Enhancing Inclusivity and Accessibility of Python Programming Tutorials for Civil Engineering Students

Yongmin Kim*, Samiran Das, and Jolly Atit Shah

School of Engineering, University of Glasgow, Singapore
Email: yongmin.kim@glasgow.ac.uk (Y.K.); samiran.das@glasgow.ac.uk (S.D.); jollyatit.shah@glasgow.ac.uk (J.A.S.)
*Corresponding author

***Abstract*—This study addresses the challenges and strategies for enhancing the inclusivity and accessibility of Python programming lectures for Civil Engineering (CVE) students at a university in Singapore. Despite their strong foundations in mathematics and physics, the diverse backgrounds and varied coding experiences of CVE students create barriers to effective learning in fast-paced, interactive coding sessions. This paper critically reflects on current teaching practices, analyzes student feedback, and proposes an action plan based on Universal Design for Learning principles and culturally responsive teaching frameworks. Findings indicate that providing pre-tutorial resources, multimodal teaching aids, and institutional support enhances engagement, reduces disparities, and improves inclusivity. Future iterations will include comprehensive evaluations to refine these strategies further.**

***Keywords*—inclusive learning, accessibility, Universal Design for Learning (UDL), interactive lectures, civil engineering**

I. Introduction

The integration of programming skills in engineering education has become increasingly critical, reflecting the growing reliance on computational methods in infrastructure projects. Python, in particular, offers versatile applications in data analysis, automation, and problem-solving—making it an indispensable skill for Civil Engineering (CVE) students. The Civil Engineering Skills module taught at a university in Singapore is designed to equip students with these competencies, enabling them to analyze and process engineering data efficiently. However, the diverse backgrounds of CVE students present challenges in delivering coding education inclusively and effectively.

CVE students often possess strong foundations in mathematics and physics, which support their logical problem-solving abilities. However, not all students begin the module with the same level of programming experience. This disparity is compounded by cognitive barriers such as learning disabilities, language proficiency challenges, and differences in educational backgrounds. Singapore's multicultural environment adds another layer of complexity, as students' familiarity with English—the medium of instruction—varies. As programming education relies heavily on both technical and linguistic comprehension, these factors can create significant disparities in learning outcomes.

The module currently employs tutorial-based lectures, a format praised for its interactivity and practical focus. While this approach engages students through live coding demonstrations and immediate feedback, it also demands rapid comprehension and real-time problem-solving, which can be overwhelming for some learners. Cognitive overload, accessibility barriers, and a lack of flexibility in learning styles have emerged as recurring issues in this teaching context. These challenges highlight the need for a more inclusive approach that accommodates diverse learning needs and styles.

Inclusive education frameworks such as Universal Design for Learning (UDL) provide valuable guidance for addressing these challenges. By emphasizing multiple means of representation, engagement, and expression, UDL offers a proactive approach to designing learning environments that cater to a wide range of abilities and preferences. Additionally, culturally responsive teaching practices recognize the importance of tailoring educational content to reflect students' diverse experiences and cultural realities. This paper critically examines the existing teaching practices in the Python programming module, analyzes student feedback, and proposes an action plan to enhance inclusivity and accessibility. By combining theoretical insights with practical strategies, this study aims to bridge the gaps in current teaching practices and foster a more equitable learning environment

Python programming is an essential skill for CVE students, enabling them to analyze construction data and automate processes effectively. The Civil Engineering Skills module integrates Python coding with real-world engineering applications. However, disparities in prior coding experience, cognitive barriers, and cultural diversity present challenges to achieving inclusivity and accessibility in teaching. This paper explores these challenges, reflects on current teaching practices, and

Manuscript received February 20, 2025; accepted June 3, 2025; published September 18, 2025.

proposes actionable strategies to enhance the learning experience for all students.

II. Methodology

This study employs a mixed-methods approach to evaluate the effectiveness of current teaching practices and identify areas for improvement. Firstly, current teaching practices were critically discussed based on inclusion and accessibility from the perspective of higher education. Second, data were collected through surveys administered to 70 CVE students enrolled in the Python programming module. The survey comprised both quantitative and qualitative components, focusing on four key areas: general experience (Table I), pacing and cognitive load (Table II), content clarity and accessibility (Table III), and inclusivity and support (Table IV). Quantitative ratings were analyzed to identify trends, while open-ended responses provided deeper insights into students' experiences and challenges (Table V).

