Board 297: Foundational Strategies to Support Students with Diverse Backgrounds and Interests in Early Programming

Aakash Gautam, San Francisco State University
Prof. Shasta Ihorn

Shasta Ihorn is an Assistant Professor of Psychology at San Francisco State University.

Prof. Ilmi Yoon

Professor Ilmi Yoon, Professor of Computer Science at San Francisco State University (SFSU), is an expert in gamification and game development, particularly in interactive media, 3D over the Internet, and network information visualization. She has collabo

Anagha Kulkarni, San Francisco State University

Anagha Kulkarni is an Associate Professor of Computer Science at San Francisco State University. Her research investigates problems at the intersection of information retrieval (IR), natural language processing (NLP), and machine learning (ML). Her work a

Michael Savvides, San Francisco State University

Foundational Strategies to Support Students with Diverse Backgrounds and Interests in Early Programming

Introduction

Previous research has identified numerous challenges in teaching computer programming in the classroom, including students' varying prior knowledge and experiences [1, 2]. These challenges have drawn attention to various pedagogical strategies and curricular materials [3, 4, 5]. The problem has also prompted a number of technological advances and design solutions [6, 3]. As a field, we have moved forward. However, there remains a significant gap in making introductory programming courses accessible to all students.

Students in introductory programming classes come from diverse backgrounds and have a wide range of interests. Some have completed comprehensive introductory computing courses in high schools, while others have not. Some students have access to a rich ecosystem of computing resources, whilst others do not. These characteristics are heavily influenced by larger historical, social, and economic challenges. Individual abilities and interests also differ. Some students are less comfortable exploring computing systems, while others are more at ease exploring uncertain technological problems.

Furthermore, because the content of introductory programming is deemed "simple enough", many institutions with limited resources, including ours, rely on graduate students to teach courses. Graduate students frequently teach for a semester or two before they graduate. As a result, many introductory programming instructors do not have the time or resources to iterate and enhance their pedagogical practice. As student interest in computing grows, introductory programming class sizes are expanding, requiring institutions to provide several sections of the same course, resulting in variations in instruction quality and student learning gains.

Our ANON project is situated within these pedagogical and institutional complexities. The project aims to support low-income students in their early computing journey. A cohort of freshmen students participates in a year-long co-curricular program supported by a network of educators and peer mentors. The project places a strong emphasis on fostering student retention in introductory programming classes by providing academic and community support.

In this paper, we focus on a week-long workshop conducted between the two academic terms of the program, where we implemented three evidence-based foundational practices for supporting students with diverse backgrounds and interests in introductory programming. These three practices are: (1) enabling multiple encounters with programming constructs, (2) facilitating collaborative learning, and (3) implementing pedagogical strategies for differentiation. These three practices are not novel; in fact, they are supported by extensive research in computing education and cognitive science [7, 8, 9, 10]. We provide reflections on strategies to adapt these practices to support instructors in resource-constrained settings in enabling computing for all.

Methodology

The approach discussed in the paper is exploratory and incremental. The first author, who also teaches an introductory programming course, observed that towards the end of the semester, many students who completed his introductory programming course voiced uncertainty regarding various concepts covered in the class. The discussion with the students shed light on the need for more hands-on practice to master the concepts. This sparked the idea for a separate week-long workshop for a group of students who were both involved in the ANON project and had recently finished the introductory programming course. The first workshop was held in January 2022 with a group of ten students, and it was repeated in January 2023 with a group of eleven students.

Workshop Sessions

Given the diversity of the student body, students were asked to highlight issues covered in the introductory programming class that they struggled with before to the workshop and on the first day of the workshop. The gathered topics were introduced primarily through a three-way structure: (1) large group discussion with an example programming problem, (2) working with a smaller group to solve problems, allowing for more hands-on practice, and (3) homework assignment to work on the programming problems individually.

The majority of the workshop sessions followed the same format. The sessions began with a discussion of the topic at hand as well as session's goals. Certain sessions necessitated a more thorough introduction to the subject (e.g., object-oriented programming). During the session, we conducted small group discussions to uncover what students found easy or hard and why. This enabled us to tailor the subsequent hands-on programming session. Following the presentation and the discussion, we divided the students into smaller groups and provided them with problems to work on. Instructors and mentors were present in the smaller groups to scaffold their exploration of the problem. Typically, the students solved two problems, after which we all convened in the larger group to discuss. After the discussion, we again went back to the smaller groups to work on programming problems. The workshops involved senior undergraduate students who mentored the students in solving the problem. The near-peer mentor system allowed for more interaction and guidance.

