2026 ASEE ANNUAL CONFERENCE June 21–24, 2026 | Charlotte Convention Center, NC
Host Sponsor UNC Charlotte
ASEE AMERICAN SOCIETY FOR ENGINEERING EDUCATION
Paper ID #52566

Understanding Programming Misconceptions: A Study of First-Year Intermediate Students in Pakistan

Dua e Sehar Butt, Islamabad Model College for Girls Korang Town Islamabad

My name is Dua, and I'm a first-year student (Grade 11) at Islamabad Model College for Girls Korang Town (IMCG), Islamabad, Pakistan. Having completed my matriculation recently, I've always been deeply interested in computer science—not just as a subject, but as a way of thinking. What fascinates me most is how differently every student's brain works, and how one teaching approach can't possibly fit everyone. While sitting in my programming class, I noticed that many of my classmates hold misconceptions about what programming really is. Some believe it requires an extraordinary talent or deep mathematical genius, while others see it as something boring or impossibly hard. That observation sparked a thought in me: what if the real problem isn't programming itself, but how it's being understood and taught?

Sabah-ud-din Waqar, University of Oklahoma

Mr. Sabah ud din Waqar is a PhD student in Engineering Education at the University of Oklahoma. He holds an MS in Software Engineering and a bachelor's degree in Computer Science. His research focuses on the role of emotions in engineering education, particularly how emotional factors affect learning outcomes and student experiences. With over nine years of teaching experience in Pakistan, he has developed a strong interest in emotions in engineering learning and their impact on student engagement. He is also a certified emotional intelligence practitioner, working to enhance educational methodologies in engineering. Mr. Waqar is passionate about fostering a positive learning environment for engineering students.

Dr. Javeed Kittur, The University of Oklahoma

Javeed Kittur is an Assistant Professor in the Engineering Pathways program at the University of Oklahoma and serves as the Co-Director of Research in the Honors College, bringing a global academic background that includes a Ph.D. in Engineering Education Systems and Design from Arizona State University. Prior to joining OU, he served as an Assistant Professor in Electrical and Electronics Engineering at KLE Technological University, India, and is a certified IUCEE International Engineering Educator and Associate Editor for the Journal of Engineering Education Transformations. Since 2022, he has mentored over 40 undergraduate engineering students, many of whom have published in leading conferences and journals, demonstrating his strong commitment to impactful student mentorship. His excellence in teaching and mentoring has been recognized with multiple prestigious awards, including the 2026 Provost's Distinguished First-Year Student Mentoring Program - Outstanding Mentor Award, 2025 Nancy L. Mergler Faculty Mentor Award, the 2025 Brandon H. Griffith Outstanding Faculty Member Award, and the 2024 SGA Outstanding Faculty Award. Dr. Kittur's research advances engineering education through innovative, data-driven approaches, with a focus on generative AI integration, learning analytics, student retention, broadening participation, and holistic faculty development across cognitive, affective, and psychomotor domains.

Understanding Programming Misconceptions: A Study of First-Year Intermediate (Grade 11) Students in Pakistan

Abstract

Learning computer programming is often one of the most challenging experiences for students in the first year of intermediate education in Pakistan, which is roughly equivalent to Grade 11 or the first year of upper-secondary college. Many students enter programming courses with limited exposure to computational thinking, which can lead to the development of programming misconceptions, that is, persistent misunderstandings about how programming concepts and code actually work. These misconceptions can hinder students' understanding of fundamental concepts such as variables, loops, conditionals, functions, and algorithms. Although programming misconceptions have been widely studied internationally, little research has examined how they emerge in the Pakistani education system, where differences in teaching methods, access to technology, and prior coding exposure may create distinct learning challenges. Understanding these context-specific misconceptions is important for designing effective and locally relevant teaching strategies.