The analysis is informed by relevant literature on inclusive education, including Universal Design for Learning [1], culturally responsive teaching [2], and the intersection of cognitive and cultural barriers in programming education [3, 4]. These frameworks guide the development of an action plan aimed at addressing the identified challenges.

TABLE I. Student Feedback on General Experience

No.	Questions
1	How engaging did you find the live coding sessions? (Not Engaging = 1, Extremely Engaging = 5)
2	How comfortable do you feel following along with the coding demonstrations in real-time? (Very Uncomfortable = 1, Very Comfortable = 5)
3	How well do you understand the concepts covered in the live coding sessions? (Very Poorly = 1, Very Well = 5)

TABLE II. Student Feedback on Pacing and Cognitive Load

No.	Questions
4	Do you find the pace of the live coding sessions manageable? (Never Manageable = 1, Always Manageable = 5)
5	Do you feel overwhelmed by the amount of information presented during the sessions? (Always = 1, Never = 5)
6	Would you benefit from more pauses or breaks during the sessions to process the information?

TABLE III. Student Feedback on Content Clarity and Accessibility

No.	Questions
7	How clear are the explanations provided alongside the coding demonstrations? (Very Unclear = 1, Very Clear = 5)
8	Do you have any difficulty understanding the programming terminology used in the sessions? (High Difficulty = 1, No Difficulty = 5)
9	Are the visual elements (e.g., code displayed on the screen) easy to follow? (Very Difficulty = 1, Very Easy = 5)

TABLE IV. Student Feedback on Inclusivity and Support

No.	Questions
10	Do you feel that the sessions are inclusive and accessible for all students, regardless of prior coding experience? (Very exclusive = 1, Completely = 5)
11	Do you think additional resources (e.g., pre-tutorial materials, video recordings) would help improve your learning experience? (No, Not at all = 1, Yes, it would be very helpful = 5)

TABLE V. Open Ended Questions for Student Feedback

No.	Questions
12	Please tick which learning activities would be most helpful for improving your learning experience?
13	What do you like most about the live coding sessions?
14	What challenges do you face when following along with the live coding sessions?
15	Are there any specific areas where you think the sessions could be improved?
16	Do you have any suggestions for making the sessions more inclusive or accessible for all students?

III. Critical Reflection on Teaching Practice

A. Disparity in Prior Coding Experience

Civil Engineering students enter the Python programming module with diverse levels of coding experience, a challenge highlighted by the initial survey conducted at the beginning of the course. While many students possess strong mathematical and logical problem-solving abilities, their familiarity with programming concepts varies significantly. This disparity creates challenges in ensuring all students progress at the same pace. Research by Bittermann *et al.* [5] emphasizes how differences in prior knowledge can lead to unequal learning outcomes, often leaving students with limited programming backgrounds at a disadvantage. Additionally, students with greater prior experience demonstrate higher engagement and efficiency in processing new information [6].

The tutorial-based, interactive format adopted for the module assumes students can adapt quickly to new programming concepts. While engaging for some, this approach places students with limited experience under significant strain. These learners often struggle to process information and execute code at the same pace as their peers, leading to frustration and reduced confidence. Such discrepancies exacerbate learning inequalities and disadvantage students who require a more methodical, step-by-step approach. Shen *et al.* [7] advocate for learning-by-doing methods, supported by interactive computer tutors, as a way to address these disparities. Immediate hands-on practice, combined with structured guidance, has been shown to improve engagement and comprehension, making it a crucial consideration for educators.

B. Cognitive Overload and Pace of Learning

Interactive live coding sessions are designed to mimic real-world problem-solving environments. However, the fast-paced nature of these sessions can overwhelm students, particularly those with learning disabilities, attention challenges, or language barriers. Nieminen [4] notes that teaching formats requiring simultaneous processing of multiple streams of information—such as verbal explanations, on-screen code demonstrations, and real-time application—often disadvantage students who require additional time for comprehension and provide immediate feedback. Supported by Computer Science Education Research [6, 8], this approach enhances practical learning.