Data Collection and Analysis

Different types of data were collected for this study. First, we conducted a survey with students who took introductory programming classes. We ran the survey with 251 students across three semesters from Fall 2021 to Fall 2022. The survey sought to understand the student's level of confidence in their ability to write programs and the importance they placed on programming in their future career.

Following the semester-long introductory programming classes, we conducted a week-long workshop for a small group of students who were part of the ANON project. We report on the two workshops that we conducted in January 2022 and January 2023. The data contains reflections noted down by the instructor and near-peer mentors during the workshop. We also report on the data from the pre-workshop and a post-workshop survey.

Findings

First, we report on the confidence and interest of students enrolled in introductory programming courses. We draw upon their response to list topics in introductory computer science that remained difficult for the students. These topics form the foundation of our week-long workshop. We then present our reflections on the workshop.

From the Initial Survey

In the first day of the introductory programming classes, students were asked to complete a survey. The survey aimed to understand if the students had access to reliable computer where they could install a software (IntelliJ) and a reliable Internet connectivity to submit assignments. This was included to ensure that lack of material resources did not hinder students' learning. Along with these questions, we asked the students about their prior experience with programming, their confidence in learning programming, and the importance they placed on programming in the future. For the latter two, students were asked to rate from 0 to 100, where 0 denoted minimum confidence or importance.

The students' prior experiences in programming varied. 106 students had no prior programming experience, while 145 had previously encountered programming. 52 of the 145 had prior knowledge of Java, the language used in our introductory and intermediate programming classes. 93 of the students had worked with a variety of programming systems and languages, including HTML, Scratch, Python, and in one instance, C++. Similarly, there was quite a wide variance in the level of confidence expressed by the students. 74 out of the 241 who shared their ratings, rated their confidence as greater than eighty percent. 23 respondents reported a level of Similarly, the significance they placed on programming ranged from 15 to 100, with an average of 84.

Variation was also apparent in the students' expressions regarding their motivation to join the class. A group of students with prior experience in the class expressed a desire to expand their knowledge by stating, "I want to reinforce my foundation for java" and, in another students' case " I have taken C# and I really enjoyed it and I do wanna extend my knowledge and move onto java". In contrast, a substantial number of students were concerned about their ability to learn in class. One student with a self-reported confidence level of 28 out of 100 stated, "I am nervous for this class because I have no background in coding". Few others noted the lack of confidence stemming from the misconception that programming is "math-intensive," with some asking, "Do I need to take math classes to help with this course?" We observed that students with greater confidence had a clearer idea of what they wanted to learn, whereas the lack of knowledge of the topics to be covered in the introductory course appeared to erode the students' confidence, as one student stated, "computer science intimidates me because it's almost like a new language to learn".

As stated in the introduction, this diversity in programming skills poses a challenge for adapting pedagogical strategies. Some schools can afford to create different level courses, but many institutions, like ours, are unable to provide nuanced differentiation in early class offerings. We must work within these constraints and develop pedagogical strategies to differentiate between students with diverse programming backgrounds and interests.

Observations and Reflections From the Workshop

We aimed to tailor the workshop to the students' experiences in their introductory programming course due to the disparity in students' prior programming knowledge, confidence, and perceived importance of programming. In order to accomplish this, we asked the students for a list of difficult topics, which we then covered in the workshop.

Below, we report the high-level observations and reflections from the workshop sessions.

Varying challenges in introductory programming: Comparable to the larger classes, students in the smaller group expressed varying levels of comfort with introductory programming topics. In our first workshop, conducted in January 2022, students were asked to write diary entries describing the topics they had found either interesting, difficult to learn but have mastered, or with which they continue to struggle. The diary entry pointed to stark differences in the student's level of comfort. For instance, one student found the introductory course to be a repetition of their high-school programming writing,"[The intro course] for me was like a review class. I was probably ahead or at least knew more than the average student in [the class] because I have taken an AP CSA course in high school which teaches mainly Java." In contrast, another student reported continuing difficulties with programming fundamentals: "I still have trouble remembering syntaxes ..."

We had focused on enabling smaller groups to meet and discuss problem-solving strategies, which seem to help. Ideas shared by near-peers facilitated in deeper discussion, drawing forth the nuances of the challenges faced and alternative approaches they could take. Whereas in a larger class or with an instructor-led discussion, some of the nuanced approaches could be missed, a collaborative discussion among near-peers facilitated in leveling the knowledge gap, especially for those who felt they were behind. Students' reflections on the workshop revealed the importance of small-group collaboration, as one student wished for "time for group activities or a time where we would solve questions as a group, in order to build community and make it easier to understand certain concepts from peers."

