JOURNAL OF RESEARCH, INNOVATION, AND STRATEGIES FOR EDUCATION
VOL. 2, NO. 6, PP. 16-30, DECEMBER 2025
https://doi.org/10.70148/rise.v2i6.2

Developing Computational Thinking Skills Through a Text-Based Programming Approach: A Research Framework

Fariza Zahari [1,2]*, Ahmad Zamzuri Mohamad Ali [1]

[1] Faculty of Art, Sustainability & Creative Industry, Universiti Pendidikan Sultan Idris, Perak, Malaysia
[2] Electrical Engineering Department, Politeknik Sultan Salahuddin Abdul Aziz Shah, Selangor, Malaysia

ABSTRACT
Computational thinking has been identified as a fundamental requirement within 21st-century education curricula. It encompasses systematic problem analysis and the development of algorithmic solutions to address problems efficiently. Prior studies have predominantly focused on unplugged activities and Programming Assistance Tools (PATs), such as Alice and Scratch, in the development of computational thinking skills. However, empirical investigations examining the effectiveness of computational thinking within text-based programming environments remain limited. This study aims to develop a conceptual framework that integrates both theoretical and practical dimensions in the teaching of computational thinking, while also evaluating the effectiveness of two text-based programming languages: Python and C++ in enhancing these skills. The construction of this framework is grounded in two key models: Model of Learning Computational Thinking, which emphasizes cognitive development, and Programming in Seven Steps, which focuses on structured programming strategies. As a result, an integrative conceptual framework is proposed to support a more structured and reflective teaching and learning process. This framework is expected to inform future research and curriculum development initiatives related to the cultivation of computational thinking skills, particularly within text-based programming contexts.

ARTICLE HISTORY
Received 20 August 2025
Revised 15 October 2025
Accepted 21 October 2025

KEYWORDS
Computational Thinking; Unplugged Activities; Programming Assistance Tools; Text-Based Programming; Structured Programming

1. Introduction

The rapid development of current technology has further increased the need for programming skills in many job sectors such as cyber technology, software engineering, the Internet of Things (IoT) industry, banking, administration and so on (Eberhard et al., 2017; Ogegbo et al., 2024). Programming is now a fundamental part of the curriculum of

CONTACT Fariza Zahari fariza@psa.edu.my Faculty of Art, Sustainability & Creative Industry, Universiti Pendidikan Sultan Idris, Perak, Malaysia. ISSN : 3030-6582 (Printed), eISSN : 3030-5330 (Online). DOI 10. 10.70148/rise.v2i6.2.

many relevant programs at higher education institutions in order to satisfy this demand. The interest of students in pursuing a specialisation in programming, however, remains restricted (Abdunabi et al., 2019). A considerable number of students regard programming as a tough course that is challenging for them to grasp (Figueiredo & García-Peñalvo, 2018; Ibrahim, Nurul Farhana, & Zaidatun, 2018; Siti Rosminah & Ahmad Zamzuri, 2012).

Students have challenges in mastering programming courses due to their difficulties in identifying and comprehending essential concepts, including variable types, expressions, Boolean logic, loops, and abstraction (Grover & Jackiw, 2019; Grover, Qian & Lehman, 2017; Pea, & Cooper, 2016), a weak foundational understanding of syntax elements such as semicolons, parentheses/brackets, and quotation marks (Qian & Lehman, 2017), lack of proficiency in English (Mhashi & Alakeel, 2013), limited mathematical skills (Psycharis & Kallia, 2017), and other related weaknesses. Nonetheless, other prior research has highlighted programming problem-solving skills as the principal factor leading to students' inability to succeed in this field (Figueiredo & García-Peñalvo, 2018; Mhashi & Alakeel, 2013; Tuomi et al., 2018). Consequently, an ability to resolve programming-related problems is an essential basic talent that must be developed prior to students engaging in advanced and practical programming (Flórez et al., 2017).

Problem-solving abilities in programming include a systematic approach that involves problem analysis, flowchart construction, algorithm development, code writing, program testing, and error debugging (Ibrahim et al., 2018). Problem-solving skills in programming are similar to computational and logical thinking. This is because the mind tries to turn real problems into a set of step-by-step directions that a computer can follow (So et al., 2020; Ahmad Zamzuri, 2019). Computational thinking does not entail thinking like a computer (Wing, 2008); rather, it involves understanding how computers process and execute problem-solving tasks through logical and structured procedures (Garcia-Penalvo & Mendes, 2018; Yadav, Mayfield, Zhou, & Hambrusch, 2014; Hemmendinger, 2010). Computational thinking fundamentally consists of five essential approaches for problem-solving: decomposition, pattern recognition, abstraction, algorithmic thinking and generalisation (Ahmad Zamzuri, 2019; Chee, Meng, Wu, Seow, & Liu, 2018).

Computational and logical thinking skills are seen as essential competencies that should be imparted to students from the school level onwards (Yadav et al., 2014). As a result, numerous countries have initiated the introduction of computational thinking into the educational curriculum at the school level (Garvin, Killen, Plane, & Weintrop, 2019; Tsai, Wang, & Hsu, 2019; Flórez et al., 2017) and Malaysia is no exception to this trend. Malaysia has officially incorporated computational thinking into its national curriculum via initiatives such as MyDigitalMaker programme, initiated in 2016, alongside comprehensive teacher training and professional learning communities organised by MDEC in partnership with the Ministry of Education (MDEC, 2016;2022). Empirical studies conducted in Malaysia support the beneficial effects of integrating computational thinking in primary and secondary education (Azmi et al., 2024; Felicia et al., 2017; Ung et al., 2021).