For many students, especially those unfamiliar with coding, the combination of rapid concept delivery, code troubleshooting, and error correction creates significant cognitive strain. CVE students, despite their strong foundations in mathematics and physics, often find coding unfamiliar, resulting in delays as they attempt to follow live demonstrations. Kioko and Makoelle [3] emphasize that teaching practices failing to account for varied processing speeds alienate certain learners, ultimately hindering their educational experience.

C. *Digital Accessibility Challenges*

The reliance on visual and verbal elements in tutorial-based lectures poses accessibility challenges, particularly for students with visual impairments. Although the current cohort does not include visually impaired learners, the lack of assistive tools such as screen readers limits the inclusivity of programming education. Hadwen-Bennett [9] identifies multiple barriers faced by visually impaired students, emphasizing the need for accessible programming environments. Meulen *et al.* [10] further argue that modern programming tools often exclude visually impaired learners due to their incompatibility with screen readers and other assistive technologies.

The absence of accessibility features in tutorial lectures risks excluding a broader range of students, including those with hidden disabilities or situational challenges. While not immediately evident in the current cohort, addressing these gaps proactively is essential to ensuring long-term inclusivity.

D. *Lack of Flexibility in Learning Styles*

The synchronous, step-by-step nature of live coding sessions assumes that all students learn effectively through real-time demonstrations. However, this approach does not align with the preferences of students who require more time for reflection or prefer alternative methods such as written materials or tutorials. Comprehensive lecture materials were provided to students in advance, but these resources often lacked the interactivity necessary to engage learners during live sessions.

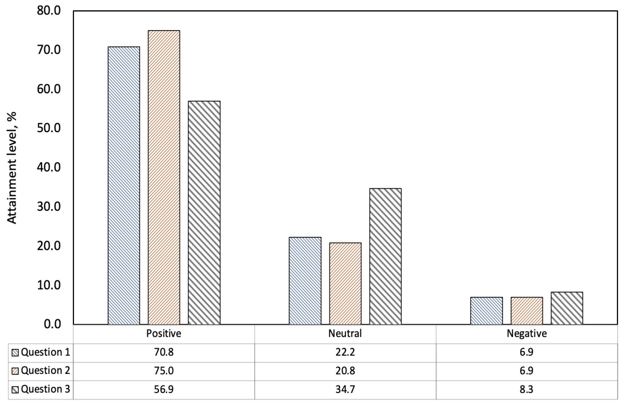
Dolmage [11] critiques traditional instructional methods for prioritizing the needs of a narrow range of students while marginalizing others who benefit from diverse teaching strategies. Without flexible teaching modalities, students requiring additional time or different forms of engagement may struggle to keep pace with their peers. Addressing these limitations necessitates the integration of multimodal instructional methods, such as asynchronous tutorials, annotated guides, and self-paced exercises, which cater to a broader spectrum of learning styles.

IV. Feedback and Analysis

A. *Engagement and Comfort*

The engagement score (3.94/5.00) indicates that most students found the sessions engaging, with 70.8% rating them as 4 or 5. However, 6.9% rated engagement as low (2), highlighting a gap in addressing the needs of all learners (Fig. 1). Engagement appeared to correlate with prior coding experience; students with little or no prior experience struggled to keep up, reducing their ability to engage fully. This underscores the importance of scaffolding through pre-session tutorials or introductory materials to support diverse learners. Open-ended responses revealed specific requests for scaffolding materials, such as annotated code examples and video demonstrations, to better prepare for sessions.

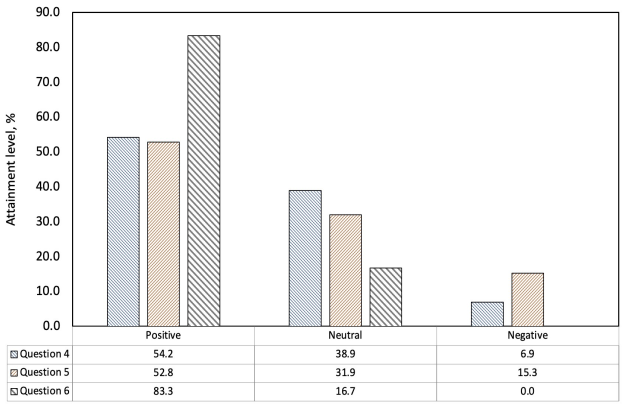
The comfort score (4.00/5.00) reveals that while 75.0% of students felt comfortable following the live coding demonstrations, 6.9% experienced discomfort. This discomfort likely stemmed from the fast-paced delivery and cognitive overload. Real-time coding demonstrations require students to simultaneously process verbal explanations, on-screen code, and application concepts, which can be overwhelming for neurodiverse learners or those unfamiliar with programming. Nieminen [4] emphasizes the importance of reducing cognitive load to create more inclusive learning environments. Incorporating digital accessibility tools, such as annotated codes and live captioning, could help alleviate these challenges.