This qualitative study explores three questions: (1) What types of programming misconceptions do first-year intermediate computer science students in Pakistan hold? (2) What factors contribute to the emergence and persistence of these misconceptions? (3) How do students' perceptions of task cost and value influence their engagement and learning in programming? Data were collected through semi-structured, in-person interviews with participants, each lasting approximately 30–45 minutes, providing in-depth qualitative insights into student learning. The study investigates not only the types of misconceptions students hold but also underlying factors such as reliance on rote memorisation, difficulties with logic-based thinking, and limited opportunities for hands-on coding practice during earlier schooling. The participants were intermediate computer science students studying algorithms, flowcharts, and Python in a public college in Pakistan. Findings reveal perceptions of programming as complex and cognitively demanding, persistent conceptual misunderstandings, emotional frustration, avoidance behaviours, and heavy reliance on external aids. Students also suggested strategies to reduce cognitive load and enhance learning value, including visual tools, gamified learning, and earlier exposure to programming. These insights highlight the importance of concept-focused teaching, teacher training, and interactive tools to improve programming education outcomes in Pakistan.

Introduction

Computer programming is a core component of Science, Technology, Engineering, and Mathematics (STEM) education, as it develops students' abilities in problem-solving, algorithmic thinking, and computational reasoning [1,2]. As programming skills are increasingly required across academic and professional domains, introductory programming courses play a critical role in shaping students' foundational understanding and long-term engagement with computing [1,2].

Despite its importance, learning to program remains challenging for novice learners. Research consistently shows that students struggle with fundamental programming concepts such as variables, conditionals, loops, functions, and algorithms [3,4]. These difficulties often lead to the formation of misconceptions, meaning persistent misunderstandings or incorrect mental models about how programming concepts and code work [5,6]. Misconceptions in

programming can result in repeated errors, surface-level learning, reduced confidence, and high attrition rates in introductory programming courses [6,7].

A substantial body of international research has examined programming misconceptions among novice learners. Studies have documented common misunderstandings related to variable assignment, control flow, iteration, and program execution tracing [3,5,6]. Research has shown that many novice programmers struggle not only with syntax, but also with understanding how programs execute step by step [5,6,8]. Similarly, Sorva [9] emphasised the role of flawed "notional machines" in shaping incorrect reasoning about program behaviour. These findings suggest that misconceptions are closely tied to how students conceptualise program execution rather than simply to a lack of practice.

In response, researchers have explored instructional strategies aimed at reducing misconceptions, including visual programming environments, explicit instruction on program tracing, and concept-focused pedagogies [9,10,11]. While these approaches have shown promise, their effectiveness is often influenced by students' prior experiences, learning contexts, and motivational beliefs [10]. As a result, misconceptions cannot be fully understood without considering the broader educational and cultural settings in which learning occurs.

In Pakistan, first-year intermediate students are students in the first year of upper-secondary college education, roughly equivalent to Grade 11. For many students enrolled in computer science at this level, this is their first formal exposure to programming. First-year intermediate computer science students in Pakistan are typically introduced to programming with little or no prior exposure to coding or computational thinking [12]. Instruction at this level is often constrained by limited infrastructure, resource shortages, and competing instructional demands, resulting in fewer opportunities for hands-on practice and conceptual exploration [13,14]. While international research has established that novice programmers experience persistent misconceptions related to fundamental programming concepts [6], there is a scarcity of empirical studies examining such misconceptions specifically within the Pakistani intermediate education context.

Moreover, existing studies have largely focused on identifying what misconceptions students hold, with less attention to why these misconceptions persist. Recent work in computing education highlights the importance of motivational and affective factors in learning programming, particularly students' perceptions of effort, difficulty, and task value [15,16]. When students perceive programming tasks as excessively demanding, time-consuming, or confusing, the perceived cost of learning may outweigh the perceived value, leading to disengagement and shallow learning strategies [16,17].

Although programming misconceptions have been widely studied at the university level, there is limited research examining these misconceptions among intermediate-level students, especially in Pakistan [6]. This gap is important because intermediate students often experience their first formal exposure to programming at this stage, and their misconceptions may strongly influence future learning and persistence in computing. Moreover, the Pakistani context presents distinct challenges such as limited infrastructure, constrained instructional time, and restricted access to technology, which may contribute to the development and persistence of misconceptions [13,18].

