A Complete Redesign of CS1 for Engineering Students

Mr. Yuxuan Chen, University of Illinois Urbana-Champaign

Yuxuan Chen is a Master of Science student in Computer Science at the University of Illinois Urbana-Champaign. His primary research interests focus on computer science education and artificial intelligence. He is dedicated to enhancing student learning experiences and accessibility in computing education through both innovative technology and research-driven teaching practices.

Mr. Chenyan Zhao, University of Illinois Urbana-Champaign

Master of Science student in Computer Science at University of Illinois Urbana-Champaign. Research interest in Automatic Short Answer Grading using AI

Mr. Kangyu Feng, University of Illinois Urbana-Champaign

Kangyu Feng is a student in the Master of Computer Science program at the University of Illinois Urbana-Champaign. His primary interests include artificial intelligence, machine learning, game development, and computer science education. He is passionate about enhancing the structure and content of foundational math and computer science courses.

Dr. Mattox Alan Beckman, University of Illinois Urbana-Champaign

Mattox Beckman is a teaching associate professor in the Siebel School of Computing and Data Science at the University of Illinois Urbana-Champaign. He earned his doctorate from UIUC in 2003 under Sam Kamin, specializing in programming languages. He was a senior lecturer at the Illinois Institute of Technology for thirteen years before returning to UIUC as a teaching faculty. He is currently interested in CS Education, particularly in regards to upper level courses such as programming languages and competitive programming.

Prof. Mariana Silva, University of Illinois Urbana-Champaign

Mariana Silva is a Teaching Associate Professor in the Siebel School of Computing and Data Science at the University of Illinois Urbana-Champaign and co-founder and CEO of PrairieLearn Inc., a company dedicated to empowering instructors with tools to enhance teaching workflows without compromising educational quality. Before joining CS@Illinois in 2017, she was a lecturer in the Department of Mechanical Science and Engineering at the same university for five years. Silva has extensive experience in course development across engineering, computer science, and mathematics and is passionate about advancing teaching innovations that benefit students and instructors alike. She is an expert in the development and application of computer-based tools for teaching and learning in large STEM university courses. Her current research investigates the use of educational technologies to enhance computer-based assessments and centralized computer-based testing centers. This includes leveraging Large Language Models (LLMs) for automated short-answer grading and the creation of robust, randomized question generators to improve equity, accessibility, and scalability in teaching, learning and testing practices.

A Complete Redesign of CS1 for Engineering Students

Abstract

Introductory computer science (CS) courses, also referred to as CS1, are essential for equipping computer science and other engineering students alike with foundational programming skills. However, traditional CS1 courses are tailored for CS majors, leaving engineering students without the specialized programming knowledge needed for their disciplines. This paper presents a complete redesign of the CS 101 course at the University of Illinois Urbana-Champaign to meet the specific needs of fundamental programming for engineering students. The new curriculum prioritizes teaching foundational programming concepts in lectures while providing engineering-focused applications in lab activities and mini-projects. New topics, such as pseudocode and debugging, were introduced in lectures to deepen engineering students' understanding of programming fundamentals and enhance adaptability to new programming languages. Homework and exams were restructured into computer-based assessments featuring auto-grading and randomized problem variations to encourage mastery learning. Bi-weekly mini-projects were designed to connect programming concepts and skills with practical engineering applications. To evaluate the impact of the redesigned CS 101 course, a CS1 assessment was developed to measure students' understanding of programming fundamentals, pseudocode interpretation, and Python-specific skills. Future work will focus on incorporating group activities into lab sessions, expanding mini-project offerings, and refining the assessment tools to further align with the needs of engineering students.

1 Introduction

Introductory computer science (CS) courses, commonly known as CS1 [1], serve a critical role in equipping students with important computational skills, including error handling strategies [2, 3], code-writing proficiency and syntactic accuracy [4, 5], and the development of viable mental models for problem-solving [6, 7, 8]. While traditionally designed for CS majors, CS1 courses are witnessing a surge in enrollment from non-CS students [9], as computational tools and methods are expanding to a wider range of fields outside CS. In response, researchers have explored approaches to creating or redesigning CS1 courses to fit the needs of non-CS major students [10].

At the University of Illinois Urbana-Champaign, CS 101 is required by most non-CS engineering students and has historically equipped these students with Python and MATLAB programming skills essential for solving engineering problems in their respective fields. However, over time, this course has struggled to maintain its relevance and effectiveness. Older teaching methods,

confusion between two programming languages, and an overemphasis on diverse engineering applications and Python libraries within lectures have diluted its focus on programming fundamentals. As a result, many engineering students lack the foundational programming knowledge needed to effectively apply computational tools in more advanced engineering courses. Moreover, this deficiency leaves students underprepared to meet the programming and computational demands of many engineering careers upon graduation. Recognizing these challenges, we seek to redesign CS 101 to better align with the needs of engineering students.

Previous research provides valuable insights into CS1 redesign. Efforts to redesign CS1 courses for non-CS major students have introduced specialized approaches tailored to different student interests. Some redesigns have focused on media computation [11, 12]. Others took a data-centric approach, emphasizing more on database management and data visualization [13, 14]. These approaches are primarily designed for general non-CS students and are interest-based, making them unsuitable for addressing the specific needs of engineering students.

Beyond interest-based approaches, some redesigns target specific student groups, such as those in STEM disciplines [15], including engineering [16]. For example, Brown [17] describes a systematic redesign of an engineering programming course using MATLAB using a Backward Design framework [18]. Peteranetz et al. [19] applies Computational Creativity Exercises to increase engineering students' learning of fundamental programming. These studies show that engineering-specific CS1 courses can be tailored to meet the unique needs of engineering students; however, they are either still under development [17] or have been designed for specific institutional contexts [19], making direct adoption challenging.

In this paper, we describe the complete redesign of the CS 101 for engineering students at the University of Illinois Urbana-Champaign, focusing on restoring core programming fundamentals in lectures while integrating diverse engineering applications into lab sections and bi-weekly mini-projects. Based on feedback from interviews with 14 faculty members across the CS and other engineering departments, we have developed a curriculum that balances CS1 programming fundamentals with practical engineering applications. This study was reviewed by the university's Institutional Review Board and determined to be exempt from Human Subjects Research (IRB24-1130).