A variety of computational thinking skill-building activities have been introduced at the school level, including unplugged activities (Bell & Vahrenhold, 2018; Rodriguez, Rader, & Camp, 2016; Thies & Vahrenhold, 2013) and Programming Assistance Tool (PAT) (Yuana & Maryono, 2016 ; Koorsse & Cilliers, 2015). Unplugged activities are interactive, kinaesthetic learning exercises that let students investigate computing ideas without actually coding or using computers (Bell & Vahrenhold, 2018; Rodriguez et al., 2016). Numerous previous studies have found that unplugged activities have the potential to support school students in visualizing computing concepts through kinesthetic-based tasks such as binary number modules, binary search, network sorting, and various others (Bell & Vahrenhold, 2018; Chee et al., 2018; Curzon, McOwan, Plant, & Meagher, 2014; Feaster, Segars, Wahba, & Hallstrom, 2011a; Rodriguez, Kennicutt, Rader, & Camp, 2017; Thies & Vahrenhold, 2012; Thies & Vahrenhold, 2013). Also, unplugged activities have been shown to help students comprehend programming ideas better (Curzon et al., 2014). Similarly, the Programming Assistance Tool (PAT) is an application software designed to expose students to programming concepts and strengthen their understanding of them (Yuana & Maryono, 2016; Koorsse & Cilliers, 2015). The Programming Assistance Tool (PAT) has a simplified Integrated Development Environment (IDE) with a drag-and-drop tool to make it easier to use, especially for individuals who just started learning out (Shepherd et al., 2018). Students can learn programming more easily with PAT since they don't have to worry about creating code or making syntax mistakes (García-Peñalvo & Mendes, 2018; Wen et al., 2023). There are various PAT software tools

available as open-source resources; however, most previous studies recommend the use of Scratch (Dohn, 2019; Marcelino, Pessoa, Vieira, Salvador; Nada Sharaf, Ahmed Ghada, Ahmed Adel, Abdennadher, & Berkling, 2019 & Mendes, 2018) and Alice (Csapo Gabor, 2019; Sivaraman et al., 2019) as effective platforms for teaching and learning programming concepts at the school level. Despite PAT's success in increasing students' interest and comprehension of programming principles, research indicates that students still find it difficult to relate these ideas to creating actual code (Henry & Dumas, 2018). This is because actual code writing is text-based and resembles computer language, whereas PAT utilizes an object-based approach, where students use drag-and-drop command blocks within a command window.

While unplugged activities and Programming Assistance Tools (PAT) effectively enhance conceptual comprehension, their efficacy in facilitating the transfer of these concepts to practical programming abilities is ambiguous. Thus, these methodologies are deemed more suitable for cultivating computational thinking skills in students at the elementary and lower secondary levels, specifically those in Forms 1, 2, and 3 (Zhao et al., 2022; Juricic & Radosevic, 2019; Bell & Vahrenhold, 2018). For upper secondary levels, such as Forms 4, 5, and 6, as well as pre-university programmes, the focus should be directed more toward the use of actual programming languages, in order to better prepare students for programming courses at the tertiary level (Zhao et al., 2022; Gonzalez et al., 2018; Yuen et al., 2018).

In light of these uncertainties and issues, the utilisation of actual programming languages as a method for developing computational thinking abilities offers a viable alternative. This method is especially important for students in pre-university programs, including matriculation, foundation in science, and diploma courses, as well as those in the upper secondary level. This urges the question of whether programming language is best suited for the purpose. Previous research shows the choice of programming language may significantly impact student learning outcomes, particularly among beginning learners. Although the main objective is to develop computational thinking skills, choosing a suitable programming language becomes important when programming tasks are involved. The concern comes from the fact that some computer languages can make learning them hard and stressful, which could make it harder for students to reach their goal of improving their computational thinking skills.

This study aims to analyse the effect of programming language selection on the development of computational thinking skills in novice students. This comparison focusses on Python and C++, both of which are high-level programming languages commonly utilized for teaching structured programming to novices. To achieve this aim, three particular goals have been established:

i. to develop two sets of laboratory worksheets designed to foster computational thinking skills using two text-based programming languages, Python and C++
ii. to develop an assessment instrument for measuring computational thinking skills in the context of text-based programming
iii. to analyses the effects of using Python and C++ on the development of computational thinking skills among novice learners.

2. Literature review

2.1. Computational thinking skills

Computational thinking skills are mental abilities that involve combining fundamental concepts and reasoning from the field of computer science to solve problems in a variety of disciplines, not only programming or artificial intelligence. Computational thinking skills are the process of formulating problems and solving them so that the solution can be represented in a form that can be effectively carried out by computers or humans themselves (Wing, 2006). Computational thinking is a cognitive process that encompasses the development and application of diverse knowledge and skills, including problem formulation, definition, analysis, abstraction, and logic, to devise solutions that can be efficiently executed by information processing agents (Brennan & Resnick, 2012).

Computational thinking skills are the most important element in 21st-century skills (Khasyyatillah et al., 2024; Salleh Hudin, 2023; Voogt et al., 2013). With the development of the Fourth Industrial Revolution and advancements in digital technology, the necessity to teach computational thinking abilities in the students from as early as school level and university preparation has become even more crucial. (Khasyyatillah et al., 2024; Lockwood & Mooney, 2017, 2018). This can assist in equipping students with computational thinking abilities, enabling them to maintain competitiveness and relevance in a world increasingly governed by intelligent technology and automation.