To address this gap, the present study investigates programming misconceptions among first-year intermediate computer science students in Pakistan. Using Cost–Value Theory as a

guiding framework, this study explores how students' perceptions of task cost and value shape their engagement and learning in programming. Data were collected through classroom observations and semi-structured, in-person interviews to develop a context-sensitive understanding of how misconceptions emerge and persist in this setting.

Research Questions

1. What types of misconceptions do first-year intermediate computer science students in Pakistan hold regarding programming concepts such as variables, loops, conditionals, functions, and algorithms?
2. What factors contribute to the emergence and persistence of these misconceptions, including prior learning experiences, instructional methods, and access to technology?
3. How do students' perceptions of task cost and value, as described by Cost–Value Theory, influence their misconceptions, engagement, and learning in programming?

Theoretical Perspective

This study is grounded in Cost–Value Theory, which extends the Expectancy–Value framework developed by Eccles and Wigfield [16]. The theory argues that students' motivation, engagement, persistence, and achievement are influenced not only by how much they value a task but also by how much effort, time, and emotional strain they believe the task requires. In this framework, value includes the perceived usefulness, importance, interest, and enjoyment of a task, while cost includes perceived effort, time demands, anxiety, and emotional exhaustion associated with task completion [16,17].

Applied to programming education, this theory suggests that when students perceive programming as valuable because of its relevance to future careers or its problem-solving nature, they are more likely to persist despite challenges. However, if they perceive programming as excessively demanding, time-consuming, or confusing, the perceived cost may outweigh the perceived value. This imbalance can reduce motivation and engagement, and it can also contribute to the formation and persistence of misconceptions [16,17].

In the context of Pakistani intermediate-level education, where exposure to programming is limited and instruction often emphasises rote memorisation rather than conceptual understanding, students may experience higher perceived costs because of cognitive overload and frustration [18]. This theoretical lens enables a deeper understanding of how misconceptions, such as confusing assignment with equality or misinterpreting loop logic, emerge and persist when students' motivational and cognitive resources are imbalanced. Thus, Cost–Value Theory provides a useful interpretive framework for exploring the interplay between students' beliefs, affective experiences, and conceptual misunderstandings in programming.

Methods

This study employed a qualitative research design to explore the complex and context-dependent nature of programming misconceptions among intermediate-level students. Qualitative inquiry is suitable for capturing students' perceptions, experiences, and emotions in a detailed and context-specific manner, which are not easily measurable through quantitative methods [19,20].

This study was informed by a phenomenological orientation, aiming to understand students' lived experiences with programming misconceptions [21,22]. Through this approach, the study

seeks to reveal how students interpret programming tasks, how they feel during learning, and how these experiences contribute to misconceptions. Semi-structured interviews were used as the primary data collection tool, enabling flexibility to probe more deeply into participants' thoughts while maintaining consistency across interviews [23]. Interviews were supplemented with classroom observations to triangulate findings and strengthen the credibility of the results [24].

Participants

The participants were five Grade 11 students enrolled in the Intermediate in Computer Science (ICS) programme at a public college in Pakistan. Purposeful sampling was used to ensure variation in academic performance and prior exposure to programming.

1. *Initial Contact*: The researcher visited the ICS classroom and provided a brief explanation of the study.
2. *Invitation*: A call for volunteers was made, and interested students completed a short screening form capturing academic standing, prior programming experience, and willingness to participate.
3. *Selection*: Among the students who volunteered, five participants were selected to ensure diversity in academic performance and programming background.
4. *Consent*: Participation was voluntary, and written consent was obtained from both students and their guardians.

This small, focused sample enabled in-depth exploration of each student's conceptual understanding and emotional experiences during programming tasks. Pseudonyms were assigned to protect participants' identities. Table 1 summarises the demographic and academic characteristics of the participants. It shows that all participants were female, aged 16–17, and had varying levels of prior programming experience. The table also indicates the programming topics studied by each participant, providing context for interpreting their misconceptions.

