Design and Development of Computational Thinking-Based Laboratory Worksheet for Text-Based Programming

Fariza Zahari[1, 2*]& Ahmad Zamzuri Mohamad Ali[1]
[1]Faculty of Art, Sustainability & Creative Industry, Universiti Pendidikan Sultan Idris, Malaysia
[2]Electrical Engineering Department, Politeknik Sultan Salahuddin Abdul Aziz Shah, Malaysia
**Corresponding author: Fariza Zahari (farizahari80@gmail.com)*

Received: 25 February 2026
Received in revised form: 25 May 2026
Accepted: 18 June 2026
Published: 11 November 2026

ABSTRACT

Programming is an essential skill in 21st-century education, yet students often experience difficulties in learning text-based programming, which are closely associated with limited computational thinking (CT) skills. To address this issue, this study adopts a design development research approach to develop laboratory worksheets for teaching text-based programming guided by the Learning Model of Computational Thinking and the Programming in Seven Steps (PSS) framework. The worksheets were designed to provide a structured and scaffolded learning experience that supports the development of CT skills. The usability and user experience of the developed worksheets were evaluated using the User Experience Questionnaire (UEQ) involving n = 28 students. The UEQ measures six dimensions: attractiveness, clarity, efficiency, dependability, stimulation, and novelty. The findings indicate that students reported a highly positive user experience across all dimensions. Overall, the results suggest that well-designed CT-based laboratory worksheets can enhance usability and promote a more engaging learning experience in text-based programming. These findings contribute to the development of effective instructional materials that support computational thinking in programming education.

Keywords:
Computational thinking; Laboratory worksheet; Text-based programming; User experience questionnaire

Introduction

Programming is widely regarded as an essential form of twenty-first-century literacy that plays an important role in supporting national technological development and competitiveness (Rosli et al., 2024). It constitutes a fundamental component of innovation in information technology. Today, a wide range of technologies, including robotics, artificial intelligence, virtual learning, cryptocurrency, the Internet of Things, and cloud computing, are largely driven by programming languages. (Khangate et al., 2024; Ogegbo & Adebunmi, 2024).

The contemporary job market has been reshaped by the rapid growth of digital technology, with computing-related fields such as software development, web and internet application development, artificial intelligence, and cybersecurity playing a dominant role across diverse industries and sectors (Barbosa et al., 2024; Mäkelä & Stephany, 2024; Ramos et al., 2025). Given the importance of this field, programming courses have been introduced as early as the school level, where programming is embedded as a foundational component of modern curricula. Furthermore, most universities and polytechnics have established programming as a core course in the early stages of study.

However, meeting this demand remains challenging because relatively few individuals are interested in pursuing a profession in programming (Abdunabi et al., 2019). Although enrolment in Computer Science and Information Technology programs is high, students' programming proficiency often falls short of the current demands of the computing industry (Cazalas et al., 2024; Dasuki & Quaye, 2016).

Programming is widely regarded as a challenging subject for students in universities, polytechnics, and colleges due to several factors. In particular, students frequently struggle to comprehend the step-by-step execution of code (Mhashi & Alakeel, 2013), to identify different types of variables, and to understand expressions, Boolean logic, loops, and

e-ISSN 2600-8572 | DOI: https://doi.org/10.11113/itlj.v10.270

abstractions (Singh, 2022; Weeda et al., 2024). They also encounter difficulties with fundamental syntax, such as the correct use of semicolons, parentheses or brackets, and quotation marks (Qian & Lehman, 2017). Beyond these technical challenges, numerous studies have shown that a lack of programming problem-solving skills is a primary contributor to students' difficulties in mastering the discipline (Figueiredo, 2017; Mhashi & Alakeel, 2013; Tuomi et al., 2018).

Literature Review

Computational Thinking Skills

Programming problem-solving skills are closely associated with computational thinking, as they involve translating real-world problems into step-by-step instructions that a computer can execute (Liu et al., 2024; Wu et al., 2024). It is important to note that computational thinking does not imply "thinking like a computer"; rather, it refers to understanding how computers carry out problem-solving processes (García-Peñalvo & Mendes, 2018; Hemmendinger, 2010; Wing, 2008; Yadav et al., 2014). In general, computational thinking comprises five key elements of problem-solving: decomposition, pattern recognition, abstraction, algorithmic thinking, and generalisation (Looi et al., 2018; Mohamad Ali, 2019).

Unplugged activities are an instructional technique for improving computational thinking skills in schools that has received increasing scholarly attention (Bell, 2019; Rodriguez et al., 2017; Thies & Vahrenhold, 2013). These activities involve kinaesthetic and interactive learning experiences that introduce fundamental computational ideas without the use of digital equipment or formal programming languages (Bell, 2019; Rodriguez et al., 2017). Although the lack of real code may seem to limit their educational depth, empirical data suggests otherwise. Prior research has shown that unplugged activities, such as binary representation, search algorithms, and network-based sorting activities, help learners conceptualise abstract computational processes through structured hands-on tasks (Bell & Vahrenhold, 2018; Curzon et al., 2014; Looi et al., 2018; Rodriguez et al., 2017). Furthermore, their teaching value goes beyond introductory programming ideas. Unplugged activities, in general, serve as cognitive scaffolds for learners' problem-solving abilities and logical reasoning, allowing computational thinking ideas to be transferred across multiple disciplinary contexts (Curzon et al., 2014).

Although unplugged activities can enhance students' comprehension of computational thinking concepts, a critical question persists: to what these activities facilitate a deeper grasp of programming and the capacity to write code? This is due to the scarcity, or absence, of longitudinal research examining the possible effects of unplugged activities on the cultivation of programming abilities. Moreover, some short-term researches have indicated that students struggle to discern explicit links between unplugged activities and fundamental programming ideas (Henry & Dumas, 2018; Taub et al., 2012). Unplugged activities fail to encompass the entirety of programming-related tasks, such as accurate code writing, compiler utilisation, and debugging processes (Taub et al., 2012). Furthermore, unplugged activities have demonstrated minimal positive impacts among secondary school students, as they predominantly utilise kinaesthetic methods that tend to be more engaging for primary school learners (Feaster et al., 2011). Consequently, unplugged activities must be meticulously crafted to align with students' cognitive levels, ensuring they transcend mere amusement and yield significant learning effects.

In addition to unplugged activities, Programming Assistance Tools (PATs) are extensively used to help students build computational thinking abilities (Chen et al., 2025; Koorsse et al., 2015; Yuana & Maryono, 2016). PATs are application-based software designed to introduce programming principles and improve students' understanding through guided and supportive learning experiences. These tools often offer a simplified Integrated Development Environment (IDE), with drag-and-drop features that simplify technical tasks for inexperienced users (Shepherd et al., 2018). There are numerous open-source PATs available, including Scratch, Alice, TouchDevelop, Beetle Blocks, Kodu, Code.org, AgentSheets, and Blockly. Prior studies have found Scratch (Dohn, 2020; García-Peñalvo & Mendes, 2018) and Alice (Csapó, 2019; Sivaraman et al., 2019) as successful platforms for teaching and learning programming fundamentals at the school level.