Numerous studies have proposed models and theoretical frameworks for understanding computational thinking. Among them, Brennan and Resnick (2012) developed a framework to examine and assess the development of computational thinking among young designers, particularly using the visual programming environment, Scratch. In their study, they highlighted three core dimensions of computational thinking: computational concepts, computational practices, and computational perspectives. Seven computational ideas were identified and implemented in Scratch-based activities, including sequence, loops, events, parallelism, conditionals, operators, and data. To assess students' computational behaviours and perspectives, the researchers used interviews and observations. In addition, three methods were utilised to evaluate the effectiveness of the suggested framework: analyse student projects, interviews, and design situations. However, the study's findings indicate that the framework is deficient in standardised assessment instruments and well-defined evaluation metrics, emphasising the need for further research to increase its validity and usefulness in many educational contexts.

Kalelioglu et al. (2016) did a systematic review of available literature to try to come up with a complete framework for computational thinking. A review of 125 papers from six databases and digital libraries led to creating the theoretical framework. The selected articles were reviewed by three researchers, who examined several criteria including the study objectives, target population, theoretical foundation, elements of computational thinking, research methods, and scope of the studies. This study used a qualitative approach, with data analysed using statistical methods. The studies highlighted four major components required for the application and assessment of computational thinking: the cognitive component, which includes logical thinking, problem-solving abilities, algorithmic thinking, modelling, and simulation. The technical component involves knowledge and skills in the use of digital tools and programming languages. The pedagogical component refers to instructional approaches and the design of learning environments that foster the development of computational thinking. Finally, the social and affective component promotes motivation, social engagement, and attitudes towards technology. This paper gives an entire theoretical framework with numerous aspects, which serves as an essential foundation for understanding and developing computational thinking skills in current education. Furthermore, it provides a framework for continuing development and expansion of research in accordance with current technological and pedagogical advances (Kılıç et al., 2020; Kong & Abelson, 2019; Palts & Pedate, 2020).

Subsequently, the study by Palts and Pedaste (2020) introduced A Model for Developing Computational Thinking Skills. The model was developed based on a systematic review of 65 scholarly articles, retrieved through academic search platforms including EBSCO Discovery Service and the ACM Digital Library. The researchers categorized the selected articles into three main areas: frameworks, definitions, and dimensions of computational thinking. Based on the findings from this systematic literature review, six clusters of computational thinking skill dimensions were identified from six key articles, as presented in Table 1.

Table 1. Six clusters of computational thinking skill dimensions

		Wing, 2006	Barr et al., 2011	CSTA & ISTE, 2011	Brennan & Resnick, 2012	Moreno-León et al., 2015
Defining the problem	Problem formulation Problem formulation	Problem formulation Problem formulation	-	Formulating problems	-	-
	Abstraction	Abstraction	Abstraction	Abstraction	Abstraction and modularizing	Abstraction
	Problem reformulation	Problem reformulation	-	-	-	-
	Decomposition	Problem decomposition	Problem decomposition			Problem decomposition
Solving the problem	Data collection and analysis	-	Data collection, data analysis, data representation, simulation	Organizing and analyzing data	Reusing and remixing	User interactivity, data representation
	Algorithmic design, parallelization and iteration, automation	Automation	Automation, algorithms and procedures, parallelization	Algorithmic thinking, automating solutions	Being incremental and iterati	Parallelism, logical thinking, synchronization, flow control
Analyzing the solution	Generalization	-	-	Generalization	-	-
	Testing and evaluation	Systematic testing	-	Identifying, analyzing, and implement solution	Testing and debugging	-

The findings from this literature review demonstrate that the definition of computational thinking skills across all six articles is consistent, emphasizing a systematic, step-by-step process for solving problems.

The fundamental principles of computational thinking begin with problem identification and conclude with testing and evaluation. This problem-solving approach is iterative and cyclical, progressing methodically and continuously until a practical and accurate solution is achieved. These conclusions have led to the formulation of the Model for Developing Computational Thinking Skills, as shown in Figure 1, which provides the conceptual foundation for defining computational thinking in this study.

Figure 1. Conceptual foundation for defining computational thinking

2.2. Text programming

Text-based programming refers to a programming approach that involves writing code to perform specific tasks. Each program consists of a set of command terms known as keywords and lines of instructions for specific processes, referred to as syntax. Most programming languages use English as the primary medium, making it easy for users to understand; however, this language is not directly interpretable by computers. Therefore, a compiler or interpreter is required to translate the written code into machine language so it can be executed by the computer.
Many text programming languages come from C language, like C++, C#, Java, Python, and more (Umezawa et al., 2023). These languages have common keywords and syntax like C, so basic programming ideas can apply to all of them. Text programming is widely used in many fields, including computing, software development, cybersecurity, manufacturing, and others.

Bringing text-based programming to high school students, like those in grades 10, 11, 12, and pre-university, is seen as a good way to give students key skills for their future studies and jobs (Gonzalez et al., 2018; Yuen, Reyes, & Zhang, 2018). This method helps students build problem-solving abilities and take advantage of the strengths of text-based programming. Languages such as Python, Java, and C++ allow for the creation of complex applications in areas like web development, data science, and artificial intelligence, among others.

An important question comes up when deciding which programming language is most effective in assisting students to develop concepts in computers. Which side succeeds better: C or Python? It is well known that C is quite efficient and allows for a low-level management of system resource (Arboleda et al., 2023; Fatima, 2023; Lion et al., 2022). That is why teaching fundamental possibilities of programming, for example, data structures and memory allocations, might include a C language (Arboleda et al., 2023; Fatima, 2023; Munawar & Naveed, 2022). Unfortunately, C is quite complex and might be a problem for newcomers, which can pose a barrier to their early days of learning (Munawar & Naveed, 2022). Contrary to that, Python is acquired rather differently since the syntax is simpler and easier to learn which indeed makes the experience better (Munawar & Naveed, 2022; Salayev, 2024). This implies that learners can grasp programming's logical and structural concepts instead of getting lost in complicated visually appealing codes. In this situation, it begs the question: Is it better to understand a program in greater detail, such as in the case of learning C and being able to create new concepts or models, or to understand the concept that is sufficiently versatile and therefore easy to learn, such as Python?