Table 1. Demographic and academic characteristics of participants

Participant	Gender Identity	Age	Academic Standing	Prior Programming Experience	Programming Topics Studied
S1	Female	16	Average	None	Algorithms, Flowcharts, Python Basics
S2	Female	17	Above Average	Python, C, Algorithms	Python Conditionals, Loops, Python Basics
S3	Female	16	Average	Basics of C and HTML, Algorithms	Algorithms, Variables, Loops, Python Basics
S4	Female	17	High	HTML basics, Algorithms	Functions, Algorithms, Loops, Python Basics
S5	Female	16	Below Average	Algorithms, Flowcharts	Flowcharts, Conditional Logic, Python Basics

Data Collection

Semi-structured interviews were used as the primary data collection method, as they allow flexibility to explore participants' perspectives while maintaining consistency across interviews [23]. An interview protocol was developed based on the research questions and Cost–Value Theory. A pilot interview was conducted prior to data collection to assess clarity

and relevance, and minor refinements were made to the interview questions. All interviews were conducted in person and audio-recorded. Field notes were taken during and after each interview to capture non-verbal cues and contextual details. Interviews were transcribed verbatim and translated into English for analysis. To preserve accuracy and meaning, translations were reviewed alongside the original recordings, and ambiguous segments were clarified by revisiting the audio.

Data Analysis

Data were analysed using thematic analysis following the procedures outlined by Braun and Clarke [24]. The analysis proceeded through the following stages:

- *Familiarisation:* Transcripts were read and re-read to gain an overall understanding of participants' experiences and to note preliminary ideas related to programming misconceptions, emotional responses, and perceived cost and value.
- *Coding:* Initial codes were generated through line-by-line analysis, capturing meaningful features of the data relevant to misconceptions, cognitive effort, emotional reactions, and value perceptions.
- *Theme Development:* Related codes were grouped into broader candidate themes, such as cognitive overload, fear of syntax errors, and low perceived value.
- *Reviewing Themes:* Themes were reviewed for internal coherence and consistency across the dataset. Codes that overlapped conceptually were merged, and themes were refined to ensure clear boundaries.
- *Defining and Naming Themes:* Final themes were clearly defined, named, and interpreted in relation to Cost–Value Theory and the research questions.

The analysis involved familiarisation with the data, generation of initial codes, development of themes, and refinement of thematic interpretations. NVivo 12 software was used to organise and manage the coding process. The coding process followed three phases (see Table 2).

Table 2. Coding phases used in thematic analysis

Phase	Details	Examples
1	Initial open coding (line-by-line)	Cognitive overload, syntax focus, loop confusion, fear of errors, external help, motivation, avoidance
2	Code refinement and reduction	Misconceptions about variables, misconceptions about loops, emotional frustration, lack of autonomy, suggestions for improvement
3	Theme development	(1) Perceived complexity and cognitive overload; (2) Conceptual misunderstandings and surface-level learning; (3) Emotional frustration, avoidance, and motivation; (4) Reliance on external aids and lack of autonomy; (5) Suggestions for reducing cost and increasing value

The initial coding phase resulted in 42 codes. After removing redundancies and merging conceptually similar codes, 27 major codes remained. These were then grouped into five overarching themes based on conceptual similarity and relevance to the research questions. While themes were derived inductively from the data, interpretation was informed by the Cost–Value theoretical lens [21].

Findings

To answer the research questions, data from semi-structured interviews were analysed using thematic analysis. First, the transcripts were coded line by line to capture patterns in students'

experiences, beliefs, and emotional responses towards programming. Initial codes included "difficulty perception", "syntax focus", "loop misunderstanding", "external help", "emotional frustration", "avoidance", "future relevance", and "suggested improvements". These codes were then grouped into broader themes that represent the underlying structure of students' experiences. The coding process and theme development are summarised in Table 2. The themes presented below are supported by multiple participants' accounts and are interpreted through Cost–Value Theory to explain how perceived costs and values shape misconceptions and engagement.