Scratch and Alice are drag-and-drop programming environments that eliminate the need for students to write code or deal with syntax problems (García-Peñalvo & Mendes, 2018; Koorsse et al., 2015). While Programming Assistance

Tools can enhance students' engagement and comprehension of programming principles, research indicates that learners frequently struggle to apply these concepts to practical coding assignments (Henry & Dumas, 2018). This limitation occurs because actual programming practice relies on text-based coding, that requires students to comprehend and generate formal programming syntax, while PATs utilize an object-oriented, block-based technique where commands are created by arranging visual blocks within a programming environment (Lye et al., 2014). As a result, the representational gap between text-based programming and block-based environments may limit students' capacity to internalise syntactic rules and to transmit conceptual understanding to real-world coding contexts.

Unplugged activities and Programming Assistance Tools, in particular, are helpful in promoting conceptual knowledge; nevertheless, their direct impact to the acquisition of real programming writing abilities is unclear. As a result, these approaches are better suited to developing computational thinking skills among primary and lower secondary students (Mohamad Ali, 2019). Conversely, at the upper secondary and pre-university levels, teaching methodologies should progressively prioritise the utilisation of authentic text-based programming languages to adequately equip students for the cognitive and technical requirements of programming courses at the tertiary level (Gonzalez et al., 2018; Yuen et al., 2019).

Consequently, it is evident that a structured instructional arrangement is required to facilitate the understanding of the fundamental principles of text-based programming by higher education students and to facilitate the systematic progression to more sophisticated problem-solving tasks. To meet this requirement, it is necessary to develop a set of laboratory worksheets that are pedagogically informed and utilise scaffolded progression to facilitate incremental learning and reduce cognitive strain. The capacity of these worksheets to cultivate robust conceptual understanding and to encourage the sustained development of problem-solving skills in text-based programming contexts is further enhanced by the integration of computational thinking elements.

Programming Language for Novice

Prior research indicates that selecting an appropriate programming paradigm and language is crucial in assisting novice learners. Structured programming has been selected as the best strategy because it provides a clear and systematic framework for program development (Hendrix et al., 2018; Mohamad Ali, 2019). This method allows students to understand control flow, variables, and decision structures without being overwhelmed by the high cognitive demands associated with sophisticated abstractions seen in object-oriented paradigms.

Although a structured paradigm can accommodate a variety of programming languages, empirical evidence indicates that not all languages yield comparable pedagogical results. For example, students have been criticised for having to comprehend concepts such as inheritance, encapsulation, classes, and objects before they can construct fundamental programs in languages like Java (Dasuki & Quaye, 2016; Khoirom et al., 2020). This level of complexity frequently introduces extraneous cognitive burden, resulting in learners concentrating more on syntactic and structural issues than on core problem-solving principles. This challenge has also been linked to a decrease in learning motivation, recurrent syntax errors, and increased confusion among novice students in numerous studies.

Conversely, Python has become the preferred option in educational institutions as a consequence of its user-friendly data structures, concise syntax, and simplicity of learning (Ling et al., 2021). These features not only assist in the reduction of syntax-related errors but also enable students to focus on the development of logical reasoning and problem-solving processes rather than the struggle with intricate syntactic rules (Ateeq et al., 2014; Kadam & Oyelere, 2026). Consequently, the utilisation of Python in this investigation is consistent with the extant literature that emphasises its efficacy in instructing structured programming to novice learners.

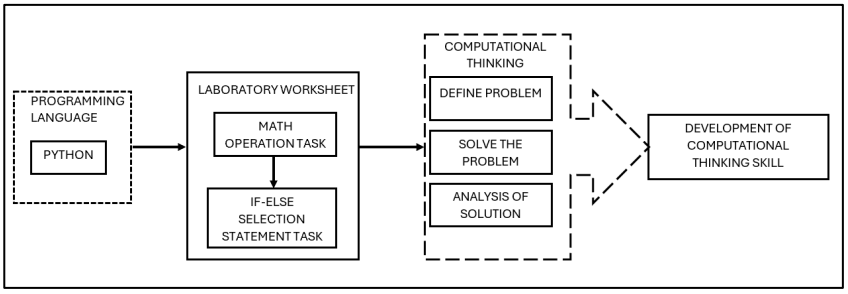
Conceptual Model

The conceptual model of this study, as illustrated in Figure 1 demonstrates the relationship between the study's variables and the computational thinking skills model. The Python programming languages are essential for the development of laboratory activities that are intended to cultivate computational thinking within this framework. The laboratory worksheets 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

e-ISSN 2600-8572 | DOI: https://doi.org/10.11113/itlj.v10.270

Thinking presents a thorough framework for recognising and cultivating essential computational thinking abilities, whereas the PSS Model delivers a systematic, sequential approach to programming education. This work aims to provide successful instructional tools by merging these two models, so facilitating the systematic and accessible acquisition of computational thinking skills, notably through Python programming activities. These tools seek to connect theoretical principles with practical applications, thereby improving students' problem-solving skills and programming expertise.