3. Discussion

3.1 Theoretical Framework

3.1.1 Model of Learning Computational Thinking

The theoretical framework guiding this study is derived from the Model of Learning Computational Thinking (Palts & Pedaste, 2020) and the Programming in Seven Steps (PSS) Model (Erümit et al., 2019). The Model for Developing Computational Thinking Skills, depicted in Figure 1, identifies three core stages of computational thinking: problem identification, problem-solving, and solution analysis.

The problem identification stage focuses on four important skills: The first one is problem formulation, abstraction, problem reformulation, and decomposition. The process begins with problem formulation where the first objective is to establish what the problem is. Then comes abstraction where the focus is made on the key aspects of the problem and other information is ignored. Then, problem reformulation involves restating the problem in terms of prior challenges that have been faced or solved. Last, decomposition entails identifying the elements of a problem, which include its inputs, outputs, and processes and breaking the problem down into its individual parts.

The second phase of this model is the problem-solving phase and it consists of the following four problem-solving skills: The data collection and analysis, algorithm design, parallelization and iteration, and automation. The third

phase is solution analysis which consists of generalization which is the process of solving the problem of the solution i.e. transferring this problem solving process to a broader approach and testing and evaluation which is evaluating and ensuring that the output meets and is accurate to the requirements of the problem. This process is iterative and is carried out to meet the needs and expectations of the user until the output is satisfactory.

3.1.2 Programming in Seven Steps (PSS) Model

The second theoretical framework referenced is the *Programming in Seven Steps (PSS) Model* (Erümit et al., 2019), designed to provide a structured guide for teachers introducing programming in secondary schools. This model was developed through a collaborative process involving a design team and a design evaluation team, ensuring its robustness and practical applicability in educational settings. The study culminated in the identification of seven systematic steps for teaching programming, aimed at simplifying and organizing the learning process for both teachers and students. The seven steps include understanding the problem, designing the algorithm, coding the algorithm, testing, identifying and correcting errors (debugging), running the program, and evaluating the solution.

3.1.3 Relationship between Model of Learning Computational Thinking and Programming in Seven Steps

The Programming in Seven Steps (PSS) model can be integrated into the Model of Learning Computational Thinking, as both emphasize a systematic approach to problem-solving. The PSS model's processes, such as solution design, algorithm building, coding, and debugging, directly assist the development of essential concepts and practices in computational thinking, including abstraction, algorithmic reasoning, and error detection and correction (Brennan & Resnick, 2012). This combination has the potential to enhance students' capacity to apply computational thinking effectively through structured programming activities (Grover & Pea, 2013). Table 2 presents the integration between the Programming in Seven Steps (PSS) model and the Model of Learning Computational Thinking.

Table 2 Integration between the Programming in Seven Steps (PSS) model and the Model of Learning Computational Thinking

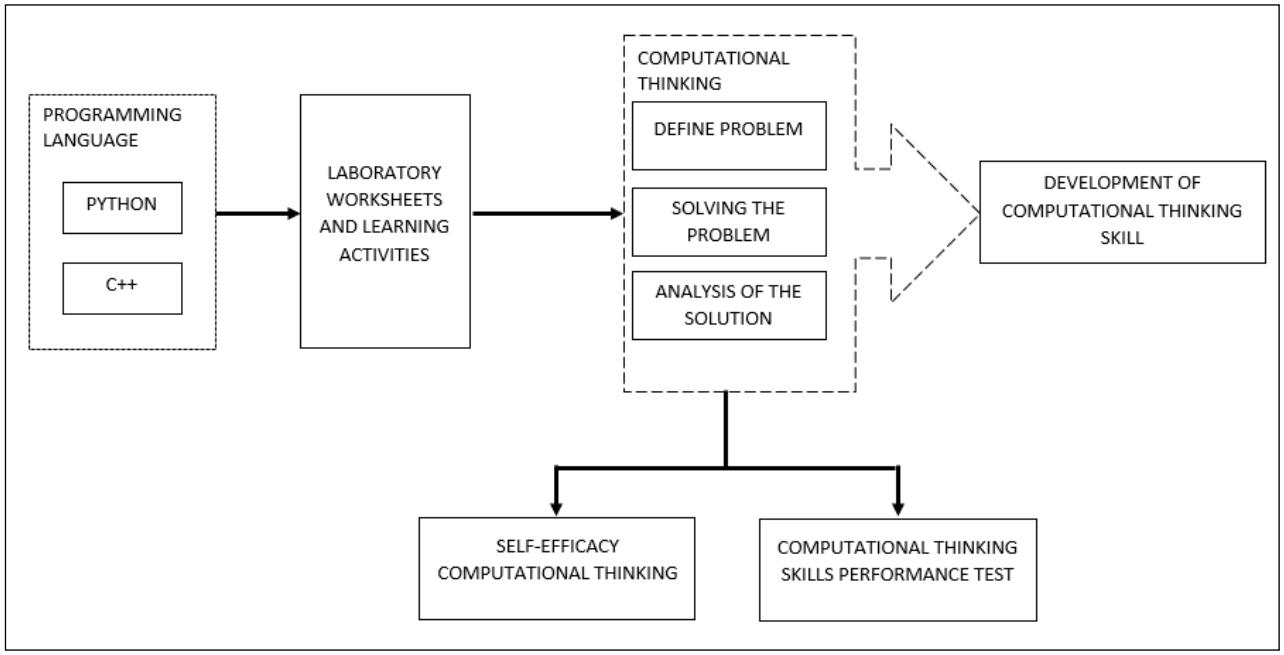
Model of Learning Computational Thinking	***Model Programming in Seven Step***	**Integration Explanation**
Define problem	Step 1: Understand the problem	The learning process begins with students identifying and analyzing the problem, employing decomposition and abstraction techniques to break it down into manageable components
Solving the problem	Step 2: Design Algoritm	Planning the solution steps in a structured and systematic manner
	Step 3: Coding	Write code based on the designed algorithm.
	Step 4: Run	Test the program results to ensure they meet requirements.
	Step 5: Debugging	Correcting errors identified within the program
	Step 6: Execute the program	Executing the program after the debugging process has been completed
Analysis of the solution	Step 7: Evaluate the solution	Students evaluate the final outcome, assess its effectiveness, and engage in generalisation