Theme 1: Perceived Complexity and Cognitive Overload

Across all participants, programming was perceived as inherently complex and intimidating, particularly when students first encountered it. Students frequently associated programming with long code, confusing statements, and overwhelming screen content. For example, S1 described her early impression as "very complex coding… it seems very confusing", while S2 stated, "By the name 'programming' itself, I feel it's very difficult." S3 similarly described programming as "complicated", reflecting early cognitive overload.

This theme aligns directly with Research Question 2, which asks about factors contributing to misconceptions. Students' initial exposure to programming in a syntax-heavy environment increased cognitive load, leading to early disengagement. In Cost–Value terms, the perceived cost, in the form of mental effort and confusion, becomes high before students even develop meaningful understanding. As a result, students may avoid deeper engagement, leading to surface-level learning and misconceptions. Importantly, the shift from initial optimism to confusion, as noted by S4 ("I used to think it was easy… now I think it's not easy at all"), shows how early cognitive overload reshapes students' beliefs about their capability and the task's value.

Theme 2: Conceptual Misunderstandings and Surface-Level Learning

Participants demonstrated consistent misconceptions about fundamental programming concepts, indicating a reliance on surface-level understanding. Variables were often described as commands rather than data containers. For instance, S1 defined a variable as "what you give to them, like what to do in it", while S3 described it as "something used instead of a number", suggesting confusion between variables and operations.

Similarly, loops were misunderstood as random repetition or a mechanism for fixing errors. S5 stated, "loops work when you don't get the required answer", while S2 explained, "one thing keeps repeating again and again", without reference to loop conditions or control structures. Syntax errors were perceived as the primary indicator of programming difficulty. S4 noted, "If there is a syntax error, I rewrite the program", indicating that correctness was prioritised over conceptual understanding.

Through the Cost–Value lens, these misconceptions increase perceived cost by encouraging time-consuming trial-and-error strategies. At the same time, perceived value decreases as programming is viewed as memorisation rather than logical reasoning. This theme directly addresses Research Question 1 by identifying the types of misconceptions students hold and showing how these emerge under high cognitive load and limited conceptual scaffolding.

Theme 3: Emotional Frustration, Avoidance, and Motivation

Emotional responses were a prominent aspect of students' learning experiences. Four of the five participants reported feelings of anger, stress, or avoidance when encountering programming difficulties. S4 shared, "I get so angry… if there is a syntax error, I rewrite the program", while S3 stated, "There is no motivation at all." S2 admitted, "I usually avoid such questions… I just skip it."

In contrast, S5 demonstrated resilience and future-oriented motivation, connecting programming to long-term academic and career goals. She explained that understanding programming basics was important for university-level study and future opportunities in information technology. This suggests that when perceived value is high, students are more willing to tolerate effort and persist despite challenges.

This theme illustrates how emotional cost reinforces misconceptions. Repeated failure increases frustration and avoidance, while perceived relevance increases motivation and persistence. The findings address Research Question 3 by highlighting how cost and value perceptions shape engagement.

Theme 4: Reliance on External Aids and Lack of Autonomy

All participants reported heavy reliance on external supports such as teachers, classmates, online resources, and AI tools. None described independent debugging or reflective problem-solving strategies. S3 stated, "I ask ChatGPT to correct it", while S1 reported using Google or teacher assistance. S2 acknowledged, "I never programmed without external help."

This reliance suggests low self-efficacy and high perceived task cost. From a Cost–Value perspective, seeking external help functions as a cost-reduction strategy. While such tools can support learning, excessive dependence may hinder conceptual development and autonomy. This theme contributes to Research Question 2 by identifying contextual and instructional factors that sustain misconceptions.

Theme 5: Suggestions for Reducing Cognitive Cost and Enhancing Learning Value

Despite their difficulties, students proposed strategies to improve programming instruction. These suggestions focused on reducing cognitive load and increasing perceived value. S1 recommended video-based and gamified learning approaches. S3 suggested introducing programming earlier in schooling. S5 emphasised practical, hands-on learning with simplified syntax, while S2 highlighted the need for greater access to computers and more time for practice.