Figure 1
Conceptual Model

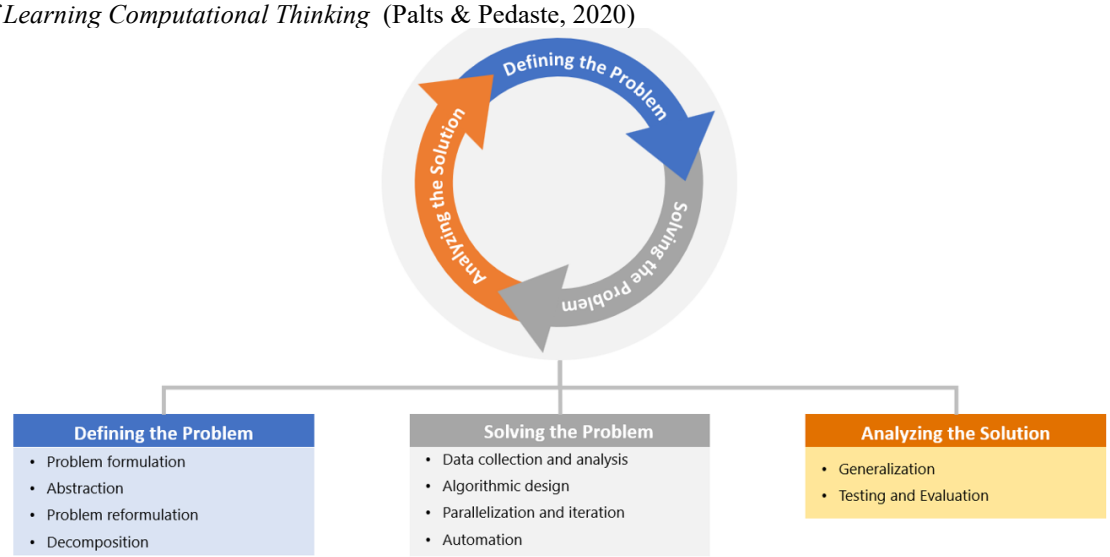


The laboratory worksheets included five elements of computational thinking skills: decomposition, pattern recognition, abstraction, algorithm design, and program writing and evaluation. The computational thinking skills are classified into three primary categories within the study's conceptual framework: conceptual knowledge, algorithmic thinking, and evaluation. The selection of an appropriate programming language is crucial for developing computational thinking skills, hence preventing excessive anxiety among students during the learning process. The project intends to establish an environment that enables students to engage with computational thinking principles efficiently and confidently through carefully developed laboratory worksheets, thereby enhancing their comprehension and practical application of programming.

Model of Learning Computational Thinking

The Model for the Development of Computational Thinking Skills, as illustrated in Figure 2, consists of three primary phases of computational thinking.

Figure 2
Model of Learning Computational Thinking (Palts & Pedaste, 2020)



e-ISSN 2600-8572 | DOI: https://doi.org/10.11113/itlj.v10.270

There are three components: problem identification, problem resolution, and solution analysis. In the problem identification phase, four skills are required: problem formulation, abstraction, problem reformulation, and decomposition. The process of problem-solving initiates with the identification of the problem, which is known as problem formulation. Following this, the ability to abstract is emphasised, which involves concentrating on the most critical aspects while disregarding less significant elements. To transform a problem into one that has been previously encountered or solved, problem reformulation can be implemented. Lastly, decomposition implies the identification of the inputs, outputs, and processes necessary to address the problem, thereby dividing it into smaller components.

The second phase of the model involves problem solving. This phase focuses on four key problem-solving skills: data collection and analysis, algorithm design, parallelization and iteration, and automation. The third phase is the analysis of solutions, which includes the ability to generalise and extend the problem-solving process to broader circumstances. Finally, testing and evaluation focus on analysing and assuring that the output accurately meets the problem's requirements. This technique is continued iteratively until the final output matches the users' expectations.

Programming in Seven Steps (PSS) Model

The model was developed to serve as a systematic reference for educators aiming to instruct programming at the secondary school level. The model was developed through a joint effort between the design team and the design-evaluation team. The study's findings resulted in the development of seven essential stages of programming training, designed to assist teachers in efficiently delivering programming education. The sequence comprises seven steps: problem comprehension, algorithm design, implementation of the method into code, execution of tests, error detection and correction (debugging), program execution, and evaluation of the final solution.

Integration of the Model of Learning Computational Thinking into Programming in Seven Steps (PSS) Model

The Programming in Seven Steps (PSS) Model can be integrated into the Model of Learning Computational Thinking, as both emphasise a systematic approach to problem solving. The procedural phase in PSS, which encompass solution planning, algorithm formulation, code implementation, and the detection and correction of errors, directly cultivate fundamental computational thinking constructs such as abstraction, algorithmic reasoning, and debugging (Brennan & Resnick, 2012). Therefore, this integration has the potential to enhance students' ability to apply computational thinking in a practical manner through structured programming activities (Grover & Pea, 2013). Table 1 illustrates the integration between the Programming in Seven Steps (PSS) Model and the Model of Learning Computational Thinking.

Table 1
Integration Model Programming in Seven Steps and 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 analysing the problem, employing decomposition and abstraction techniques to break it down into manageable components
Solving the problem	Step 2: Design Algorithm	Planning the solution steps in a structured and systematic manner
	Step 3: Coding	Write code based on the designed algorithm.
	Step 5: Run	Test the program results to ensure they meet requirements.

e-ISSN 2600-8572 | DOI: https://doi.org/10.11113/itlj.v10.270

Model of Learning Computational Thinking	Model Programming in Seven Step	Integration Explanation
	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 regarded as a supplementary framework that provides a hands-on, action-based approach to implementing the skills of the Computational Thinking Learning Model. By combining the two models, students' comprehension of programming concepts can be enhanced, while their computational thinking abilities are further developed through learning experiences that are structured, reflective, and directly applicable.

Research Questions

The research questions of this research are:

1. How can laboratory worksheets for teaching text-based programming be developed based on the Learning Model of Computational Thinking and the Programming in Seven Steps (PSS) framework?
2. What is the effect of employing the Python text-based programming language on the development of computational thinking skills among Polytechnic students?

Python was selected due to its concise and easily interpretable syntax, its high level of code readability, and its suitability as an introductory programming language capable of reducing students' cognitive load (Vidal-Silva et al., 2022). These attributes allow students to focus more effectively on reasoning processes, problem-solving, and the cultivation of computational thinking, rather than being constrained by syntactic complexities inherent in programming (Bai et al., 2021; Vidal-Silva et al., 2022).

To achieve the objectives of the study, a set of laboratory worksheets was developed by guided to the Model of Learning Computational Thinking (Palts & Pedaste, 2020), the Programming in Seven Steps (PSS) Model (Erümit et al., 2019), as well as the proposed conceptual model. The approach adopted in these laboratory worksheets is based in text-based programming and is specifically designed to cultivate students' computational thinking skills through systematically structured and progressively scaffolded Python programming activities.

Methods

The ADDIE model was selected for the design and development phase of this study because it is widely employed in the creation of computer-based instructional materials. It comprises five key stages: analysis, design, development, implementation, and evaluation (Arnab et al., 2022; Erümit et al., 2019).

To evaluate the developed laboratory worksheets, a quantitative survey research design was employed. The evaluation was conducted using the User Experience Questionnaire (UEQ) with n 28 students from a cohort of fifth-semester students who had prior exposure to fundamental programming courses. This approach enabled the systematic assessment of students' perceptions of usability and user experience across multiple dimensions.

Results

Analysis

The initial stage of the ADDIE Model is the analysis phase. This step is essential as it establishes the underlying structure for the entire instructional design process. The analysis step encompasses four critical components: needs

analysis, identification of developmental objectives, comprehension of students' background characteristics, and the selection of suitable instructional and delivery methodologies. An initially requires analysis must be performed to guarantee that the developed laboratory worksheets and research instruments effectively meet the study's objectives (Amat Yasin et al., 2021). Needs analysis is a methodological technique employed to discern discrepancies between the existing state and the intended condition. Various methodological approaches can be implemented to perform a requirements analysis, conditional upon the context and scope of the study (Abdul Muqsith, 2018). Two fundamental methodologies were implemented: unstructured interviews and an extensive literature review utilising the Google Scholar database.