	Positive	Neutral	Negative
▨ Question 1	70.8	22.2	6.9
▨ Question 2	75.0	20.8	6.9
▨ Question 3	56.9	34.7	8.3

Fig. 1. Distribution of the attainment level for the student's learning experience on engagement and comfort.

B. *Pacing and Cognitive Load*



	Positive	Neutral	Negative
▨ Question 4	54.2	38.9	6.9
▨ Question 5	52.8	31.9	15.3
▨ Question 6	83.3	16.7	0.0

Fig. 2. Distribution of the attainment level for the student's learning experience on cognitive load.

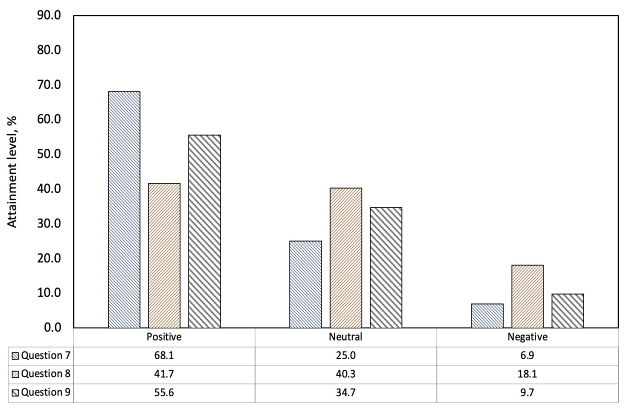
The session pace received a neutral rating from 38.9% of students, with 6.9% finding it rarely manageable (Fig.

2). Cognitive overload emerged as a recurring theme, particularly among students with learning disabilities and non-native English speakers. Live coding's fast pace often alienates students who need more time to process information, troubleshoot errors, and grasp new concepts. Harrington *et al.* [2] argue that teaching practices must accommodate varying processing speeds to ensure inclusivity.

C. *Clarity and Accessibility*

The clarity score (3.85/5.00) suggests that misunderstandings during live sessions hinder confidence and participation. Notably, 25.0% of students rated the explanations as neutral or unclear. Terminology posed a significant barrier, with 40.3% of students indicating moderate difficulty in understanding programming jargon (Fig. 3). This highlights the need for clearer instructional methods and supplementary resources.

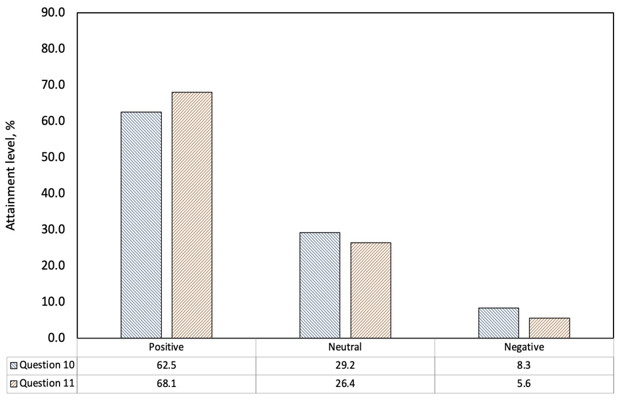
Proposed Interventions: Multimodal teaching aids, such as diagrams, flowcharts, and annotated materials, can enhance comprehension. Asynchronous resources, including video demonstrations and written guides, allow students to review and practice at their own pace. These resources address the learning gaps for students who struggle during live sessions.



	Positive	Neutral	Negative
▧ Question 7	68.1	25.0	6.9
▨ Question 8	41.7	40.3	18.1
▧ Question 9	55.6	34.7	9.7

Fig. 3. Distribution of the attainment level for the student's learning experience on clarity and accessibility.