The redesign process replaced library-specific lectures with new content aligned with CS1 learning objectives. This updated content includes topics such as pseudocode, computational thinking, and debugging skills – areas previously absent from the course but essential for engineering students to master [20, 21]. Engineering applications were moved from lectures to weekly lab sections. Additionally, we created three longer programming assignments, or mini-projects, that apply CS1 content and libraries to solve engineering problems in areas such as aerospace applications, structural analysis, and environmental data modeling. New homework assignments, aligned with the revised lecture content, were developed in PrairieLearn [22], an online platform designed to support mastery learning through randomized problem variants, auto-grading, and immediate feedback. We also incorporated frequent testing [23, 24, 25], with students completing bi-weekly quizzes in a secure and controlled testing environment.

We explored various methods that could help us evaluate the effectiveness of the redesigned CS

101 course and identified Concept Inventories (CIs) as a useful approach [26]. A CI is a standardized assessment tool designed to measure students' understanding of fundamental concepts within a particular domain. Initially introduced in physics [27], CIs have since been developed for use in various disciplines, including engineering [28, 29, 30, 31], mathematics [32], and CS [33, 34, 35]. Within CS1 specifically, the Foundational CS1 Assessment (FCS1) and its isomorphic successor, the Second CS1 Assessment (SCS1), were designed to evaluate students' CS1 programming knowledge using pseudocode [36, 37]. We initially sought to use existing CS1 CIs; however, since FCS1 and SCS1 were not publicly available, we developed our own CS1 assessment in PrairieLearn to evaluate fundamental programming knowledge among upper-level engineering students. While this assessment has not yet undergone the rigorous development process required to qualify as a validated Concept Inventory, we are actively refining it to meet these standards. Our goal is to establish a robust CS1 Concept Inventory capable of evaluating student understanding and guiding future course improvements.

The remainder of this paper is organized as follows. In the next section, we describe the methods used in the redesign of the CS 101 course. Section 3 focuses on measuring the impact of the redesigned course. In Section 4, we discuss the details of balancing the curriculum to align with engineering needs, and promoting adaptability to new programming languages through pseudocode. We address the limitations of our approach and outline directions for future work in Section 5.

2 Methods

The CS 101 course redesign was based on feedback from faculty in engineering and the CS department. Key changes included restructuring lecture content to focus on core programming concepts, introducing computer-based assessments to promote mastery learning [38], and incorporating mini-projects designed to connect programming principles with practical engineering applications.

2.1 Changes via a Community of Practice

This project aims to promote the sustainable adoption of evidence-based pedagogies through Communities of Practice (CoPs) [39, 40, 41, 42]. These CoPs consist of instructors dedicated to reform efforts and must include senior faculty to secure departmental commitment for sustaining innovations. While any number of enthusiastic lecturers and junior faculty can participate, senior faculty play a critical role in advocating for and embedding changes within the department. CoPs are encouraged to meet weekly to foster collaboration.

This model supports faculty commitment, spreads effective practices, facilitates evaluation, and provides just-in-time training. Following these principles, the initial project team included two teaching professors from the CS department — one experienced in course redesign and the other in teaching CS1 courses — and four engineering faculty who regularly teach courses requiring or benefiting from introductory programming as a prerequisite.

We initiated the redesign process by conducting faculty interviews to gather insights on how programming concepts, problem-solving skills, and pedagogical approaches could be designed to

address the unique needs of non-CS engineering undergraduate students. These interviews, conducted in the Fall of 2023, had two primary objectives:

- To identify the specific programming knowledge and skills that engineering faculty deemed essential for their students to succeed in both their courses and future professional practice, and
- To gather input from CS faculty on the key components and best practices for designing and delivering a robust CS1 introductory programming course.

We interviewed 10 faculty members from four engineering programs: Civil Engineering (CE), Aerospace Engineering (AE), Mechanical Engineering (ME), and Bioengineering (BioE). Table 1 highlights their key insights, which were derived by identifying recurring themes.

Programs	**Expectations of Engineering Students**
CE	- Retain programming skills after the class - Develop long-term adaptability to new programming languages - Learn problem-solving skills and best programming practices - Master foundational topics: file I/O, if-else statements, loops, lists/arrays
AE	- Understand numerical methods, integration, and solving equations - Familiarity with libraries: NumPy, SymPy, matplotlib, and pandas
ME	- Understand numerical methods, regression, and differential equations - Learn best programming practices and debugging skills - Use Jupyter Notebooks effectively - Master foundational topics: loops, list slicing, functional programming - Familiarity with libraries: NumPy, SymPy, matplotlib, SciPy, and pandas
BioE	- Learn random numbers, image processing, and line fitting - Understand how to handle high-dimensional data - Familiarity with libraries: Simulink from MATLAB, and matplotlib

Table 1: Summary of engineering faculty feedback on the programming tools and skills expected of students completing the CS1 course.

A key consensus among the engineering faculty was to adopt Python as the primary programming language for the redesigned CS 101 course. MATLAB would be included only as an optional component in mini-projects tailored for BioE students who need it for future coursework. The previous course design attempted to teach both MATLAB and Python simultaneously. However, we learned from the interviews that engineering students entering subsequent courses often lacked programming proficiency and were unable to program independently, despite exposure to both programming languages. We believe that attempting to teach both languages led to confusion and insufficient mastery of either language. Establishing Python as the primary language for core lectures and labs aims to address this issue, providing a unified and focused learning experience for all students.

In parallel, we interviewed four CS faculty members, each with at least five years of experience teaching CS1 courses, to ensure the redesigned curriculum aligns with best practices in

introductory programming education. Their feedback provided valuable insights into structuring CS 101 to effectively teach fundamental programming concepts while cultivating strong problem-solving and critical thinking skills. Key suggestions from their feedback are summarized in Table 2.

Topics	- Basic programming: variables, data types, conditions, loops, functions, file I/O - Advanced skills: pseudocode, debugging, recursion, computational algorithms - Data structures: lists, arrays, dictionaries, classes - Libraries: pandas for data manipulation, NetworkX for graph-based computations
Goals	- Recognize programming patterns and form good learning habits - Develop debugging skills and learn to decompose complex problems - Achieve mastery of the basic programming topics (listed above) - Understand similarities and differences between programming languages
Methods	- Provide retakes/additional resources to help students recover from missed content - Streamline course content to prioritize depth over breadth - Reinforce learning by repeating a concept in different contexts - Encourage collaboration through structured group exercises - Provide real-world problems with scaffolded instructions to enhance motivation - Ensure consistency in topics, flow, and structure if multiple languages are taught

Table 2: Faculty feedback summary on CS1 foundational concepts, essential programming skills and teaching methodologies for an effective CS1 course.