These recommendations indicate students' awareness of the cost–value imbalance in their learning experiences. The suggestions support instructional approaches that are visual, contextual, and practice-oriented, directly addressing Research Question 3.

Discussion

This study explored intermediate-level computer science students' programming misconceptions, emotional responses, and motivational beliefs within the Pakistani educational context. Interpreted through Cost–Value Theory, the findings show that students' learning

difficulties are shaped not only by conceptual gaps but also by an imbalance between high perceived cognitive and emotional costs and low perceived instructional value [16,17]. The discussion below situates these findings within existing literature and highlights how contextual, cognitive, and motivational factors jointly influence programming learning.

Students' strong perception of programming as complex and intimidating reflects a well-documented challenge among novice programmers. Prior research shows that beginners often experience cognitive overload when exposed early to syntax-heavy instruction without sufficient conceptual scaffolding [3,6]. In this study, students associated programming with long code, confusing symbols, and fear of errors, indicating that the initial learning environment amplified mental effort before conceptual understanding could develop. Similar findings have been reported in secondary and introductory programming contexts, where excessive emphasis on syntax leads to early disengagement and shallow learning strategies [4,19].

From a Cost–Value Theory perspective, this perceived complexity represents a high cognitive cost that students must bear before recognising any meaningful value in programming tasks [16,17]. When effort outweighs perceived benefit, learners are more likely to disengage or adopt avoidance strategies. This pattern is particularly pronounced in examination-oriented systems such as Pakistan's, where correctness is prioritised over exploration and sense-making [13,18]. As a result, students' early optimism may be replaced by frustration, reshaping their beliefs about both their competence and the usefulness of programming.

The findings reveal persistent misconceptions related to variables, loops, and program logic. Students often interpreted variables as commands or placeholders rather than abstract data containers, and loops as mechanisms for correcting errors rather than controlled repetition structures. These misconceptions align closely with earlier studies documenting novice learners' difficulty in forming accurate mental models of programming constructs [3,6,19]. Research consistently shows that without explicit attention to conceptual reasoning, students rely on memorisation and trial-and-error approaches [4,5].

Within the Cost–Value framework, such surface-level learning increases perceived cost by forcing students into inefficient debugging practices while simultaneously reducing perceived value, as programming is experienced as rote and meaningless [16]. When students fail to see logical coherence or relevance, their investment in learning declines. This finding supports previous work indicating that conceptual misunderstanding is both a cognitive and motivational issue, not merely a technical one [6,20].

Emotional responses played a central role in shaping students' engagement. Feelings of frustration, anger, and avoidance were common when students encountered errors or unfamiliar structures. Prior studies have shown that negative emotions such as anxiety and frustration are strongly associated with programming failure and dropout, especially among novices [6,7]. In this study, repeated failure increased emotional cost, reinforcing avoidance behaviours such as skipping questions or rewriting code without reflection.

However, students who connected programming to future academic or career goals demonstrated greater persistence. This supports Cost–Value Theory's assertion that high task value can buffer the negative effects of high effort and emotional strain [16,17]. When learners perceive programming as useful for higher education or employment, they are more willing to

tolerate difficulty. Similar findings have been reported in computing education research, where relevance and career alignment increase motivation despite challenging content [1,10].

All participants reported heavy reliance on teachers, peers, online tutorials, or AI-based tools for debugging and understanding code. While external support can facilitate learning, excessive dependence often reflects low self-efficacy and high perceived task cost [6,7]. Prior research cautions that overreliance on external solutions may hinder the development of independent problem-solving skills and conceptual reasoning [4,9].

From a Cost–Value perspective, seeking external help functions as a cost-reduction strategy, allowing students to complete tasks with less mental effort [17]. In contexts where instructional time, resources, and individualised support are limited, such strategies may become adaptive but may also sustain misconceptions. This pattern is consistent with findings from Pakistani secondary education, where limited access to hands-on practice and supportive learning environments constrains autonomous engagement with programming [13,18].

