., & Peng, W. (2024). Influence of programming education modalities on the computational thinking in young children: A comprehensive review and meta-analysis. *Journal of Computer Assisted Learning*, 40(5), 2385–2397. <https://doi.org/10.1111/jcal.13038>
- Wong, G. K. W., Jian, S., & Cheung, H.-Y. (2024). Engaging children in developing algorithmic thinking and debugging skills in primary schools: A mixed-methods multiple case study. *Education and Information Technologies*, 29, 16205–16254. <https://doi.org/10.1007/s10639-024-12448-x>
- Yu, Q., Yu, K., & Li, B. (2025). Effects of block-based visual programming on K–12 students' learning outcomes. *Journal of Educational Computing Research*, 63(1), 64–98. <https://doi.org/10.1177/07356331241293163>
- Fredricks, J. A., Blumenfeld, P. C., & Paris, A. H. (2004). School engagement: Potential of the concept, state of the evidence. *Review of Educational Research*, 74(1), 59–109. <https://doi.org/10.3102/00346543074001059>