Overall, the PSS model can be considered as a complementary framework that provides a practical, action-oriented approach to implementing the components indicated in the Model of Learning Computational Thinking. The combination of these two methods has the potential to improve students' grasp of programming topics and boost their computational thinking skills by providing a structured, reflective, and application-based learning environment.

3.2 Conceptual framework

The research framework of this study, as illustrated in Figure 2, demonstrates the relationship between the study's variables and the computational thinking skills model. Within this framework, two dependent variables—the Python and C++ programming languages—are integral to the development of laboratory worksheets and learning activities designed to foster computational thinking. The laboratory worksheets and learning activities will be structured based on the Model of Learning Computational Thinking (Palts & Pedaste, 2020) and the Programming in Seven Steps (PSS) Model (Erümit et al., 2019). The Model of Learning Computational Thinking provides a comprehensive structure for identifying and developing core computational thinking skills, while the PSS Model offers a detailed, step-by-step methodology for programming instruction. By integrating these two models, the study seeks to create effective educational resources that facilitate the acquisition of computational thinking skills in a systematic and accessible manner, particularly through programming activities involving Python and C++. These resources aim to bridge theoretical concepts with practical applications, ultimately enhancing students' problem-solving abilities and programming proficiency.

Figure 2. Relationship between the study's variables and the computational thinking skills model



The selection of Python and C++ as the programming languages for the laboratory worksheets is aimed at providing students with a balanced experience in understanding and applying computational thinking skills. Python, with its simple and user-friendly syntax, allows students to focus on fundamental aspects such as problem formulation and algorithm design without being overwhelmed by complex technical requirements. This is particularly beneficial for beginners who are just being introduced to programming, as it enables them to grasp foundational concepts more quickly and effectively (Muhammad Ateeq et al., 2014; Nguyen, 2019).

The laboratory worksheets will incorporate five components of computational thinking skills: decomposition, pattern recognition, abstraction, algorithm design, and program writing and evaluation. These computational thinking skills are categorized into three main sections within the study's conceptual framework: conceptual knowledge, algorithmic thinking, and evaluation. The selection of an appropriate programming language is crucial in developing computational thinking skills to ensure that students do not experience undue anxiety during the learning process. By using carefully designed laboratory worksheets, the study aims to create an environment where students can engage with computational thinking concepts effectively and confidently, fostering both their understanding and practical application of programming.

The laboratory worksheets and learning activities will be assessed through two methods: performance tests and self-efficacy questionnaires. These methods are designed to provide a comprehensive evaluation of the effectiveness of the materials in fostering students' computational thinking skills, particularly in the context of text-based programming. This dual evaluation approach enables an in-depth understanding of both the cognitive outcomes and the students' self-confidence in applying computational thinking concepts to programming tasks.

3.3 Implication

3.3.1. Implication for students

This study provides significant implications for the development of students' skills and attitudes in the process of learning text-based programming. Through programming Python and C++ language, students are able to understand the differences in syntax structures and the logical thinking required in each language. This significantly benefits students' computational thinking skills, especially in the domains of decomposition, pattern identification, abstraction, and algorithmic thinking, which serve as the foundation for systematic problem-solving.

The activities scaffolded in a progressive manner provide students with opportunities to think critically and reflectively about the problem-solving steps they employ. Through this approach, students not only acquire technical programming skills but also develop an analytical and systematic way of thinking, which can be applied across various fields of learning.

Furthermore, this laboratory-based approach promotes collaborative learning, in which students can share ideas and problem-solving solutions with their colleagues. This setting helps students improve their motivation, curiosity, and technical communication abilities in the context of computer programming. Overall, this study helps to develop students' 21st-century skills by integrating computational thinking skills into text-based programming education.

3.3.2. Implication for teachers

This study provides practical suggestions for developing effective teaching practices based on computational thinking. The developed laboratory worksheets can be used as a teaching model to assist teachers plan structured laboratory activities that focus the development of higher-order thinking skills. This approach allows lecturers to not only assess student achievement in terms of programming results, but also in terms of the thinking processes used in solving problems.

Lecturers can evaluate students' abilities consistently and systematically with the use of the computational thinking skills evaluation tool developed in this study. The tool may be used as a basis to improve current methods of assessment to better meet the requirements of the computer science and information technology syllabuses, with an important priority on the development of critical, logical, and creative thinking abilities.

Additionally, the results of this study assist in the professional development of teachers, particularly when it comes to implementing instructional strategies based on computational thinking. Teachers can enhance their instruction, adapt lab activities to students' skill levels, and encourage active, student-cantered learning with the use of explicit, evidence-based guidelines and instruments.

Overall, the implications of this study show that a computational thinking–based teaching approach using Python and C++ can strengthen students' ability to solve problems systematically and creatively. At the same time, it provides valuable guidance for teachers to improve teaching design, assessment methods, and learning strategies in text-based programming. These implications are expected to contribute to enhancing the quality of teaching and learning in computer science education at higher education institutions.