Students' instructional suggestions offer important insights into how programming education can be improved. Their emphasis on visual explanations, gamified learning, early exposure, and hands-on practice aligns with research advocating concept-first and activity-based approaches to computing education [1,9,10]. Tools such as program visualisation systems, block-based environments, and unplugged activities have been shown to reduce cognitive load while increasing engagement and conceptual clarity [9,11,14].

These recommendations reflect students' intuitive understanding of the cost–value imbalance in their learning experiences. When instruction reduces unnecessary cognitive burden and emphasises relevance, perceived value increases and misconceptions are less likely to persist. This finding reinforces the argument that effective programming instruction must address motivational and emotional dimensions alongside technical content [16,25]. Overall, the findings suggest that reducing cognitive load and increasing perceived value can improve programming learning by supporting both conceptual understanding and emotional resilience.

Limitations

One major limitation of this study is the gender imbalance in the sample. All participants were female, which limits the transferability of the findings to the broader student population in Pakistan. In addition, the sample size was small and drawn from a single course context, further restricting the transferability of the results. Future research should address these limitations by including male participants to capture possible gender-based differences in programming misconceptions and learning experiences. Studies with larger and more diverse samples across multiple institutions and instructional contexts are also needed. Furthermore, longitudinal research could provide deeper insight into how students' perceptions of cost and value evolve over time as their programming competence develops.

Conclusion

This study provides a qualitative understanding of how first-year intermediate computer science students in Pakistan experience the challenges of learning programming. The findings highlight that students struggle not only with technical misunderstandings but also with motivational barriers and emotional responses that hinder engagement. Through the lens of

Cost–Value Theory, these challenges can be understood as an imbalance between the high perceived cost of learning and the low perceived value of programming education.

The participants viewed programming as complex and error-prone, which made the learning process mentally demanding and emotionally stressful. Their misconceptions about fundamental concepts such as variables, loops, and conditionals further increased cognitive effort and reduced confidence. However, when students connected programming to their future goals or practical applications, they experienced greater motivation and persistence. This demonstrates that enhancing the perceived value of programming can help offset its cognitive and emotional costs.

To improve programming education in Pakistan, teaching strategies should emphasise conceptual understanding, practical engagement, and relevance to students' lives. Curriculum designers should consider introducing programming concepts at earlier educational stages and providing access to visual and interactive learning tools. Teacher training should also focus on motivational and psychological aspects of learning, not only technical instruction. By integrating these elements, programming education can become more meaningful, accessible, and rewarding for students.

Ultimately, this study underscores the need for culturally responsive and motivationally informed approaches to programming education. When students perceive programming as both valuable and attainable, they are more likely to persist, develop critical computational thinking skills, and prepare for meaningful participation in the digital world.

AI Disclosure: AI tools (for example, ChatGPT) were used for language editing and formatting. All content was reviewed and verified by the authors.

References

1. Z. Liu, Z. Gearty, E. Richard, C. H. Orrill, and S. Kayumova, "Bringing computational thinking into classrooms: A systematic review on supporting teachers in integrating computational thinking into K–12 classrooms," *International Journal of STEM Education*, vol. 11, Art. no. 51, 2024.
2. F. K. Cansu, "An overview of computational thinking," *International Journal of Computer Science Education in Schools*, vol. 3, no. 1, pp. 17–30, 2019.
3. C. S. Miller and A. Settle, "Learning to get literal: Investigating reference-point difficulties in novice programming," *ACM Transactions on Computing Education (TOCE)*, vol. 19, no. 3, pp. 1–17, 2019.
4. R. Weeda, S. Smetsers, and E. Barendsen, "Unravelling novices' code composition difficulties," *Computer Science Education*, vol. 34, no. 3, pp. 414–441, 2024.
5. L. Bidlake, E. Aubanel, and D. Voyer, "Investigating the progression of the mental models formed by programmers learning parallel programming," *ACM Transactions on Computing Education*, vol. 25, no. 1, pp. 1–31, 2025.
6. Y. Qian and J. Lehman, "Students' misconceptions and other difficulties in introductory programming: A literature review," *ACM Transactions on Computing Education (TOCE)*, vol. 18, no. 1, pp. 1–24, 2017.
7. J. Bennedsen and M. E. Caspersen, "Failure rates in introductory programming: 12 years later," *ACM Inroads*, vol. 10, no. 2, pp. 30–36, 2019.