Difficulty in formulating strategies to get started: During the sessions, we noted that the students often struggled to get started on solving the problem. They needed scaffolds to support working on the problem, even on problems that they had mentioned having high confidence. They were able to solve the problem once they received the initial support. For instance, when a group of students encountered a problem that required them to work on compound Boolean conditions, we began with a discussion in which we charted their thought processes. Then, we worked to convert these processes into Java statements. On the subsequent days too, a significant number of students found it difficult to get started on solving the problem, such as in initializing arrays and using them in the problem.

Other students were able to devise the solution, writing it in English or pseudocode. Some students found it difficult to convert the steps into Java code. A student, for example, struggled in translating the conditions they had written into Java. The condition required compounding with both AND and OR. For example, students tried to use $1 < x < 10$ instead of $1 < x \;\&\&\; x < 10$. Even though the student had experience and confidence in writing AND and OR statements as

part of other if-else statements, they struggled to write the compound statement.

These observations suggest a need for a greater exposure to a variety of problem-solving strategies that would assist students in initiating the problem-solving process. Our introductory programming course, like many others across the country, emphasizes writing code but provides few opportunities for students to read other codes. As children progress from learning to read to reading to learn, we believe novice programmers should read code in order to learn to code. Indeed, student feedback also highlighted the need for more opportunities to view and study the code of others. When asked what they would have liked to see more of in the workshop, one participant mentioned the opportunity to "watch informative videos and also see coding projects and go over it step by step." In light of this, we have begun supplying students with code examples in our revised introductory programming course.

Barriers arising from fast-paced, broad introductory programming course: Learning is a journey where a learner incrementally construct newer understanding based on their existing knowledge and the available learning support mechanisms [11]. Vygotsky introduced the concept of a learner's Zone of Proximal Development, which is defined as "the distance between the actual developmental level as determined by independent problem solving and the level of potential development as determined by problem solving under adult guidance, or in collaboration with more capable peers." [11, pp.86]. Here, the guidance and collaboration serve as a support mechanism that will aid in the expansion of the learners' capacity to construct additional knowledge. In programming education, researchers have utilized Vygotsky's zone of proximal development to comprehend the trajectory of learning and to recommend interventions and scaffolds to support the learning process (e.g., [12, 13, 14]).

To facilitate a growing zone of proximal development, students must engage with the available scaffolds and work on problems. In our institution, as in many others across the nation, introductory programming courses move quickly and cover a wide range of topics. In our introductory course, we introduce basic elements of programming (variables, expressions and evaluation), Boolean expressions and conditional statements, loops, one and multi-dimensional arrays, functions and modular programming, fundamental concepts of object-oriented programming, and exception handling. All of these topics are introduced over the course of a semester, which consists of approximately 40 hours of lecture and recitation. This pace encourages a more performance-based learning style, in which students focus on earning a grade rather than mastering the concepts. More broadly, it diminishes students' opportunities to make mistakes, reach out to peers or teachers, and seek guidance.

During our workshop, we noted the challenge arising from the fast-paced course. One of the positive aspects of the workshop, according to the students, was the opportunity to master concepts that they had not had sufficient time to learn previously. This is evident, for instance, in a student's reflection on the workshop, in which they wrote, "Learning about objects, classes, inheritance, parent classes, child classes and more were very helpful since it was pretty briefly covered in [introductory programming class] at the end of the semester and I didn't feel like I gained a full understanding of it." In addition, students valued the opportunity to make mistakes afforded by the workshop, stating, "Overall, I feel that I just need a bit more practice writing arrays, nested loops, and understanding the file class's usage." Moreover, students appreciated the

space enabled by the workshop to make mistakes, noting, “Personally I learn better in the breakout rooms when I get to make mistakes.” Another participant appreciated the workshop, remarking, “... having a person to guide and help you was very helpful. Specially since there is very few students in the group it’s very easy to receive help from the mentor. Going over topics that are difficult or having trouble remembering was very refreshing to me.”

Working with fewer students allowed us to pay close attention to each student’s work and provide the support they required. It would have been much more challenging to provide similar support in a larger group, which is the typical situation instructors encounter when teaching introductory programming classes. Nonetheless, we recognize that the broad scope of topics and rapid introduction of them in an introductory programming course can have a negative effect on students, particularly those with no prior programming experience. As part of the course, students should be encouraged to master fewer but more fundamental programming concepts, according to our argument. This change would permit students to make more errors and provide opportunities to learn from those errors. We believe that this will go a long way toward providing differentiated and individualized assistance, even in larger classroom settings.

Discussion

In this paper, we describe our findings from two one-week workshop designed to assist students who had completed introductory programming courses. The workshop expanded on concepts from the introductory programming course that the students found difficult to grasp and had not mastered. We designed the workshop to enable collaborative learning, facilitate multiple encounters with programming constructs, and explored ways to provide tailored support to each student.