4. Conclusion

Computational thinking is a critical cognitive skill that plays a significant role in enhancing the effectiveness of programming education. It provides students with a structured framework for problem-solving, emphasizing logical and sequential reasoning that is applicable to both programming tasks and real-life challenges. The mastery of programming skills necessitates engagement in the computational thinking process as a prerequisite to writing code. Additionally, the selection of an appropriate programming language is pivotal, as it can substantially influence the development of students' computational thinking abilities. Accordingly, this study seeks to propose a conceptual framework for cultivating computational thinking skills through text-based programming among TVET students, aimed at preparing them for higher education in university settings.

Declarations

Acknowledgements
None.

Competing Interests
None.

Ethical Approval
None.

Author's Contribution
Author[1]: Writing – original draft
Author[2]: Supervision

Data availability
None.

References

Abdunabi, R., Hbaci, I., & Yu Ku, H. (2019). Towards Enhancing Programming Self-Efficacy Perceptions Among Undergraduate Information. *Journal of Information Technology Education: Research*, *18*, 185–206.

Ahmad Zamzuri, M.A. (2019). *Coding di sekolah: Panduan untuk guru dan pelajar*. UPSI Press.

Arboleda, F. J. M., Arias, M. R., & Riveros, J. A. H. (2023). Performance of Parallelism in Python and C++. *IAENG*

International Journal of Computer Science, *50*(2).

Azmi, I. N., Atan, N. A., Tahir, L., Paslan, N. A., & Inda, A. (2024). *View of Integration of Computational Thinking in Learning Computer Programming with Gamification Elements to Foster Student Programming Skill and Student Performance*. Innovative Teaching and Learning Journal. https://itlj.utm.my/index.php/itlj/article/view/156/139

Barr, D., Harrison, J., & Conery, L. (2011). Computational Thinking: A Digital Age Skill for Everyone. *Learning & Leading with Technology*, *v38*, 20–23. https://eric.ed.gov/?id=EJ918910

Bell, T., & Vahrenhold, J. (2018). *CS Unplugged — How Is It Used , and Does It Work ?* (H.-J. Böckenhauer, D. Komm, & W. Unger (eds.); Vol. 1). Springer International Publishing. https://doi.org/10.1007/978-3-319-98355-4

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *Proceedings of the 2012 Annual Meeting of the American Educational Research Association, Vancouver, Canada (AERA 2012)*, 1–25. https://doi.org/10.1.1.296.6602

Buitrago Flórez, F., Casallas, R., Hernández, M., Reyes, A., Restrepo, S., & Danies, G. (2017). Changing a generation's way of thinking: Teaching computational thinking through programming. *Review of Educational Research*, *87*(4), 834–860. https://doi.org/10.3102/0034654317710096

Chee, K. L., Meng, L. H., Wu, L., Seow, P., & Liu, L. (2018). Analysis of linkages between an unplugged activity and the development of computational thinking. *Computer Science Education*, *28*(3), 255–279. https://doi.org/10.1080/08993408.2018.1533297

Csapo Gabor. (2019). Placing Event-Action-based Visual Programming in the Process of Computer Science Education. *Acta Polytechnica Hungarica*, *16*(2). https://doi.org/10.12700/aph.16.2.2019.2.3

CSTA, & ISTE. (2011). Operational definition of computational thinking for K-12 education. *National Science Foundation*.

Curzon, P., McOwan, P. W., Plant, N., & Meagher, L. R. (2014). Introducing teachers to computational thinking using unplugged storytelling. *Proceedings of the 9th Workshop in Primary and Secondary Computing Education on - WiPSCE '14*, 89–92. https://doi.org/10.1145/2670757.2670767

Dohn, N. B. (2020). Students' interest in Scratch coding in lower secondary mathematics. *British Journal of Educational Technology*, *51*(1), 71–83. https://doi.org/10.1111/bjet.12759

Eberhard, B., Podio, M., Radovica, E., Avotina, L., Peiseniece, L., & Sendon, M. C. (2017). *Smart work: The transformer of the labour marker due to the fourth industrial revolution*.

Erümit, K. A., Karal, H., Şahin, G., Aksoy, D. A., Gencan, A. A., & Benzer, A. İ. (2019). A model suggested for programming teaching: Programming in seven steps. *Egitim ve Bilim*, *44*(197), 155–183. https://doi.org/10.15390/EB.2018.7678

Fatima, P. (2023). Optimizing Algorithm Efficiency through Advanced Data Structures in C++: A Comparative Analysis of Performance, Scalability, and Complexity. In *International Journal of Computations, Information and Manufacturing (IJCIM)* (Vol. 3, Issue 2, p. 2023). https://doi.org/10.54489/ijcim.v3i2.256

Feaster, Y., Segars, L., Wahba, S. K., & Hallstrom, J. O. (2011). Teaching CS unplugged in the high school (with

limited success). *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education - ITiCSE '11*, 248. https://doi.org/10.1145/1999747.1999817

Felicia, A., Sabariah, S., Wong, W., & Marippan, M. (2017). *View of Computational Thinking and Tinkering: Exploration Study of Primary School Students' in Robotic and Graphical Programming*. International Journal of Assessment and Evaluation in Education. https://ejournal.upsi.edu.my/index.php/AJATeL/article/view/1983/1444

Figueiredo, J., & García-Peñalvo, F. J. (2018). Building Skills in Introductory Programming. *Proceedings of the Sixth International Conference on Technological Ecosystems for Enhancing Multiculturality - TEEM'18*, 46–50. https://doi.org/10.1145/3284179.3284190