An unstructured interview was performed with multiple lecturers instructing programming courses at the polytechnic. The interview results indicated that numerous students encounter difficulties with problem-solving tasks in programming. They frequently encounter challenges in devising the sequential procedures necessary for addressing programming inquiries. This sequential process is, in fact, intricately connected to computational thinking abilities.

The subsequent method entailed performing a search and examining pertinent literature via the Google Scholar database. Most prior research concentrated on computational thinking skills in relation to Programming Assistance Tools (PAT) like Scratch and Alice. Conversely, a limited number of studies have explored the cultivation of computational thinking skills inside text-based programming environments, including languages like Python and C++.

The laboratory worksheet and research instrument were designed to investigate the impact of utilising the Python programming language on the development of computational thinking skills among polytechnic students. A laboratory worksheet was developed specially for this study to help students achieve the desired learning objectives through structured in-class activities. The intended users of the laboratory worksheet were first-semester students participating in the Diploma in Medical Electronic Engineering and Diploma in Communication Engineering programs. These students were chosen for their ability to advance in computing or engineering disciplines. The study sample comprised students with restricted ability in text-based programming and computational thinking abilities. A screening test was conducted before the trial to confirm compliance with this criterion.

The laboratory worksheet and research instruments were developed by integrating computational thinking skills as an instructional and delivery strategy to encourage students to engage in step-by-step problem-solving processes. Students were required to organise and plan each programming solution systematically prior to code implementation.

Design

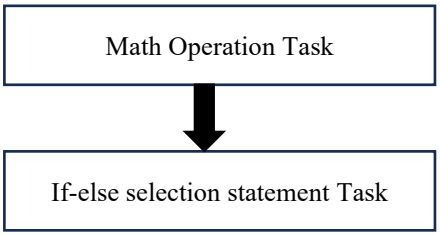
The design phase focused on planning the content, instructional strategies, learning activities, and assessment components of the laboratory worksheet and research instruments. In this study, the design of the laboratory worksheet was informed by the Model of Learning Computational Thinking and the Programming in Seven Steps (PSS) Model. The activities undertaken during this phase involved the design process and the analysis of concepts and learning activities.

Five laboratory worksheets have been developed adopting the Model of Learning Computational Thinking and the Programming in Seven Steps (PSS) Model. Each worksheet had six components of computational thinking integrated inside the text-based programming problem-solving process: decomposition, pattern recognition, abstraction, algorithmic thinking, program implementation, and program evaluation. The laboratory worksheets' content structure was developed in accordance with the Fundamental Programming syllabus provided in Malaysian polytechnics. The laboratory worksheets were developed using a scaffolded methodology in which learning activities progressed systematically from decomposition, analysing problems into smaller elements, to program evaluation. This method was adopted to enhance students' conceptual comprehension and to promote the incremental advancement of computational thinking abilities (Lee et al., 2023; Wang et al., 2024). Furthermore, each educational task within the laboratory worksheets included instructor-led interventions designed to augment understanding and elevate student learning results (Mallillin, 2022; Qobilovna, 2023; Setyorini, 2020).

The main objective of the laboratory worksheets, as determined during the analysis phase, was to enhance students' computational thinking abilities via text-based programming. The worksheets were organised progressively, starting

with problem identification (decomposition, pattern recognition, and abstraction), advancing to problem-solving (algorithm design), and concluding with solution analysis (program development and testing), as depicted in Figure 3. This sequence was structured according to specified computational thinking skill models.

Figure 3
Laboratory Worksheet Sequence



Development

Guided by the study's conceptual framework, a set of laboratory worksheets was developed using the Python programming language. The worksheets were designed in alignment with key computational thinking skills. Table 2 provides a description of the activities incorporated into the developed laboratory worksheets.

Table 2
Laboratory Worksheet Activities

Laboratory Worksheet	Activities
Laboratory Worksheet 1	Early exposure to problem-solving in programming is facilitated through the application of computational thinking skills. During the learning activities under lecturer-guided intervention, students are required to develop a program capable of generating the output for the multiplication of three integer values.
Laboratory Worksheet 2	Using a computational thinking approach, students developed programs to solve a spherical volume calculation problem with lecturer-led intervention.
Laboratory Worksheet 3	Using a computational thinking approach, students developed programs to solve a surface area calculation problem involving a box, with lecturer-led intervention.
Laboratory Worksheet 4	Using a computational thinking approach, students developed programs to classify user groups based on age categories, with lecturer-led intervention.
Laboratory Worksheet 5	Using a computational thinking approach, students developed programs to compute the sum of two numbers and subsequently classify the result as accepted or rejected, with lecturer-led intervention.
Laboratory Worksheet 6	Computational Thinking Test.

Six laboratory worksheets were developed based on the Model of Learning Computational Thinking and the Programming in Seven Steps (PSS) Model. The worksheets were organised using a scaffolded structure, beginning with problem decomposition, pattern recognition, abstraction, and algorithm design, followed by program implementation and concluding with program testing. Each worksheet employed a consistent learning activity structure aligned with computational thinking skills. Lecturer-led interventions were incorporated in laboratory worksheets 1 to 5 to support students in developing computational thinking skills through text-based programming. Laboratory worksheet 6 functioned as the Computational Thinking Test.

Figures 3 to 11 show examples of the laboratory worksheets developed in accordance with the sequence of computational thinking skills.

Figure 3
Learning Questions in the Laboratory Worksheet

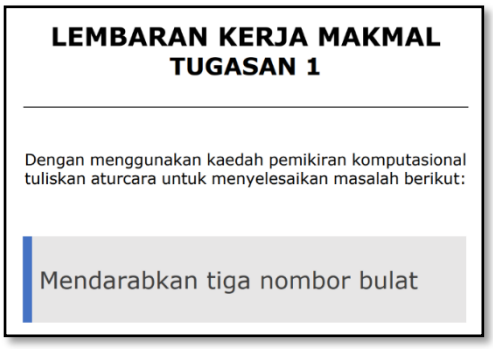


Figure 4
Decomposition Skill

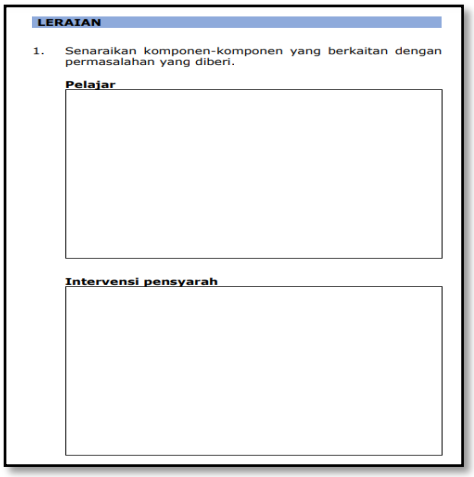
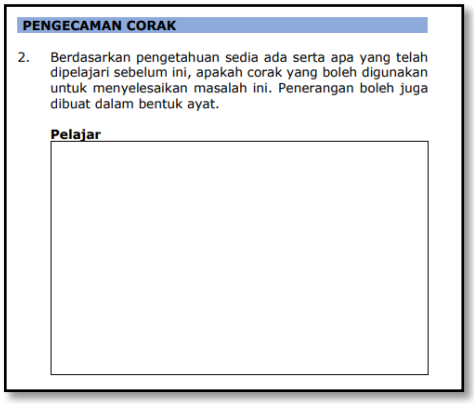