The CS faculty highlighted the importance of foundational programming skills — such as understanding variables, data types (e.g. integers, floats, strings, booleans), loops, and functions — as essential building blocks for developing computational thinking. They also highlighted teaching advanced skills, including writing and interpreting pseudocode and breaking down complex problems into smaller, manageable components. Regarding teaching methodologies, they suggested providing opportunities for students who are falling behind to recover from missed content through retakes or additional resources, helping them stay on track. They recommended reducing the amount of content covered in lectures to focus on in-depth exploration of key concepts to prioritize mastery over broad coverage. They also suggested the idea of revisiting concepts in various contexts to reinforce learning, and encouraged structured group exercises. Both engineering and CS faculty emphasized the importance of problem-solving skills and cultivating best programming practices, reinforcing the priorities of the redesigned CS 101 curriculum.

2.2 Redesign of Lecture Content and Delivery Method

The previous lecture component of CS 101 attempted to combine engineering applications with foundational programming concepts, which seemed to result in a curriculum that appeared fragmented and less effective than intended. To address this, the lecture content was extensively restructured during Spring 2024, informed by faculty feedback, to focus exclusively on CS1 programming fundamentals. This shift aims to help students build a strong foundation in core programming skills.

The redesigned lecture topics are outlined in Table 3, organized into two weekly sessions. The curriculum begins with fundamental Python constructs, including Python data types, Boolean operations, and loops, and gradually introduces advanced concepts such as pseudocode and debugging. Later in the semester, students are introduced to widely used Python libraries like NumPy and pandas and learn to apply these libraries to solve engineering problems through mini-projects. From Week 3 onward, lab sessions reinforce lecture material through retrieval practice and hands-on exercises. The course concludes with review sessions to prepare students thoroughly for final assessments.

Week	**Lecture 1**	**Lecture 2**	**Lab**
1	Python Intro	Python Data Types	
2	Boolean Operations	Loops	
3	Functions I	Functions II	Lab Intro
4	Files	Dictionaries	Variables and Expressions
5	Pseudocode	Debugging	Loops and Range
6	Exceptions	Classes I	Mathematical Equations
7	Classes II	Inheritance	DNA Sequencing
8	Unit Testing	Libraries	Debugging Skills
9	Memory	matplotlib	Encoding and Decoding
10	NumPy I	pandas I	Probabilistic Prediction
11	NumPy II	pandas II	Censorship
12	Applications I	Applications II	SymPy, SciPy, other libraries
13	Review	Review	

Table 3: Weekly lecture topics and corresponding lab sections for the redesigned CS 101

The lectures now adopt an interactive coding approach, with lecture content integrated into a JupyterLab environment. Key points that were traditionally delivered via slides are now presented in Markdown-formatted cells, while executable code blocks illustrate the concepts being discussed. The instructor executes approximately half of the code blocks as part of the lesson, while students attempt the remaining blocks during class.

These Jupyter notebooks are hosted within a PrairieLearn workspace, allowing each student to access their own copy during class. Students are encouraged to replicate the instructor's work and engage with the in-class coding problems. For each problem, they are given three to five minutes to work and are encouraged to collaborate in pairs. This interactive setup fosters an environment where students can ask questions and quickly address minor issues, such as forgetting to import NumPy, that might otherwise slow their progress if working independently.

Each lecture typically includes three to four of these coding problems. After the allotted time, the instructor either demonstrates the solution or invites a volunteer to share and explain their approach, reinforcing understanding and encouraging active participation.

After class, students retain their own Jupyter notebooks, which include the instructor's code and

their attempts at the in-class coding problems. The instructor's version is also uploaded to a separate workspace on PrairieLearn, making it accessible to students for review. This resource helps students revisit the material covered in class, which is especially beneficial for those who were unable to take complete notes or solve all the problems during class time. It also provides valuable support for students who miss class. Although the class is currently conducted entirely in person, the increasing demand suggests that an online section may be needed in the future. We believe the interactive coding approach, combined with the availability of the instructor's notebook as an interactive Jupyter resource, can help online students engage with the materials in a way that is comparable to their in-person peers [43].

In addition to the in-class activities, each lecture typically includes three or four auto-graded problems outside the Jupyter notebook. These problems are slight variations of the examples presented in class and serve as formative assessments. Students are allowed unlimited attempts to complete them, and the problems contribute a small number of points to their overall grade. Beyond providing additional practice, these practice problems can provide an indication of how actively students are engaging with the course material.

2.3 Computer-based Assessments for Mastery Learning

The homework and exams from the previous CS 101 course were revised to align with the redesigned curriculum. New homework problems were created to address content gaps and ensure coverage of CS1 programming fundamentals. All formative assessments are now auto-graded, providing immediate feedback and allowing students unlimited attempts to achieve mastery.

For homework, students are provided with computer-based problems that generate randomized variations each time they are attempted. This allows students to practice solving different versions of the same type of problem repeatedly, enabling iterative improvement and a deeper understanding of the material — an option not available in the previous homework questions. To enhance engagement and learning, innovative problem formats were introduced beyond traditional multiple-choice and coding questions. One example, shown in Figure 1, features a pseudocode-based question inspired by the Rainfall Problem [44]. In this activity, students select and sort dragable blocks with proper indentation to construct pseudocode. This interactive approach, modeled after Parsons problems [45], addresses the limitations of traditional handwritten pseudocode assessments, which are time-consuming and difficult to grade due to syntax ambiguity [46], making the learning process more engaging and effective.

The exam structure has been redesigned to incorporate principles of frequent testing and mastery learning. Instead of relying on two high-stakes exams, the course now includes six quizzes administered every two or three weeks. During the final exam period, students have the option to retake one or more of these quizzes as "second-chance" quizzes. If a student's second-chance score improves upon their original score, the new score replaces the old one. If the second-chance score is lower, the original score is replaced with the average of the two. This approach effectively makes the final exam optional for students who are satisfied with their grades by the end of the semester.