8. R. Lister, E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, et al., "A multi-national study of reading and tracing skills in novice programmers," *ACM SIGCSE Bulletin*, vol. 36, no. 4, pp. 119–150, 2004.
9. J. Sorva, V. Karavirta, and L. Malmi, "A review of generic program visualisation systems for introductory programming education," *ACM Transactions on Computing Education (TOCE)*, vol. 13, no. 4, pp. 1–64, 2013.
10. D. M. Córdova-Esparza, J. A. Romero-González, K. E. Córdova-Esparza, J. Terven, and R. E. López-Martínez, "Active learning strategies in computer science education: A systematic review," *Multimodal Technologies and Interaction*, vol. 8, no. 6, Art. no. 50, 2024.
11. J. Moreno-León, G. Robles, and M. Román-González, "Dr. Scratch: Automatic analysis of Scratch projects to assess computational thinking," *International Journal of Educational Technology in Higher Education*, vol. 12, no. 1, pp. 1–19, 2015.
12. M. A. Malik and M. A. Akram, "A retrospective study for the selection of suitable programming language for the high schools in Pakistan," *International Journal of Computer Science and Information Security (IJCSIS)*, vol. 14, no. 9, pp. 1–9, 2016.
13. H. Khalid and M. Azeem, "Issues in the teaching of computer science at the secondary level in Pakistan," *International Journal of Humanities and Social Science*, vol. 2, no. 4, pp. 277–283, 2012.
14. S. Jehan and P. Akram, "Introducing computer science unplugged in Pakistan: A machine learning approach," *Education Sciences*, vol. 13, no. 9, Art. no. 892, 2023.
15. V. Garneli, M. N. Giannakos, and K. Chorianopoulos, "Computing education in K–12 schools: A review of the literature," *Educational Technology & Society*, vol. 18, no. 1, pp. 188–199, 2015.
16. J. S. Eccles and A. Wigfield, "Motivational beliefs, values, and goals," *Annual Review of Psychology*, vol. 53, pp. 109–132, 2002.
17. J. K. Flake, K. E. Barron, C. S. Hulleman, D. B. McCoach, and M. E. Welsh, "Measuring cost: The forgotten component of expectancy–value theory," *Contemporary Educational Psychology*, vol. 41, pp. 232–244, 2015.
18. S. Saeed and M. Khan, "Challenges in teaching computer science in public secondary schools: Pakistani context," *Journal of Educational Research and Social Sciences Review (JERSSR)*, vol. 2, no. 4, pp. 103–112, 2022.
19. J. W. Creswell and C. N. Poth, *Qualitative Inquiry and Research Design: Choosing Among Five Approaches*, 4th ed. Sage, 2018.
20. J. A. Maxwell, *Qualitative Research Design: An Interactive Approach*, 3rd ed. Sage, 2013.
21. C. Moustakas, *Phenomenological Research Methods*. Sage, 1994.
22. J. Kittur and S. Tuti, "Conducting qualitative research study: A step-by-step process," *Journal of Engineering Education Transformations*, vol. 28, 2024.
23. J. W. Creswell, *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*, 5th ed. Sage, 2018.
24. V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative Research in Psychology*, vol. 3, no. 2, pp. 77–101, 2006.
25. M. Chakraborty and F. M. Nafukho, "Strengthening student engagement: what do students want in online courses?" *European Journal of Training and Development*, vol. 38, no. 9, pp. 782–802, 2014.