Figure 5
Pattern Recognition



e-ISSN 2600-8572 | DOI: https://doi.org/10.11113/itlj.v10.270

Figure 6
Input, Output and Process Component

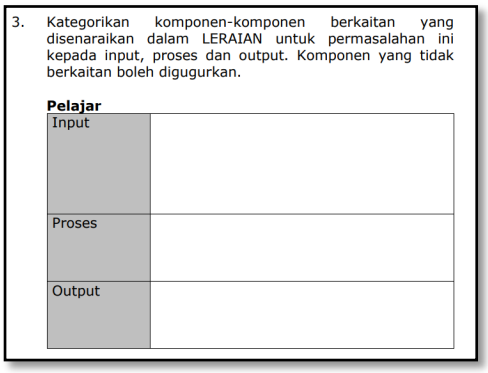


Figure 7
Variable and Data Types

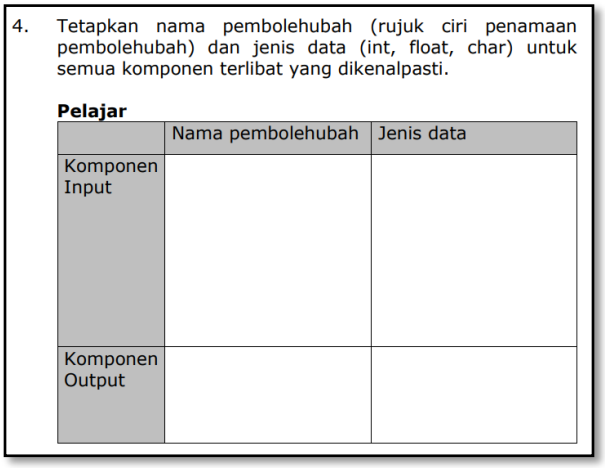


Figure 8
Writing Processes in Programming Format

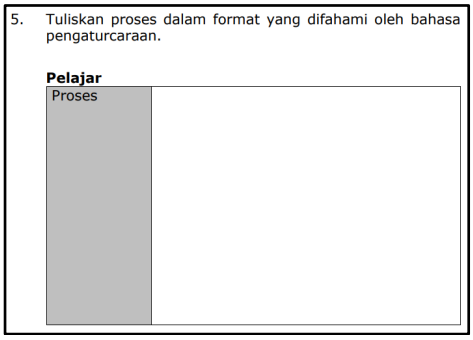


Figure 9
Algorithm

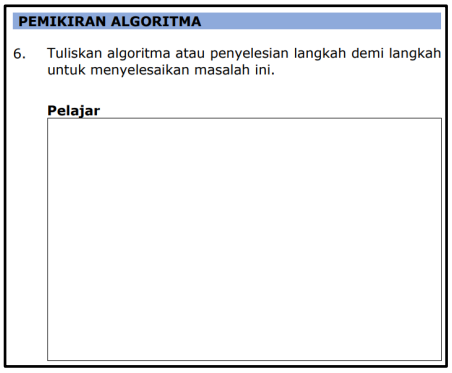


Figure 10
Programming Writing

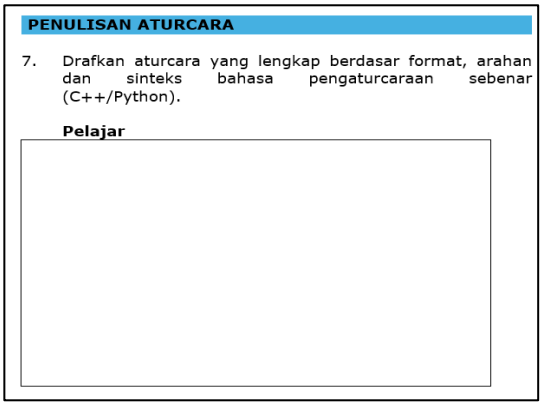
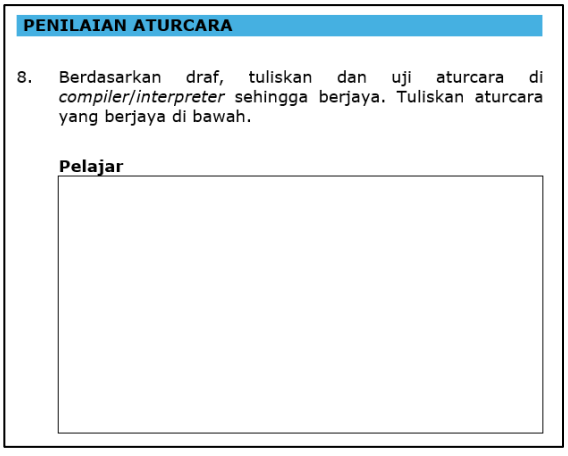


Figure 11
Programming Evaluation



Evaluation

The evaluation was carried out in two phases: formative and summative evaluations. The formative phase occurred during the development of the laboratory worksheets, aimed at detecting deficiencies and enhancing certain components within the worksheets. Data was collected via a literature study, interviews, and expert comments. The expert panel consisted of polytechnic instructors and university scholars with over five years of experience in programming. The experts unanimously agreed on the form and quality of the developed laboratory worksheets. The worksheets are well-structured and effectively facilitate the development of computational thinking skills in a systematic manner.

The summative evaluation procedure was utilised to gather feedback from students during the design and development of the laboratory worksheets. Upon the completion of the worksheets, a summative assessment was performed to evaluate their usefulness as well as the level of user satisfaction during the preliminary study. A cohort of fifth-semester students, who had previously completed the Introduction to Programming course, was chosen to engage in the usability and user satisfaction evaluation. Before interacting with the laboratory worksheets, students received foundational notes from the programming workshop. They were thereafter assigned five hours of lecture sessions to finalise the prepared worksheets. Upon conclusion of the learning session, students were mandated to complete the User Experience Questionnaire (UEQ).

The User Experience Questionnaire (UEQ) is an instrument developed to assess users' experience with a product, system, or service, including laboratory worksheets (Laugwitz et al., 2008). The UEQ employs a semantic differential approach using pairs of opposing adjectives, such as difficult to understand/easy to understand, annoying/pleasant, clear/confusing, and conservative/innovative. The questionnaire consists of 26 items organised into six dimensions: attractiveness, clarity, efficiency, dependability, stimulation, and novelty. Each item is rated using a Likert-type scale ranging from 1 to 7.