Quiz scores account for 65% of the course grade. Given the rise of large language models (LLMs), a strong homework average is no longer a reliable indicator of student learning [47].

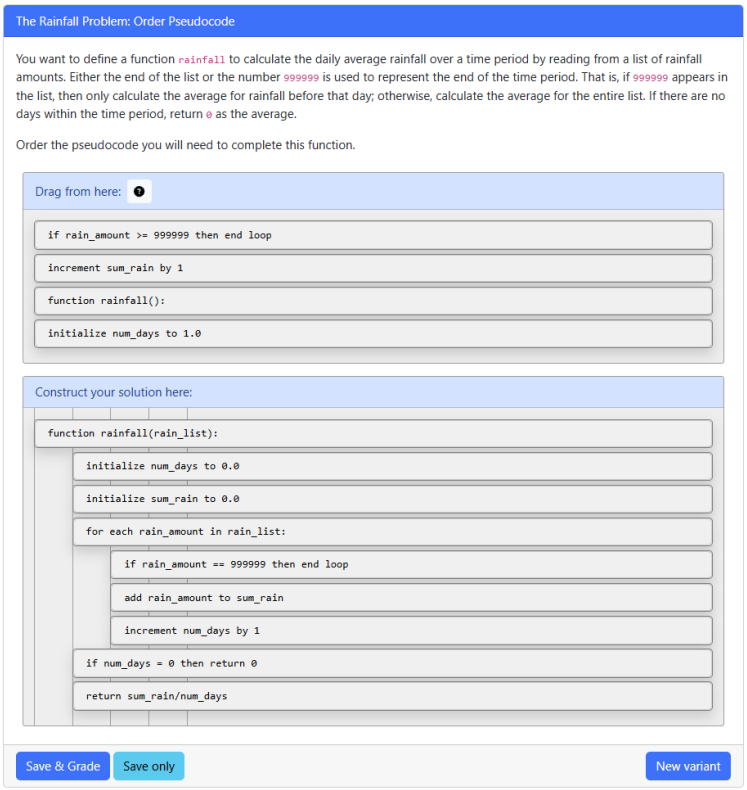


Figure 1: An example question from the pseudocode topic. Students drag and order blocks with proper indentation to create correct pseudocode that matches with the question prompt.

While the use of LLMs is not prohibited, students are encouraged to use these tools to enhance their understanding rather than rely on them as a substitute for learning. The weight of the quiz scores ensures that passing the class requires students to perform reasonably well (at least 50%) in a proctored and secured environment, maintaining the integrity of the assessment process.

2.4 Engineering Applications as Mini-Projects

While the lecture component of the redesigned CS 101 focuses exclusively on teaching CS1 programming fundamentals, the mini-projects bridge these foundational concepts with real-world engineering applications. These projects provide students with hands-on opportunities to apply their programming skills to solve practical problems relevant to various engineering fields.

Week	Topic	Library	Field
5	Spring System - Finite Element Method	NumPy, matplotlib	ME
9	Rocket System Calculation	SymPy, matplotlib	AE
11	University Rainfall Analysis	pandas, matplotlib	CE

Table 4: Mini-projects for CS 101

Three mini-projects were developed in PrairieLearn during Fall 2024, each targeting a specific

engineering discipline. Table 4 summarizes these projects, which cover diverse topics to ensure students from all engineering majors can engage with relevant real-world problems. Students are required to do all three mini-projects. Each mini-project requires approximately 5 hours of work and is completed over a two-week period. The mini-projects are distributed throughout the semester to maintain a balanced workload, and the total time commitment remains within the standard workload guidelines for the course. These mini-projects aim to motivate students by highlighting the value of programming in tackling engineering challenges within their own field as well as exposing them to applications in other engineering fields.

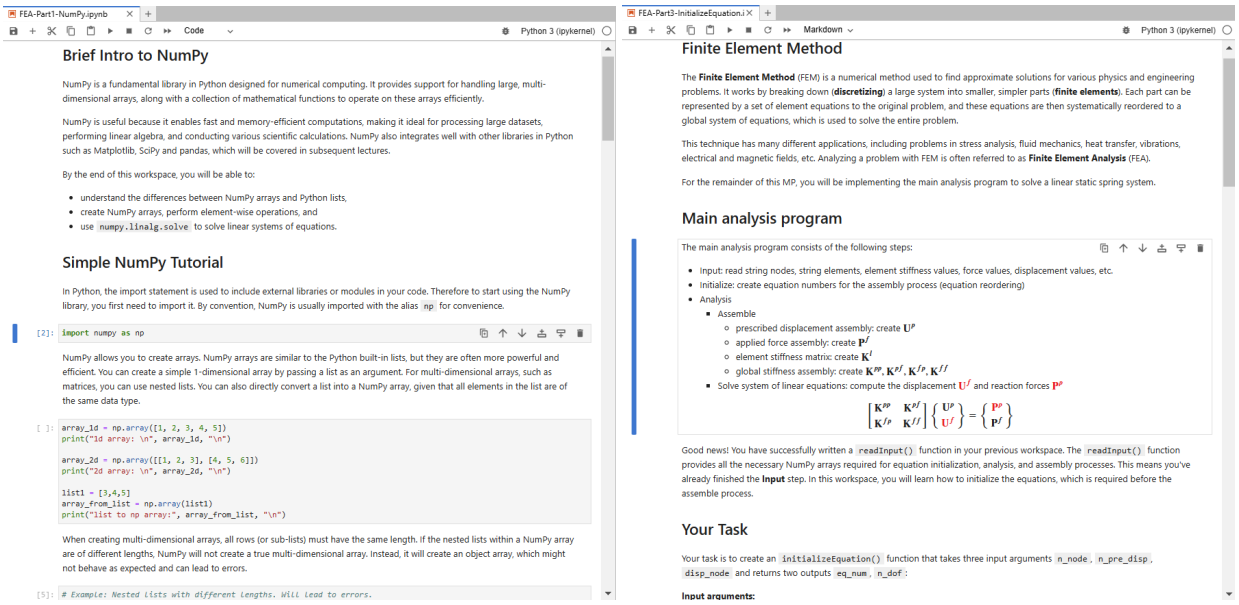
Each mini-project is divided into multiple parts, with each part being a fully developed task that builds on the concepts introduced in the previous part. Shown in Figure 2, progression locks are implemented to ensure that students can only access the next part of the mini-project after successfully completing the current one [38]. Students have unlimited attempts for each part, allowing them to iteratively refine their understanding and master the material at their own pace.