D. *Inclusivity and Support*



	Positive	Neutral	Negative
▧ Question 10	62.5	29.2	8.3
▨ Question 11	68.1	26.4	5.6

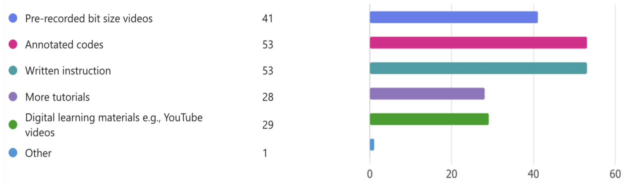
Fig. 4. Distribution of the attainment level for the student's learning experience on inclusivity and support.

The inclusivity score (3.81/5.00) reflects mixed perceptions of the sessions' accessibility (Fig. 4). Qualitative feedback emphasized the importance of culturally responsive examples and case studies. Singapore's multicultural context necessitates teaching practices that acknowledge and value students' diverse experiences.

Proposed Interventions: Incorporating case studies relevant to different cultural and engineering contexts can enhance engagement and inclusivity. Institutional support for digital accessibility features, such as screen readers, is essential to address hidden or situational challenges.

V. Action for Enhancing Learning Experience

The survey results (Fig. 5) provide significant insights into students' preferences for learning activities that enhance their experience in Python programming tutorials. These findings underscore the importance of designing inclusive, accessible, and flexible learning environments, particularly for Civil Engineering students with diverse backgrounds and varying levels of prior coding experience. This section critically analyzes the results in the context of existing literature, highlighting their implications for teaching practices and future research. By integrating UDL principles, culturally responsive teaching, and flexible modalities, the plan seeks to create a more equitable learning environment. However, further testing and iteration are necessary to evaluate the effectiveness of these strategies in a live coding context.



● Pre-recorded bit size videos	41
● Annotated codes	53
● Written instruction	53
● More tutorials	28
● Digital learning materials e.g., YouTube videos	29
● Other	1

Fig. 5. Preferred learning activities for enhancing the student's learning experience.

A. *Annotated Codes and Written Instructions: Bridging Knowledge Gaps*

The overwhelming preference for annotated codes and written instructions (53 votes each) underscores their critical role in fostering inclusivity and comprehension. These resources provide scaffolding for students who may struggle to follow fast-paced, live coding demonstrations, as supported by Universal Design for Learning (UDL) principles [1]. Annotated codes act as a roadmap, demystifying the thought processes behind coding decisions, while written instructions offer an opportunity for students to revisit content at their own pace.

This finding aligns with Bittermann *et al.*'s [5] discussion on addressing disparities in prior knowledge, where providing clear, accessible materials can help level the playing field for students with varying experience levels. Additionally, these resources are particularly beneficial for non-native English speakers who may face language barriers in comprehending technical jargon [12]. Future iterations of the module should prioritize the development of comprehensive annotated codes and detailed written guides to support diverse learners effectively.

B. *Pre-recorded Bite-Sized Videos: Flexibility and Accessibility*

The preference for pre-recorded bite-sized videos (41 votes) reflects a growing demand for self-paced learning tools that offer flexibility. These videos allow students to control the pace of their learning, pausing, rewinding, and revisiting content as needed. This aligns with Shen *et al.*'s [7] emphasis on "learning by doing", where immediate application and practice are essential for mastering programming skills.

Moreover, bite-sized videos reduce cognitive overload by breaking down complex topics into manageable segments, a strategy supported by Nieminen's [4] recommendations for minimizing cognitive load in technical education. To maximize their effectiveness, these videos should include captions, transcripts, and visually engaging content to cater to a broad range of learning styles and abilities.

C. *Digital Learning Materials: Beyond the Classroom*

The preference for digital learning materials, such as YouTube videos (29 votes), highlights the value of external resources in complementing formal instruction. Platforms like YouTube provide diverse, community-driven content that can address specific topics in engaging and innovative ways. This aligns with findings from Kioko and Makoelle [3], who emphasize the importance of leveraging external resources to enhance inclusivity.

However, relying solely on external content may not fully align with course objectives or address the unique needs of the student cohort. To bridge this gap, educators should curate a list of high-quality digital resources that align with the curriculum and integrate them into the learning experience. This approach ensures consistency and relevance while providing students with additional avenues for exploration and practice.