The reliability study employing Cronbach's alpha produced a coefficient of 0.907 for the 26 items. A Cronbach's alpha value of 0.70 is typically deemed adequate for research, although values surpassing 0.80 signify robust dependability (Nunnally, 1978). The coefficient derived from this usability and user satisfaction evaluation indicates that the laboratory worksheets possess excellent internal consistency and are appropriate for the intended measurement in the study (Hair et al., 2019). Table 3 presents the mean and standard deviation for each dimension.

Table 3
UEQ Score for Each Dimension (n = 28)

Dimension	Mean (M)	Standard Deviation (SD)	Interpretation
Attraction	6.8452	0.32371	Very positive
Clarity	6.9345	0.16567	Very clear and easy to understand
Efficiency	6.8214	0.31074	Effective and efficient
Reliability	6.5000	0.87315	Acceptable and reliable
Simulation	6.8304	0.30470	Give motivation and enthusiasm
Novelty	6.7857	0.35168	Creative and innovative

Discussion

The first objective of this study was to develop laboratory worksheets for teaching text-based programming based on the Learning Model of Computational Thinking and the Programming in Seven Steps (PSS) framework. The design of the laboratory worksheets emphasised several important instructional design elements intended to support novice learners in developing computational thinking skills systematically through text-based programming activities.

One of the main design elements incorporated into the worksheets was scaffolded progression. The learning activities were organised sequentially, beginning with problem decomposition and pattern recognition before progressing towards abstraction, algorithm construction, code implementation, debugging, and program evaluation. This structured arrangement was intended to reduce students' cognitive burden while gradually strengthening their ability to solve

e-ISSN 2600-8572 | DOI: https://doi.org/10.11113/itlj.v10.270

programming problems systematically. Such a scaffolded approach is particularly important for novice learners, as beginners often experience difficulties in connecting programming syntax with underlying logical reasoning and problem-solving processes.

In addition, the laboratory worksheets integrated computational thinking components into every stage of the learning activities. Students were required to identify inputs, outputs, processes, algorithms, and evaluation procedures before implementing program code. This design encouraged students to engage actively in computational thinking practices rather than focusing solely on syntax memorisation. Lecturer-guided interventions and structured prompting techniques were also embedded within the worksheets to assist students in analysing problems step-by-step, reflecting on their solutions, and improving debugging processes during programming activities. Furthermore, the worksheets utilised authentic text-based programming tasks using Python to ensure that students were exposed to realistic coding practices aligned with higher education learning requirements.

The usability evaluation results provide strong empirical support for the effectiveness of the developed worksheets. The clarity dimension achieved the highest mean score ($M = 6.93$, $SD = 0.17$), signifying that the laboratory worksheets were comprehensible and did not cause confusion during utilisation. The exceptional clarity demonstrates the efficacy of the interface design and the methodical arrangement of the content, both of which are critical for usability and seamless user interaction. Likewise, the attractiveness dimension exhibited a high mean score ($M = 6.85$, $SD = 0.32$), indicating that the product effectively captivated people technically and emotionally, therefore enhancing total user approval. The findings underscore that clarity and the appearance are essential in improving user experience and promoting increased user engagement during the learning process (Zuriana et al., 2025).

The dimension of efficiency ($M = 6.8214$) and dependability ($M = 6.5000$) both reflected acceptable perceptions concerning the effectiveness and reliability of the laboratory worksheets in use. The dependability measure had a notably greater standard deviation ($SD = 0.87315$), indicating variability in user verdicts. This heterogeneity may be ascribed to disparities in users' prior experience or familiarity with the worksheets. Nonetheless, the constantly elevated mean score indicates that the worksheets are predominantly well-received and considered reliable by most users.

The stimulation ($M = 6.8304$) and novelty ($M = 6.7857$) dimensions also recorded high scores, reflecting the worksheets' ability to motivate learners and provide a creative and innovative learning experience. Effective stimulation elements have the potential to enhance users' active engagement, whereas the novelty aspect contributes to a more appealing and less monotonous learning process (Alemneh et al., 2025).

Overall, the high usability and satisfaction score ($M = 6.76$, $SD = 0.38$) confirms that the developed laboratory worksheets achieved Objective 1. The findings demonstrate that the integration of the Learning Model of Computational Thinking and the PSS framework resulted in instructional materials that are clear, engaging, reliable, and effective in supporting the development of computational thinking skills among Polytechnic students.

Conclusion

The design of the laboratory worksheets, based on computational thinking skills, provides a systematic and incremental learning module that assists students in enhancing problem-solving capabilities, especially in text-based programming. By adopting computational thinking approaches like decomposition, pattern identification, abstraction, algorithmic thinking, coding, and evaluation. The worksheets serve as an instructional tool that efficiently merges academic comprehension with practical application. In conclusion, the implementation of laboratory worksheets grounded in computational thinking can significantly improve students' conceptual comprehension while systematically advancing their computational thinking abilities. This work solves a research gap by offering organised learning resources specialised for text-based programming. The worksheets reveal the potential to enhance teaching and learning experiences, elevate programming education outcomes in higher education, and create opportunities for future research to broaden the applications of computational thinking beyond many programming domains.

Conflict of Interest

Authors declare that there is no conflict of interest regarding the publication of the paper.

e-ISSN 2600-8572 | DOI: https://doi.org/10.11113/itlj.v10.270

Acknowledgements

The authors would like to express their sincere appreciation to the Faculty of Art, Sustainability & Creative Industry at Universiti Pendidikan Sultan Idris for their professional collaboration and valuable academic input. Appreciation is also extended to the Electrical Engineering Department at Polytechnic Sultan Salahuddin Abdul Aziz Shah for facilitating this study and providing essential resources. The authors acknowledge the expert panel for their valuable feedback in validating the research instruments. Finally, sincere thanks are extended to all participating students for their cooperation and active involvement, which were crucial to the successful completion of this research.

References

Abdul Muqsith, A. (2018). Pembangunan model ENi berasaskan aktiviti inkuiri bagi program latihan kemahiran kejuruteraan Institut Latihan Kemahiran Malaysia. In *Tesis Ijazah Doktor Falsafah*.