Figure 2: An example of the first mini-project showcasing the progression locking mechanism. Part 1 introduces a Python library NumPy. Sequential parts solve an engineering application using the library.

Every mini-project begins with an introduction to a Python library, presented as “Part 1” (Figure 3a). This in-depth library tutorial helps students quickly grasp the key features of the library, allowing students to apply them to solve engineering-specific problems in subsequent parts of the project (Figure 3b). This design allows students to learn about different Python libraries, understand their applications in engineering contexts, and gain practical experience simultaneously.

3 Measuring the Impact of the Redesigned Course

We developed a CS1 assessment during Summer 2024 to evaluate the effectiveness of the redesigned CS 101 course. This assessment was designed to assess fundamental programming knowledge among upper-level engineering students. It consists of 15 multiple-choice questions focusing on conceptual understanding, pseudocode interpretation, and basic Python programming topics. In Fall 2024, a total of 200 upper-level engineering students enrolled in structural analysis, fluid mechanics, and computational mechanics courses completed the CS1 assessment in a time slot of 50 minutes. Table 5 shows the gender distribution of these students. Note that these



(a) Part 1: NumPy (b) Part 3: InitializeEquation

Figure 3: Examples from Jupyter notebook sections of the first mini-project. Students start with an introduction to NumPy and progress to creating arrays to support the assembly process in the Finite Element Method.

students had not experienced the redesigned CS 101 course due to their academic progression; they either took the previous version of CS 101 or completed an entirely different CS1 course. This baseline dataset will be useful for future comparisons to evaluate the impact of the redesigned course on CS1 programming knowledge for engineering students.

Course	Male	Female	Non-Binary	Total
Structural Analysis	80	17	1	98
Fluid Mechanics	44	9	0	53
Computational Mechanics	43	4	2	49
Total	167	30	3	200

Table 5: CS1 assessment student distribution

Figure 4 shows the types of questions and the corresponding student scores on the CS1 assessment. Certain questions are variants of each other, with students randomly assigned to only one of the variants during the assessment. For example, a student might receive either the “If-Else v1” or “If-Else v2” question, but not both. Similarly, each student answered only one of the three “Function Parameters II” variants. These question variants were intentionally designed with structural similarities, differing only slightly to test equivalent knowledge domains.

Current results from our CS1 assessment indicate that students had difficulty solving two of the pseudocode interpretation questions, “List Summation” and “While Tracing”, achieving mean scores of 0.475 and 0.381 respectively. The “List Summation” question requires students to analyze pseudocode and select the best holistic description of its functionality. The “While

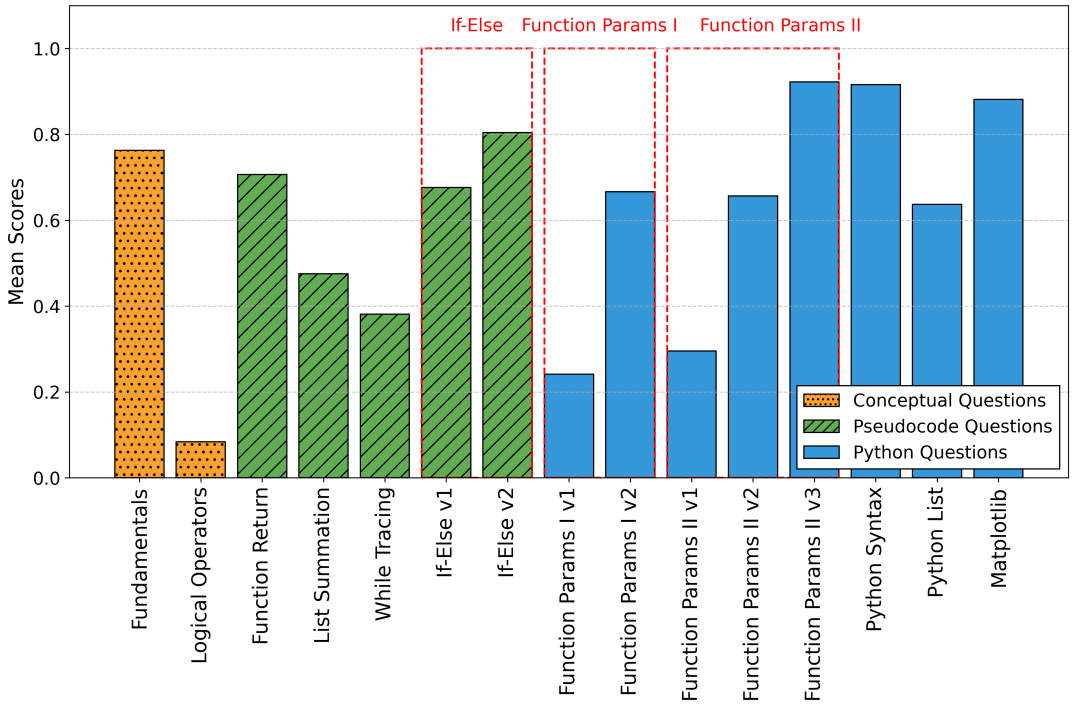


Figure 4: Mean scores of 200 upper-level engineering students who took the CS1 assessment, categorized by question types. Related question variants are grouped in red dashed boxes, with only one variant randomly assigned to each student during the assessment.

Tracing" question tests students' ability to trace the execution of a pseudocode while-loop step by step. These findings indicate that engineering students face challenges in both abstracting the overall purpose of code and accurately following its procedural logic. The "Logical Operators" conceptual question also received a significantly low mean score (0.084); however, we believe this is due to the level of difficulty of the question itself, which will be further explained in the next section.

We observed significant differences in students' performance on the Python programming questions with variants. A Mann-Whitney U test indicated a statistically significant difference on scores between "Function Parameters I v1" and "Function Parameters I v2" ($p < 0.01$). Similarly, pairwise comparisons using the Mann-Whitney U test among the three 'Function Parameters II" variants revealed statistically significant differences between each pair ($p < 0.0033$, Bonferroni-adjusted). These findings were unexpected, as the question variants were structurally similar and designed to assess the same knowledge domains.