D. *More Tutorials: Hands-on Practice and Collaborative Learning*

The demand for more tutorials (28 votes) emphasizes the importance of hands-on practice and real-time feedback in mastering programming skills. Tutorials foster active learning, allowing students to apply concepts in a supportive environment where they can seek clarification and collaborate with peers. This preference aligns with Dolmage's [11] critique of traditional teaching methods, which often overlook the need for diverse engagement strategies.

Expanding tutorial sessions or incorporating small-group discussions and project-based assessments can address this need. These approaches not only enhance understanding but also build a sense of community among students, particularly in a multicultural setting like Singapore.

VI. Conclusion

The survey results underscore the importance of adopting inclusive, accessible, and flexible teaching practices to meet the diverse needs of Civil Engineering students in Python programming tutorials. By leveraging annotated codes, written instructions, pre-recorded videos, and expanded tutorials, educators can create a more equitable and effective learning environment. These strategies not only address current challenges but also lay the foundation for continuous improvement and innovation in programming education.

While the survey provides valuable insights, further research is needed to evaluate the effectiveness of these proposed strategies in practice. Longitudinal studies tracking student performance and engagement before and after implementing these changes could provide empirical evidence of their impact. Additionally, qualitative methods such as focus groups and interviews could offer deeper insights into students' experiences and preferences, particularly for underrepresented groups such as non-native English speakers and students with disabilities.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

YK and SD designed this research framework; YK and JAS wrote the paper and analyzed the data; YK reviewed and edited the paper; all authors had approved the final version.

References

[1] S. Baglieri, "Toward inclusive education? Focusing a critical lens on universal design for learning," *Canadian Journal of Disability Studies*, vol. 9, no. 5, pp. 42–74, 2020.

[2] C. N. Harrington, A. Desai, A. Lewis, S. Moharana, A. A. Ross, and J. Mankoff, "Working at the intersection of race, disability and accessibility," in *Proc. the 25th International ACM SIGACCESS Conference on Computers and Accessibility*, New York, 2023, pp. 1–18.

[3] V. K. Kioko and T. M. Makoelle, "Inclusion in higher education: Learning experiences of disabled students at Winchester University," *International Education Studies*, vol. 7, no. 6, pp. 106–116, 2014.

[4] J. H. Nieminen, "Assessment for inclusion: Rethinking inclusive assessment in higher education," *Teaching in Higher Education*, vol. 29, no. 4, pp. 841–859, 2024.

[5] A. Bittermann, D. McNamara, B. A. Simonsmeier, and M. Schneider, "The landscape of research on prior knowledge and learning: A bibliometric analysis," *Educational Psychology Review*, vol. 35, no. 2, p. 58, 2023.

[6] T. Dong and G. Yang, "Towards a pattern language for interactive coding tutorials," in *Companion Proceedings of the 4th International Conference on Art, Science, and Engineering of Programming*, Porto, 2020, pp. 102–105.

[7] R. Shen, D. Y. Wohn, and M. J. Lee, "Comparison of learning programming between interactive computer tutors and human teachers," in *Proc. the ACM Conference on Global Computing Education*, Hyderabad, 2019, pp. 2–8.

[8] M. Seyam, D. S. McCrickard, S. Niu, A. Esakia, and W. Kim, "Teaching mobile application development through lectures, interactive tutorials, and Pair Programming," in *Proc. 2016 IEEE Frontiers in Education Conference (FIE)*, Erie, 2016, pp. 1–9.

[9] A. Hadwen-Bennett, S. Sentance, and C. Morrison, "Making programming accessible to learners with visual impairments: A literature review," *International Journal of Computer Science Education in Schools*, vol. 2, no. 2, pp. 3–13, 2018.

[10] A. V. der Meulen, M. Hartendorp, W. Voorn, and F. Hermans, "The perception of teachers on usability and accessibility of programming materials for children with visual impairments," *ACM Transactions on Computing Education*, vol. 23, no. 1, pp. 1–21, 2022.

[11] J. T. Dolmage, *Academic Ableism: Disability and Higher Education*, University of Michigan Press, 2017, p. 244.

[12] P. J. Guo, "Non-native English speakers learning computer programming: Barriers, desires, and design opportunities," in *Proc. the 2018 CHI Conference on Human Factors in Computing Systems*, Montreal, 2018, pp. 1–14.