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. https://www.informingscience.org/Publications/4308

Alemneh, C., Munye, T., Mohammed, N., Shasho, G., Gizaw, E., & Siraj, Z. (2025). Improving students' engagement level in laboratory activities: A case study of second-year biology department students in the 2025 E.C academic year. Science Journal of Education, *13*(4), 117–125. https://doi.org/10.11648/j.sjedu.20251304.11

Amat Yasin, A., Masri, R., Adnan, M., & Mohamed, F. (2021). Pembangunan model pedagogi STEM Matematik berasaskan nilai dan akhlak di sekolah rendah: Satu analisis keperluan. *Jurnal Pendidikan Sains Dan Matematik Malaysia, 11,* 40-49. https://doi.org/10.37134/jpsmm.vol11.sp.4.2021

Arnab, S., Petridis, P., Spatioti, A. G., Kazanidis, I., & Pange, J. (2022). A comparative study of the ADDIE instructional design model in distance education. *Information, 13*(9), Article 402. https://doi.org/10.3390/INFO13090402

Ateeq, M., Habib, H., Umer, A., & Ul, M. (2014). C++ or Python? Which one to begin with: A learner's perspective. In *2014 International Conference on Teaching and Learning in Computing and Engineering (LaTiCE)* (pp. 64–69). IEEE 64–69. https://doi.org/10.1109/LaTiCE.2014.20

Bai, H., Wang, X., & Zhao, L. (2021). Effects of the problem-oriented learning model on middle school students' computational thinking skills in a Python course. *Frontiers in Psychology, 12*, Article 771221., 1–14. https://doi.org/10.3389/fpsyg.2021.771221

Barbosa, E., Lima, Y., Lyra, A., Salazar, H., Argôlo, M., de Almeida, M. A., Moreira, J., Babashahi, L., Barbosa, C. E., & de Souza, J. M. M. (2024). AI in the workplace: A systematic review of skill transformation in the industry. *Administrative Sciences, 14*(6), Article 127. https://doi.org/10.3390/admsci14060127

Bell, T. (2019). *CS Unplugged — How Is It Used , and Does It Work ?* (Issue 1).

Bell, T., & Vahrenhold, J. (2018). *CS unplugged: How is it used, and does it work?* In H.-J. Böckenhauer, D. Komm, & W. Unger (Eds.), *Adventures between lower bounds and higher altitudes* (pp. 497–521). Springer. https://doi.org/10.1007/978-3-319-98355-4_29

Brennan, K., & Resnick, M. (2012). *New frameworks for studying and assessing the development of computational thinking*. Paper presented at the Annual Meeting of the American Educational Research Association (AERA 2012), Vancouver, Canada.

Cazalas, J. M., Roberson, C., Furqan, Z., & Cazalas, J. (2024). From degree to developer: The creation and evolution of a CS course designed to bridge the academia-industry gap. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (pp. 186–192). ACM. https://doi.org/10.1145/3626252.3630860

Chen, Y., Xiao, S., Song, Y., Li, Z., Sun, L., & Chen, L. (2025). MindScratch: A visual programming support tool for classroom learning based on multimodal generative AI. *International Journal of Human-Computer Interaction, 41*(21), 13650–13668. https://doi.org/10.1080/10447318.2025.2475991

Csapó, G. (2019). Placing event-action-based visual programming in the process of Computer Science education. *Acta Polytechnica Hungarica*, *16*(2), 35–57. https://doi.org/10.12700/aph.16.2.2019.2.3

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

Dasuki, S., & Quaye, A. (2016). Undergraduate students' failure in programming courses in institutions of higher education in developing countries: A Nigerian perspective. *The Electronic Journal of Information Systems in Developing Countries*, *76*(1), 1–18. https://doi.org/10.1002/j.1681-4835.2016.tb00559.x

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

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

Feaster, Y., Segars, L., Wahba, S. K., & Hallstrom, J. O. (2011). Teaching CS unplugged in the high school (with limited success). In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education (ITiCSE '11)* (pp. 248–252). Association for Computing Machinery. https://doi.org/10.1145/1999747.1999817

Figueiredo, J. A. Q. (2017). How to improve computational thinking: A case study. *Education in The Knowledge Society*, *18*(4), 35–51. https://doi.org/10.14201/eks20171843551

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

Gonzalez, F., Lopez, C., & Castro, C. (2018). Development of computational thinking in high school students: A case study in Chile. In *2018 37th International Conference of the Chilean Computer Science Society (SCCC)* (pp. 1–8). IEEE. https://doi.org/10.1109/SCCC.2018.8705239

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher, 42*(1), 38–43. https://doi.org/10.3102/0013189X12463051

Hair, J. F., Black, W. C., Babin, B. J., & Anderson, R. E. (2019). *Multivariate data analysis (8th ed.)*. Cengage Learning.

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

Hendrix, R., & Weeks, M. (2018). First programming language for high school students. In *Society for Information Technology & Teacher Education International Conference Proceedings 2018* (pp. 1896–1903). Association for the Advancement of Computing in Education (AACE). https://www.learntechlib.org/p/182788/

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

Kadam, A. A., & Oyelere, S. S. (2026). *View of Selecting Suitable Programming Languages for Beginner-Level Instruction. International Journal of Technology in Education and Science, 10*(1), 133-161. https://doi.org/10.46328/ijtes.5061

Khangate, K., Rahatwal, A. K., & Kanade, P. V. (2024). The crucial role of programming languages in today's world. *International Journal of Advance Research, Ideas and Innovations in Technology*, *10*(1), 6–9. https://s3.ap-southeast-1.wasabisys.com/ijmanuscripts/manuscripts/v10i1/V10I1-1141.pdf

Khoirom, S., Sonia, M., Laikhuram, B., Laishram, J., & Singh, T. D. (2020). *Comparative Analysis of Python and Java for Beginners*. 4384–4407. https://www.irjet.net/archives/V7/i8/IRJET-V7I8755.pdf

Khoirom, S., Sonia, M., Laikhuram, B., Laishram, J., & Singh, T. D. (2020). Comparative analysis of Python and Java for beginners. *International Research Journal of Engineering and Technology, 7*(8), 4384–4407. https://www.irjet.net/archives/V7/i8/IRJET-V7I8755.pdf

Koorsse, M. (2012). *An evaluation of programming assistance tools to support the learning of IT programming: A case study in South African secondary schools* (Doctoral thesis, Nelson Mandela Metropolitan University, South Africa). https://dspacecris-nc1.mandela.ac.za/entities/thesis/319c6493-d3b2-4d27-b0fc-c07be35c896d

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

Laugwitz, B., Held, T., & Schrepp, M. (2008). Construction and evaluation of a user experience questionnaire. In A. Holzinger (Ed.), *HCI and usability for education and work* (Vol. 5298, pp. 63–76). Springer. https://doi.org/10.1007/978-3-540-89350-9_6

Lee, H. Y., Wu, T. T., Lin, C. J., Wang, W. S., & Huang, Y. M. (2023). Integrating computational thinking into scaffolding learning: An innovative approach to enhance science, technology, engineering, and mathematics hands-on learning. *Journal of Educational Computing Research, 62*(2), xx–xx. https://doi.org/10.1177/07356331231211916

Ling, H. C., Hsiao, K. L., & Hsu, W. C. (2021). Can students' computer programming learning motivation and effectiveness be

enhanced by learning Python language? A multi-group analysis. *Frontiers in Psychology, 11*, Article 600814. https://doi.org/10.3389/fpsyg.2020.600814