4 Discussion

4.1 Balanced Course Curriculum

During the course redesign process, we conducted interviews with faculty members from the engineering and CS departments to identify the programming knowledge and skills engineering students needed to succeed in their coursework and future professional practice. However, it was

not feasible to incorporate all the suggestions from both groups. For example, CS faculty recommended including recursion as a topic in CS1 lectures (Table 2). After careful consideration, we decided not to include recursion in the initial offering, as it is a rather complicated CS concept that would require significant lecture time, potentially taking focus away from other fundamental programming concepts. We may introduce basic recursion in future course iterations if curriculum adjustments allow. Similarly, engineering faculty recommended teaching numerical methods, such as interpolation, integration, regression, and solving differential equations (Table 1). These are critical skills for engineering practice, but we believe they are beyond the scope of CS1. To address this, we introduced tasks in mini-projects where students could begin exploring numerical methods, such as learning and solving differential equations in an engineering application context. We made sure to provide sufficient guidance and simplified the code implementation process in these tasks. We informed students that numerical methods are covered in subsequent courses. The redesigned CS 101 curriculum should remain relevant without overloading engineering students with advanced topics; otherwise, it risks developing the same issues faced by the old CS 101.

4.2 Adaptability to New Programming Languages

Faculty from engineering and CS departments all shared valuable perspectives on teaching different programming languages. As previously mentioned, engineering faculty recommended only teaching Python in CS 101 as the primary programming language, but they expressed the importance of equipping students with the ability to adapt to new programming languages, as engineering fields frequently utilize different programming language tools like MATLAB, R, and SQL. Without the ability to adapt, students risk being stuck to Python, not having structural understanding of programming required to learn new languages with relative ease. During the interviews, CS faculty stated that understanding the similarities and differences between different programming languages is essential for developing transferable skills. Some CS faculty have experience in teaching multiple languages at the same time in their courses, and explained that maintaining consistency in concept topics and structure flow for the different languages was important.

Due to the scope of CS 101 focusing on a single language to avoid overwhelming engineering students, we recognized that we could not create scenarios of direct comparisons between different programming languages to guide students. This was one of the reasons we introduced pseudocode as a key component of the curriculum. By removing the unique syntax of specific programming languages, pseudocode allows students to focus on the underlying logic and structure of programming. We believe that emphasizing abstract concepts through pseudocode provides students with a deeper understanding of programming principles [48, 49]. We hope that this approach would encourage students to approach new languages with confidence in future courses.

5 Limitations and Future Work

We recognize the limitations of the current CS1 assessment and are taking steps to refine it for improved effectiveness and clarity. As mentioned in Section 3, the “Logical Operators”

conceptual question had the lowest mean score (0.084) among upper-level engineering students who recently took the assessment. After careful analysis and feedback from the course's teaching assistants, we believe the issue lies not in the students' understanding but in the question's difficulty level.

We also acknowledge an imbalance in the distribution of conceptual, pseudocode, and Python-specific questions in the original assessment. To address this, we will develop additional conceptual and pseudocode questions to achieve a more balanced distribution. We believe this adjustment will lead to a more comprehensive evaluation of students' skills across the three different question types.

Students enrolled in the redesigned CS 101 course may require several semesters to progress to the mid-level or upper-level courses recruited for the CS1 assessment. As they do, we will collect comparison data between these students and those who completed the previous version of the course. We will recruit students from eight additional engineering courses to pilot the updated CS1 assessment in Spring 2025, anticipating at least 500 participants. Over the next two years, we will continue data collection as part of our longitudinal study to measure the long-term effects of the CS1 redesign on engineering students' ability to apply computational tools in their respective fields.

We plan to revise the existing labs and incorporate group activities using the Process-Oriented Guided Inquiry Learning (POGIL) framework [50, 51]. POGIL is an instructional approach in which students work in structured groups with assigned roles, actively exploring concepts and constructing their own understanding rather than passively receiving information [52]. By integrating POGIL, we aim to create a more engaging and student-centered learning environment for CS 101 students.

Additionally, we will expand the scope of mini-projects in CS 101 by developing two new projects focused on real-world applications across different fields. The five mini-projects will be strategically distributed across weeks 3, 5, 7, 9, and 11 of the course schedule. To accommodate the needs of students in programs requiring MATLAB proficiency, we will create MATLAB versions of all mini-projects and lab activities. Students enrolled in programs that require MATLAB knowledge will be assigned to designated lab sections where they will receive targeted MATLAB support.

6 Conclusion

In this paper, we presented a complete redesign of the CS 101 course at the University of Illinois Urbana-Champaign to better support non-CS engineering students in developing fundamental programming skills. The revised curriculum emphasizes core programming concepts in lectures while integrating engineering-based applications through labs and mini-projects. Pseudocode was introduced in lectures to enhance adaptability to new programming languages. A CS1 assessment was developed to evaluate the impact of the course redesign. A longitudinal study will be conducted to assess the long-term effects of the redesign and guide future improvements.

Acknowledgments

We thank the Computers and Education research group at the University of Illinois Urbana-Champaign for their helpful feedback on earlier drafts of this paper. This material is based upon work supported by the Strategic Instructional Innovations Program in the Grainger College of Engineering at the University of Illinois Urbana-Champaign.

References

[1] M. Hertz, "What do CS1 and CS2 mean? investigating differences in the early courses," in *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '10. New York, NY, USA: ACM, 2010, p. 199–203. [Online]. Available: https://doi.org/10.1145/1734263.1734335

[2] G. Marceau, K. Fisler, and S. Krishnamurthi, "Measuring the effectiveness of error messages designed for novice programmers," ser. SIGCSE '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 499–504. [Online]. Available: https://doi.org/10.1145/1953163.1953308

[3] B. A. Becker, K. Goslin, and G. Glanville, "The effects of enhanced compiler error messages on a syntax error debugging test," in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '18. New York, NY, USA: ACM, 2018, p. 640–645. [Online]. Available: https://doi.org/10.1145/3159450.3159461

[4] A. Stefik and S. Siebert, "An empirical investigation into programming language syntax," *ACM Trans. Comput. Educ.*, vol. 13, no. 4, Nov. 2013. [Online]. Available: https://doi.org/10.1145/2534973

[5] A. Leinonen, H. Nygren, N. Pirttinen, A. Hellas, and J. Leinonen, "Exploring the applicability of simple syntax writing practice for learning programming," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 84–90. [Online]. Available: https://doi.org/10.1145/3287324.3287378

[6] L. Ma, J. Ferguson, M. Roper, and M. Wood, "Investigating the viability of mental models held by novice programmers," in *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 499–503. [Online]. Available: https://doi.org/10.1145/1227310.1227481

[7] A. P. Ambrósio, F. M. Costa, L. Almeida, A. Franco, and J. Macedo, "Identifying cognitive abilities to improve cs1 outcome," in *2011 Frontiers in Education Conference (FIE)*, 2011, pp. F3G–1–F3G–7.