These three elements included in the workshop are foundational learning practices; however, in settings with limited resources such as ours, it remains difficult to implement them in larger classroom settings. Considering this, below, we list down three implications for introductory programming classes that we believe can enable us to support students with diverse backgrounds and interests in introductory programming classes.

- **Incorporate reading code as a critical approach to learn to program**. Students require multiple programming experiences. It is difficult to provide multiple encounters if we only emphasize writing programs. Reading existing code, which is analogous to reading-to-learn, can provide students with opportunities to learn diverse and alternative problem-solving strategies.
- **Facilitate collaborative learning opportunities by incorporating near-peer learning groups.** The success of programs such as Process oriented guided inquiry learning (POGIL) has paved the way for more collaborative learning in introductory classes. Programming and computing are, after all, highly collaborative endeavors.
- **Enable rooms to make mistakes.** Students’ varying prior programming experiences pose a challenge for introductory programming courses. This complicates the implementation of pedagogical strategies for differentiation. Encouraging students to explore, make mistakes, and learn from them can provide the space required for students to receive the appropriate

level of assistance. We observed that allowing such space may necessitate two closely related strategies: (1) reducing the breadth of the topics covered in the introductory course, and (2) introducing the topics slowly.

Conclusion

Supporting students with diverse interests and background to learning programming remains a challenging task, especially in resource-constrained settings. We argue that by enabling multiple encounters with programming constructs, facilitating collaborative learning, and implementing pedagogical strategies for differentiation, instructors can provide students with the support they need to succeed in introductory programming courses. While these practices are not novel, we noted during our workshop that these strategies are effective in improving student engagement and learning outcomes. By adapting these practices to resource-constrained institutions, we can help realize a broader and inclusive computing community of learners.

References

[1] 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.

[2] K. L. Lewis, J. G. Stout, N. D. Finkelstein, S. J. Pollock, A. Miyake, G. L. Cohen, and T. A. Ito, "Fitting in to move forward: Belonging, gender, and persistence in the physical sciences, technology, engineering, and mathematics (pstem)," *Psychology of Women Quarterly*, vol. 41, no. 4, pp. 420–436, 2017.

[3] C. Mayfield, S. K. Moudgalya, A. Yadav, C. Kussmaul, and H. H. Hu, "Pogil in cs1: Evidence for student learning and belonging," in *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1*, 2022, pp. 439–445.

[4] E. Soep, C. Lee, S. Van Wart, and T. Parikh, "Code for what," in *Popular Culture and the Civic Imagination: Case Studies of Creative Social Change*. New York University Press,, 2020, pp. 89–99.

[5] A. G. S. Raj, J. M. Patel, R. Halverson, and E. R. Halverson, "Role of live-coding in learning introductory programming," in *Proceedings of the 18th koli calling international conference on computing education research*, 2018, pp. 1–8.

[6] D. Weintrop and U. Wilensky, "Transitioning from introductory block-based and text-based environments to professional programming languages in high school computer science classrooms," *Computers & Education*, vol. 142, p. 103646, 2019.

[7] M. Kendal and K. Stacey, "The impact of teacher privileging on learning differentiation with technology," *International Journal of Computers for Mathematical Learning*, vol. 6, pp. 143–165, 2001.

[8] M. J. Van Gorp and S. Grissom, "An empirical evaluation of using constructive classroom activities to teach introductory programming," *Computer Science Education*, vol. 11, no. 3, pp. 247–260, 2001.

[9] C. McDowell, L. Werner, H. Bullock, and J. Fernald, "The effects of pair-programming on performance in an introductory programming course," in *Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, 2002, pp. 38–42.

[10] A. Gautam, W. Bortz, and D. Tatar, "Abstraction through multiple representations in an integrated computational thinking environment," in *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 2020, pp. 393–399.

[11] L. S. Vygotsky and M. Cole, *Mind in society: Development of higher psychological processes*. Harvard university press, 1978.

[12] J. Whalley and N. Kasto, "A qualitative think-aloud study of novice programmers' code writing strategies," in *Proceedings of the 2014 conference on Innovation & technology in computer science education*, 2014, pp. 279–284.

[13] A. R. Basawapatna, A. Repenning, K. H. Koh, and H. Nickerson, "The zones of proximal flow: guiding students through a space of computational thinking skills and challenges," in *Proceedings of the ninth annual international ACM conference on International computing education research*, 2013, pp. 67–74.

[14] N. Anderson and T. Gegg-Harrison, "Learning computer science in the" comfort zone of proximal development"," in *Proceeding of the 44th ACM technical symposium on Computer science education*, 2013, pp. 495–500.