Liu, Z., Gearty, Z., Richard, E., Orrill, C. H., Kayumova, S., & Balasubramanian, R. (2024). Bringing computational thinking into classrooms: A systematic review on supporting teachers in integrating computational thinking into K–12 classrooms. *International Journal of STEM Education, 11*, Article 51. https://doi.org/10.1186/S40594-024-00510-6

Looi, C.-K. M., How, M.-L., Longkai, W., 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

Lye, S. Y., Hwee, J., & Koh, L. (2014). Computers in Human Behavior Review on teaching and learning of computational thinking through programming : What is next for K-12 ? *Computers in Human Behavior*, *41*, 51–61. https://doi.org/10.1016/j.chb.2014.09.012

Mäkelä, E., & Stephany, F. (2024). *Complement or substitute? How AI increases the demand for human skills* [Preprint]. arXiv. http://arxiv.org/abs/2412.19754

Mallillin, L. L. D. (2022). Teaching and learning intervention in the educational setting: adapting the teacher theory model. *International Journal of Educational Innovation and Research*, *1*(2), 99–121. https://doi.org/10.31949/ijeir.v1i2.2493

Mhashi, M. M., & Alakeel, A. (2013). Difficulties facing students in learning computer programming skills at Tabuk University. In *Recent advances in modern educational technologies* (pp. 15–24). WSEAS Press. https://www.wseas.us/e-library/conferences/2013/Morioka/EDU/EDU-01.pdf

Mohamad Ali, A. Z. (2019). *Coding di sekolah panduan guru dan pelajar*. Penerbit Universiti Pendidikan Sultan Idris.

Ateeq, M., Habib, H., Umer, A., & Rehman, M. U. (2014). C++ or Python? Which one to begin with: A learner's perspective. In *Proceedings of the 2014 International Conference on Teaching and Learning in Computing and Engineering (LaTiCE 2014)* (pp. 64–69). IEEE. https://doi.org/10.1109/LaTiCE.2014.20

Nunnally, J. C. (1978). *Psychometric theory (2nd ed.)*. McGraw-Hill.

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

Palts, T., & Pedaste, M. (2020). A model for developing computational thinking skills. *Informatics in Education*, *19*(1), 113–128. https://doi.org/10.15388/infedu.2020.06

Qian, Y., & Lehman, J. (2017). Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education, 18*(1), Article 1, 1–24. https://doi.org/10.1145/3077618

Qobilovna, A. M. (2023). Program for the development of factors of communicative competence in students. *International Journal of Pedagogics, 3*(11), 131–137. https://doi.org/10.37547/ijp/Volume03Issue11-25

Ramos, H. C. P., Caro, O. C., Bardales, E. S., Huatangari, Q. L., Trigoso, J. A. C., Guevara, J. L. M., & Santos, R. C. (2025). Artificial intelligence skills and their impact on the employability of university graduates. *Frontiers in Artificial Intelligence*, *8*, 1629320. https://doi.org/10.3389/FRAI.2025.1629320/BIBTEX

Rodriguez, B., Kennicutt, S., Rader, C., & Camp, T. (2017). Assessing computational thinking in CS unplugged activities. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 501–506). Association for Computing Machinery. https://doi.org/10.1145/3017680.3017779

Rosli, N. M., Mohd Matore, M. E. @ E., & Husnin, H. (2024). Synergizing the integration of coding and programming competencies through quadruple helix model for Malaysian vocational education. In *International Journal of Academic Research in Progressive Education and Development, 13*(4), 3739-3747. https://doi.org/10.6007/ijarped/v13-i4/24377

Setyorini, T. (2020). Improving student learning outcomes using image media on learning science materials about sense tools in SDN 2 Tuksongo students. *Social, Humanities, and Educational Studies (SHEs): Conference Series, 3*(3), 1336–1341. https://doi.org/10.20961/shes.v3i3.46667

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

Singh, S. (2022). Identifying learning challenges faced by novice/beginner computer programming students: An action research approach. In *Proceedings of the 6th Software Engineering Education Workshop*. CEUR Workshop Proceedings. http://ceur-ws.org

Sivaraman, A., Zhang, T., Van den Broeck, G., & Kim, M. (2019). Active inductive logic programming for code search. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)* (pp. 292–303). IEEE. https://doi.org/10.1109/ICSE.2019.00044

Taub, R., Armoni, M., & Ben-Ari, M. (2012). CS unplugged and middle-school students' views, attitudes, and intentions regarding CS. *ACM Transactions on Computing Education*, *12*(2), 1–29. https://doi.org/10.1145/2160547.2160551

Thies, R., & Vahrenhold, J. (2013). On plugging "unplugged" into CS classes. In *Proceedings of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE '13)* (pp. 365–370). Association for Computing Machinery. https://doi.org/10.1145/2445196.2445303

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

Vidal-Silva, C., Barriga, N. A., Ortega-Cordero, F., González-López, J., Jiménez-Quintana, C., Pezoa-Fuentes, C., & Veas-González, I. (2022). Developing computing competencies without restrictions. *IEEE Access, 10*, 106568–106580. https://doi.org/10.1109/ACCESS.2022.3211973

Wang, C. Y., Gao, B. L., & Chen, S. J. (2024). The effects of metacognitive scaffolding of project-based learning environments on students' metacognitive ability and computational thinking. *Education and Information Technologies*, *29*(5), 5485–5508. https://doi.org/10.1007/s10639-023-12022-x

Weeda, R., Smetsers, S., & Barendsen, E. (2024). Unraveling novices' code composition difficulties. *Computer Science Education*, *34*(3), 414–441. https://doi.org/10.1080/08993408.2023.2169067

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *366*(1881), 3717–3725. https://doi.org/10.1098/rsta.2008.0118

Wu, T. T., Asmara, A., Huang, Y. M., & Permata Hapsari, I. (2024). Identification of problem-solving techniques in computational thinking studies: Systematic literature review. *SAGE Open*, *14*(2). https://doi.org/10.1177/21582440241249897

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. In *Proceedings of the International Conference on Teacher Training and Education (ICTTE)* (Vol. 1, pp. 962–966). https://docslib.org/doc/5938712/robomind-utilization-to-improve-student-motivation-and-concept-in-learning-programming

Yuen, T. T., Reyes, M., & Zhang, Y. (2019). Introducing computer science to high school students through logic programming. *Theory and Practice of Logic Programming, 19*(2), 248–264. https://doi.org/10.1017/S1471068418000431

Zuriana, A. B., Ashanira, M. D., & Fouad, A. S. (2025). Designing visually engaging online learning interfaces for higher education students. *Journal of Advanced Research in Computing and Applications, 39*(1), 111–124. https://doi.org/10.37934/arca.39.1.111124