[8] S. F. Mazumder and M. A. Pérez Quiñones, "The correctness of the mental model of arrays after instruction for cs1 students," in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, ser. SIGCSE 2024. New York, NY, USA: Association for Computing Machinery, 2024, p. 806–811. [Online]. Available: https://doi.org/10.1145/3626252.3630943

[9] L. J. Sax, K. J. Lehman, and C. Zavala, "Examining the Enrollment Growth: Non-CS Majors in CS1 Courses," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '17. New York, NY, USA: ACM, 2017, pp. 513–518. [Online]. Available: https://doi.org/10.1145/3017680.3017781

[10] J. Q. Dawson, M. Allen, A. Campbell, and A. Valair, "Designing an Introductory Programming Course to Improve Non-Majors' Experiences," in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '18. New York, NY, USA: ACM, 2018, p. 26–31. [Online]. Available: https://doi.org/10.1145/3159450.3159548

[11] L. Rich, H. Perry, and M. Guzdial, "A CS1 course designed to address interests of women," in *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '04. New York, NY, USA: ACM, 2004, p. 190–194. [Online]. Available: https://doi.org/10.1145/971300.971370

[12] C. B. Lee, "Experience report: CS1 in MATLAB for non-majors, with media computation and peer instruction," in *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '13. New York, NY, USA: ACM, 2013, p. 35–40. [Online]. Available: https://doi.org/10.1145/2445196.2445214

[13] D. G. Sullivan, "A data-centric introduction to computer science for non-majors," in *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '13. New York, NY, USA: ACM, 2013, p. 71–76. [Online]. Available: https://doi.org/10.1145/2445196.2445222

[14] R. E. Anderson, M. D. Ernst, R. Ordóñez, P. Pham, and B. Tribelhorn, "A Data Programming CS1 Course," in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '15. New York, NY, USA: ACM, 2015, p. 150–155. [Online]. Available: https://doi.org/10.1145/2676723.2677309

[15] J. C. Adams and R. J. Pruim, "Computing for STEM majors: enhancing non CS majors' computing skills," in *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '12. New York, NY, USA: ACM, 2012, p. 457–462. [Online]. Available: https://doi.org/10.1145/2157136.2157270

[16] F. Buitrago-Florez, M. Sanchez, V. Perez Romanello, C. Hernandez, and M. Hernandez Hoyos, "A systematic approach for curriculum redesign of introductory courses in engineering: A programming course case study," *Kybernetes*, vol. 52, no. 10, pp. 3904–3917, 2023. [Online]. Available: https://doi.org/10.1108/K-10-2021-0957

[17] P. R. Brown, "Work in progress: From scratch - the design of a first-year engineering programming course," in *2017 ASEE Annual Conference & Exposition*, no. 10.18260/1-2–29161. Columbus, Ohio: ASEE Conferences, June 2017, https://peer.asee.org/29161.

[18] G. Wiggins and J. McTighe, *Understanding by Design*, 1st ed. Alexandria, VA: ASCD, 1998.

[19] M. S. Peteranetz, A. E. Flanigan, D. F. Shell, and L.-K. Soh, "Helping Engineering Students Learn in Introductory Computer Science (CS1) Using Computational Creativity Exercises (CCEs)," *IEEE Transactions on Education*, vol. 61, no. 3, pp. 195–203, 2018.

[20] S. Gross, M. Kim, J. Schlosser, C. Mohtadi, D. Lluch, and D. Schneider, "Fostering computational thinking in engineering education: Challenges, examples, and best practices," in *2014 IEEE Global Engineering Education Conference (EDUCON)*, 2014, pp. 450–459.

[21] R. Chmiel and M. C. Loui, "Debugging: from novice to expert," in *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 17–21. [Online]. Available: https://doi.org/10.1145/971300.971310

[22] M. West, G. L. Herman, and C. Zilles, "Prairielearn: Mastery-based online problem solving with adaptive scoring and recommendations driven by machine learning," in *2015 ASEE Annual Conference & Exposition*. Seattle, Washington: ASEE Conferences, June 2015.

[23] R. L. Bangert-Drowns, J. A. Kulik, and C.-L. C. Kulik, "Effects of frequent classroom testing," *Journal of Educational Research*, vol. 85, 1991.

[24] F. M. Kika, T. F. McLaughlin, and J. Dixon, "Effects of frequent testing of secondary algebra students," *Journal of Educational Research*, vol. 85, pp. 159–62, 1992.

[25] J. W. Morphew, M. Silva, G. Herman, and M. West, "Frequent mastery testing with second-chance exams leads to enhanced student learning in undergraduate engineering," *Applied Cognitive Psychology*, vol. 34, no. 1, pp. 168–181, 2020.

[26] V. Fakiyesi, D. Fabiyi, I. Dunmoye, O. Olaogun, and N. Hunsu, "A Scoping Review of Concept Inventories in Engineering Education," in *2024 ASEE Annual Conference & Exposition*, no. 10.18260/1-2–46487. Portland, Oregon: ASEE Conferences, June 2024, https://peer.asee.org/46487.

[27] D. Hestenes, M. Wells, and G. Swackhamer, "Force Concept Inventory," *The Physics Teacher*, vol. 30, Mar 1992.

[28] K. E. Wage, J. R. Buck, T. B. Welch, and C. H. G. Wright, "The Signals and Systems Concept Inventory," in *the 2002 ASEE Annual Conference*, Jun 2002, pp. 1–29.

[29] A. Jacobi, J. Martin, J. Mitchell, and T. Newell, "A Concept Inventory for Heat Transfer," in *the Thirty-Third ASEE/IEEE Frontiers in Education*, Nov 2003.