Flórez, F. B., Casallas, R., Hernández, M., Reyes, A., Restrepo, S., & Danies, G. (2017). *Changing a Generation ' s Way of Thinking : Teaching Computational Thinking Through Programming*. *XX*(X), 1–27. https://doi.org/10.3102/0034654317710096

García-Peñalvo, F. J., & Mendes, A. J. (2018). Exploring the computational thinking effects in pre-university education. *Computers in Human Behavior*, *80*, 407–411. https://doi.org/10.1016/j.chb.2017.12.005

Garvin, M., Killen, H., Plane, J., & Weintrop, D. (2019). Primary school teachers' conceptions of computational thinking. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, *2*, 899–905. https://doi.org/10.1145/3287324.3287376

Gonzalez, F., Lopez, C., & Castro, C. (2018). Development of Computational Thinking in High School Students: A Case Study in Chile. *2018 37th International Conference of the Chilean Computer Science Society (SCCC)*, *2018-Novem*(November), 1–8. https://doi.org/10.1109/SCCC.2018.8705239

Grover, S., & Jackiw, N. (2019). *Non-Programming Activities for Engagement with Foundational Concepts in Introductory Programming*. 1136–1142.

Grover, S., & Pea, R. (2013). *Computational Thinking in K–12: A Review of the State of the Field*. Educational Researcher. https://sci-hub.se/https://doi.org/10.3102/0013189X12463051

Grover, S., Pea, R., & Cooper, S. (2016). *Factors Influencing Computer Science Learning in Middle School*. 552–557. https://doi.org/10.1145/2839509.2844564

Hemmendinger, D. (2010). A plea for modesty. *ACM Inroads*, *1*(2), 4–7. https://doi.org/10.1145/1805724.1805725

Henry, J., & Dumas, B. (2018). Perceptions of computer science among children after a hands-on activity: A pilot study. *2018 IEEE Global Engineering Education Conference (EDUCON)*, *2018-April*, 1811–1817. https://doi.org/10.1109/EDUCON.2018.8363454

Ibrahim, A., Nurul Farhana, J., & Zaidatun, T. (2018). Programming Skills and the Relation in Fostering Students' Higher Order Thinking. *Asian Social Science*, *14*(11), 76. https://doi.org/10.5539/ass.v14n11p76

Khasyyatillah, I., Osman, K., & Husnin, H. (2024). *Pengintegrasian Pemikiran Komputasional melalui Aplikasi Mudah Alih untuk Meningkatkan Disposisi Pemikiran Komputasional*. *49*(2), 99–110.

Kılıç, S., Gökoğlu, S., & Öztürk, M. (2020). A Valid and Reliable Scale for Developing Programming-Oriented

Computational Thinking. *Journal of Educational Computing Research*, 073563312096440. https://doi.org/10.1177/0735633120964402

Kong, S., & Abelson, H. (2019). Computational Thinking Education. In *Computational Thinking Education*. https://doi.org/10.1007/978-981-13-6528-7

Koorsse, M., Cilliers, C., & Calitz, A. (2015). Programming assistance tools to support the learning of IT programming in South African secondary schools. *Computers & Education*, *82*, 162–178. https://doi.org/10.1016/j.compedu.2014.11.020

Lion, D., Chiu, A., Stumm, M., & Yuan, D. (2022). Investigating Managed Language Runtime Performance Why JavaScript and Python are 8x and 29x slower than C++, yet Java and Go can be faster? *Proceedings of the 2022 USENIX Annual Technical Conference, ATC 2022*, 835–851.

Lockwood, J., & Mooney, A. (2017). Computational Thinking in Education: Where does it Fit? A systematic literary review. *International Journal of Computer Science Education in Schools*, *2*(1), 1–58. http://arxiv.org/abs/1703.07659

Lockwood, J., & Mooney, A. (2018). Computational Thinking in Secondary Education: Where does it fit? A systematic literary review. *International Journal of Computer Science Education in Schools*, *2*(1), 41–60. https://doi.org/10.21585/ijcses.v2i1.26

Marcelino, M. J., Pessoa, T., Vieira, C., Salvador, T., & Mendes, A. J. (2018). Learning Computational Thinking and scratch at distance. *Computers in Human Behavior*, *80*, 470–477. https://doi.org/10.1016/j.chb.2017.09.025

MDEC. (n.d.). *MyDigitalMaker Initiative*. Laporan Program Cikgu Juara Digital. Retrieved August 3, 2025, from https://mdec.my/mydigitalmaker

Mhashi, M. M., & Alakeel, A. L. I. M. (2013). Difficulties Facing Students in Learning Computer Programming Skills at Tabuk University. *Recent Advances in Modern Educational Technologies*, 15–24. https://doi.org/1.13189/ujer.215.391

Moreno-León, J., Robles, G., & Román-González, M. (2015). Dr. Scratch: Automatic analysis of scratch projects to assess and foster computational thinking. *RED. Revista de Educación a Distancia*, *46*, 1–23.

Muhammad Ateeq, M., Hina Habib, H., Adnan Umer, A., & Muzammil Ul Rehman, M. (2014). C++ or python?: Which one to begin with: A learner's perspective. *Proceedings - 2014 International Conference on Teaching and Learning in Computing and Engineering, LATICE 2014*, 64–69. https://doi.org/10.1109/LaTiCE.2014.20

Munawar, K., & Naveed, M. S. (2022). The Impact of Language Syntax on the Complexity of Programs: A Case Study of Java and Python. *International Journal of Innovations in Science and Technology*, *4*(3), 683–695. https://doi.org/10.33411/ijist/2022040310

Nada Sharaf, N., Ahmed Ghada, G., Ahmed Adel, A., Abdennadher, S., & Berkling, K. (2019). *Koding4Kinder : Teaching Computational Thinking to Pupils 6 sing a $ ombination of Programming and Electronics Platforms*. 634–638.

Nguyen, H. (2019). *Exploring Python as a replacement for C ++ in Imperative Programming for Computing Science at Radboud University*. Radboud University.

Ogegbo, A. A., Adebunmi, ·, & Aina, Y. (2024). *Exploring young students' attitude towards coding and its relationship with STEM career interest*. *29*, 9041–9059. https://doi.org/10.1007/s10639-023-12133-5

Palts, T., & Pedate, M. (2020). A Model for Developing Computational Thinking Skills. *Informatics in Education*, *19*(1), 113–128. https://doi.org/10.15388/infedu.2020.06

Psycharis, S., & Kallia, M. (2017). *The effects of computer programming on high school students' reasoning skills and mathematical self-efficacy and problem solving*. https://doi.org/10.1007/s11251-017-9421-5

Qian, Y., & Lehman, J. (2017). *Students ' Misconceptions and Other Difficulties in Introductory Programming : A Literature Review*. *18*(1), 1–24.

Rodriguez, B., Kennicutt, S., Rader, C., & Camp, T. (2017). *Assessing Computational Thinking in CS Unplugged Activities*. 501–506.

Rodriguez, B., Rader, C., & Camp, T. (2016). Using Student Performance to Assess CS Unplugged Activities in a Classroom Environment. *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE '16*, 95–100. https://doi.org/10.1145/2899415.2899465

Salayev. (2024). CONSIDERATIONS ON MODERN PROGRAMMING LANGUAGES. *Mental Enlightenment Scientific – Methodological Journal*, *4*(2), 175–184.

Salleh Hudin, S. (2023). A Systematic Review of the Challenges in Teaching Programming for Primary Schools' Students. *Online Journal for TVET Practitioners*, *8*(1). https://doi.org/10.30880/ojtp.2023.08.01.008

Shepherd, D., Francis, P., Weintrop, D., Franklin, D., Li, B., & Afzal, A. (2018). An IDE for easy programming of simple robotics tasks. *Proceedings - 18th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2018*, 209–214. https://doi.org/10.1109/SCAM.2018.00032

Siti Rosminah M. D. Derus, A. Z. M. A. (2012). Difficulties in learning programming: Views of students. *1st International Conference on Current Issues in Education (ICCIE 2012)*, *September 2012*, 74–79. https://doi.org/10.13140/2.1.1055.7441

Sivaraman, A., Zhang, T., Van den Broeck, G., & Kim, M. (2019). Active Inductive Logic Programming for Code Search. *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 292–303. https://doi.org/10.1109/ICSE.2019.00044

So, H. J., Jong, M. S. Y., & Liu, C. C. (2020). Computational Thinking Education in the Asian Pacific Region. *Asia-Pacific Education Researcher*, *29*(1), 1–8. https://doi.org/10.1007/s40299-019-00494-w

Thies, R., & Vahrenhold, J. (2013). *On plugging "unplugged" into CS classes*. 365. https://doi.org/10.1145/2445196.2445303

Thies, R., & Vahrenhold, J. (2012). Reflections on outreach programs in CS classes. *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education - SIGCSE '12*, 487. https://doi.org/10.1145/2157136.2157281

Tsai, M.-J., Wang, C.-Y., & Hsu, P.-F. (2019). Developing the Computer Programming Self-Efficacy Scale for Computer Literacy Education. *Journal of Educational Computing Research*, *56*(8), 1345–1360.

https://doi.org/10.1177/0735633117746747

Tuomi, P., Multisilta, J., Saarikoski, P., & Suominen, J. (2018). Coding skills as a success factor for a society. *Education and Information Technologies*, *23*(1), 419–434. https://doi.org/10.1007/s10639-017-9611-4

Umezawa, K., Nakazawa, M., & Hirasawa, S. (2023). Comparison of Biometric Information during Learning of Visual- and Text-Based Programming Languages. *8th International STEM Education Conference, ISTEM-Ed 2023 - Proceedings*. https://doi.org/10.1109/ISTEM-ED59413.2023.10305730

Ung, L. L., Labadin, J., & Nizam, S. (2021). Development of a Rubric to Assess Computational Thinking Skills among Primary School Students in Malaysia. *ESTEEM Academic Journal*, *17*(August), 11–22.

Voogt, J., Erstad, O., Dede, C., & Mishra, P. (2013). Challenges to learning and schooling in the digital networked world of the 21st century. *Journal of Computer Assisted Learning*, *29*(5), 403–413. https://doi.org/10.1111/jcal.12029

Wen, F. H., Wu, T., & Hsu, W. C. (2023). Toward improving student motivation and performance in introductory programming learning by Scratch: The role of achievement emotions. *Science Progress*, *106*(4), 1–21. https://doi.org/10.1177/00368504231205985

Wing, J. M. (2010). Computational Thinking: What and Why? *Thelink - The Magazine of the Varnegie Mellon University School of Computer Science*, *March 2006*, 1–6. http://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why

Wing, J. M. (2006). Computational thinking. *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education - ITiCSE '11*, *49*(3), 223. https://doi.org/10.1145/1999747.1999811

Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., & Korb, J. T. (2014). Computational thinking in elementary and secondary teacher education. *ACM Transactions on Computing Education*, *14*(1), 1–16. https://doi.org/10.1145/2576872

Yuana, R. A., & Maryono, D. (2016). Robomind Utilization to Improve Student Motivation and Concept in Learning Programming. *Ictte*, *1*, 962–966.

Yuen, T. T., Reyes, M., & Yuanlin, Z. (2018). *Introducing Computer Science to High School Students through Logic Programming*. *19*(November 2018), 204–228. https://doi.org/10.1017/S1471068418000431