[30] P. S. Steif and J. A. Dantzler, "A Statics Concept Inventory: Development and Psychometric Analysis," *Journal of Engineering Education*, Oct 2005.

[31] A. E. Barach, C. Jenkins, S. S. Gunawardena, and K. M. Kecskemety, "MCS1: A MATLAB Programming Concept Inventory for Assessing First-year Engineering Courses," in *2020 ASEE Virtual Annual Conference Content Access*, no. 10.18260/1-2–34958. Virtual On line: ASEE Conferences, June 2020, https://peer.asee.org/34958.

[32] J. Epstein, "Development and validation of the Calculus Concept Inventory," in *Proceedings of the ninth international conference on mathematics education in a global community*, vol. 9, 2007, pp. 165–170.

[33] G. L. Herman, M. C. Loui, and C. Zilles, "Creating the digital logic concept inventory," in *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '10. New York, NY, USA: ACM, 2010, pp. 102–106. [Online]. Available: http://doi.acm.org/10.1145/1734263.1734298

[34] L. Porter, D. Zingaro, S. N. Liao, C. Taylor, K. C. Webb, C. Lee, and M. Clancy, "BDSI: A Validated Concept Inventory for Basic Data Structures," in *Proceedings of the 2019 ACM Conference on International Computing Education Research*, ser. ICER '19. New York, NY, USA: ACM, 2019, p. 111–119. [Online]. Available: https://doi.org/10.1145/3291279.3339404

[35] R. Caceffo, S. Wolfman, K. S. Booth, and R. Azevedo, "Developing a Computer Science Concept Inventory for Introductory Programming," in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, ser. SIGCSE '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 364–369. [Online]. Available: https://doi.org/10.1145/2839509.2844559

[36] A. E. Tew and M. Guzdial, "The FCS1: a language independent assessment of CS1 knowledge," in *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '11. New York, NY, USA: ACM, 2011, p. 111–116. [Online]. Available: https://doi.org/10.1145/1953163.1953200

[37] M. C. Parker, M. Guzdial, and S. Engleman, "Replication, Validation, and Use of a Language Independent CS1 Knowledge Assessment," in *Proceedings of the 2016 ACM Conference on International Computing Education Research*, ser. ICER '16. New York, NY, USA: ACM, 2016, p. 93–101. [Online]. Available: https://doi.org/10.1145/2960310.2960316

[38] J. Garner, P. Denny, and A. Luxton-Reilly, "Mastery Learning in Computer Science Education," in *Proceedings of the Twenty-First Australasian Computing Education Conference*, ser. ACE '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 37–46. [Online]. Available: https://doi.org/10.1145/3286960.3286965

[39] G. L. Herman, L. Hahn, and M. West, "Coordinating college-wide instructional change through faculty communities," in *Proceedings of the 2015 International Mechanical Engineering Congress & Exposition*, Houston, TX, 2015.

[40] M. West, M. Silva, and G. L. Herman, "Sustainable reform of an introductory mechanics course sequence driven by a community of practice," in *ASME International Mechanical Engineering Congress and Exposition*, vol. 57588. American Society of Mechanical Engineers, 2015, p. V015T19A020.

[41] W. A. Fagen, G. L. Herman, and M. West, "Reengineering an introduction to computing course within a college-wide community of practice," in *Proceedings of the 2015 American Society for Engineering Education Annual Conference and Exposition*, Seattle, WA, 2015.

[42] G. L. Herman, I. B. Mena, J. Greene, M. West, J. Tomkin, and J. Mestre, "Creating institution-level change in instructional practices through communities of practice," in *Proceedings of the 2015 American Society for Engineering Education Annual Conference and Exposition*, Seattle, WA, 2015.

[43] J. M. Allen and F. Vahid, "Experiences in developing a robust popular online cs1 course for the past seven years," in *2020 ASEE Virtual Annual Conference Content Access*, no. 10.18260/1-2–34629. Virtual On line: ASEE Conferences, June 2020, https://peer.asee.org/34629.

[44] E. Soloway, "Learning to program = learning to construct mechanisms and explanations," *Commun. ACM*, vol. 29, no. 9, p. 850–858, Sep. 1986. [Online]. Available: https://doi.org/10.1145/6592.6594

[45] D. Parsons and P. Haden, "Parson's programming puzzles: a fun and effective learning tool for first programming courses," in *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52*, ser. ACE '06. AUS: Australian Computer Society, Inc., 2006, p. 157–163.

[46] B. A. Ulate-Caballero, A. Berrocal-Rojas, and J. Hidalgo-Céspedes, "Concurrent and distributed pseudocode: A systematic literature review," in *2021 XLVII Latin American Computing Conference (CLEI)*, 2021, pp. 1–10.

[47] A. Padiyath, X. Hou, A. Pang, D. Viramontes Vargas, X. Gu, T. Nelson-Fromm, Z. Wu, M. Guzdial, and B. Ericson, "Insights from social shaping theory: The appropriation of large language models in an undergraduate programming course," in *Proceedings of the 2024 ACM Conference on International Computing Education Research - Volume 1*, ser. ICER '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 114–130. [Online]. Available: https://doi.org/10.1145/3632620.3671098

[48] J. Kramer, "Is abstraction the key to computing?" *Commun. ACM*, vol. 50, no. 4, p. 36–42, Apr. 2007. [Online]. Available: https://doi.org/10.1145/1232743.1232745

[49] B. Yulianto, H. Prabowo, R. Kosala, and M. Hapsara, "Novice Programmer = (Sourcecode) (Pseudocode) Algorithm," *Journal of Computer Science*, vol. 14, no. 4, pp. 477–484, Apr 2018. [Online]. Available: https://thescipub.com/abstract/jcssp.2018.477.484

[50] S. R. Simonson, *POGIL: An Introduction to Process Oriented Guided Inquiry Learning for Those Who Wish to Empower Learners*. Sterling, VA: Stylus Publishing, 2019.

[51] C. Kussmaul, "Process Oriented Guided Inquiry Learning (POGIL) for Computer Science," pp. 373–378, 2012.

[52] H. H. Hu and T. D. Shepherd, "Using POGIL to help students learn to program," *ACM Trans. Comput. Educ.*, vol. 13, no. 3, Aug. 2013. [Online]. Available: https://doi.org/10.1145/2499947.2499950