2026 ASEE ANNUAL CONFERENCE June 21–24, 2026 | Charlotte Convention Center, NC

Host Sponsor UNC Charlotte

Paper ID #51641

Intro to Programming in the Age of AI: Innovative Approaches for First-Year Learners

Dr. Xiaoxiao Du, University of Michigan

Xiaoxiao Du is a Lecturer at the University of Michigan. She teaches robotics courses and supervises multidisciplinary teams and design experiences. She is interested in promoting student learning and team collaboration through innovative curricular design and data-informed evaluation.

Intro to Programming in the Age of AI:

Innovative Approaches for First-Year Learners

Abstract

This complete evidence-based paper presents several innovative approaches for first-year learners in an introduction to programming courses. As artificial intelligence (AI) becomes increasingly integrated into everyday technology, the way introductory programming is taught needs to evolve to prepare students for these new realities. This work closely examines a first-year introduction to programming course at a large Midwestern University as a case study. The course implemented several key methods to adapt instructions in the AI era by fostering student engagement, accessibility, and mastery. A central element of the course design is a semi-flipped classroom model. Core instructional content is delivered through "bite-sized" pre-recorded videos and readings, while in-class time is devoted to active learning, including hands-on project support and collaborative problem-solving. The PrairieLearn platform is employed to facilitate both pre-class preparation and in-person office hour/project help, offering students interactive and adaptive practice opportunities. Interactive presentation tools and online platforms, such as live polls and engaging sessions, are incorporated during class sessions to promote engagement, gauge student understanding, and encourage participation in real time. Additionally, frequent, low-stakes, mastery-based assessments, such as traditional pencil-and-paper quizzes, are integrated and complemented by digital platforms to provide real-time feedback and track student progress. To further boost motivation and contextualize learning, the course integrates hands-on robotics applications. These projects allow students to directly apply programming and algorithmic concepts to real-world robotic concepts and challenges, which provides a strong motivation for students to participate in the in-person learning experiences. Throughout the course, digital course materials and activities are designed to be accessible to all students, with the hope that students of varying backgrounds and abilities can fully participate and benefit from the curriculum. Through analysis of student feedback, performance data, and engagement metrics, we find these approaches show effectiveness in fostering understanding, mastery, and enthusiasm for programming, as demonstrated by high class performance and project completion and positive qualitative student feedback. We hope this work provides insights and practical recommendations for educators seeking to adapt introductory computer science education to the demands and opportunities presented by advances in AI, ultimately aiming to better equip first-year learners for future study and careers in computational and multidisciplinary engineering fields.

Introduction

As artificial intelligence (AI) becomes increasingly integral to technology and society, the way introductory programming is taught needs to adapt and evolve to better support first-year

learners. This paper presents a case study for a first-year "Introduction to Algorithms and Programming" course at a large Midwestern university and discusses several instructional approaches. Key innovations include robotics-based projects that connect programming concepts to real-world engineering, frequent mastery-based assessments, peer-assisted learning, and flexible, accessible materials tailored to diverse student needs. Analysis of student feedback and performance data demonstrates that these approaches foster greater engagement, mastery, and enthusiasm for programming, offering practical insights for educators seeking to prepare first-year learners for computational fields shaped by the rise of AI.

The concept of introductory engineering and programming courses have been much studied in the literature. Some common strategies include project-based learning [1], management simulation and game-based learning for design courses [2], non-traditional delivery such as online [3], peer mentoring [4], Lego activities and multimedia case studies [5]. Although relevant, the strategies proposed in this paper are distinct from these prior work in two primary aspects. Our course explores introductory programming in a robotics context (using real-world wheels robots as a platform for programming and algorithms learning and assessment). We also emphasize strategies that assist the instructional team in addressing concerns for "AI coding" by leveraging in-person mastery-based and robotic-project-based assessments, peer-assisted teaching, and real-world applications aspects.

As AI methods develop and as generative AI tools become increasingly accessible, there has also been recent work on generative AI use in first-year engineering. For example, Ref. [6] highlights the importance of recognizing the diverse student experience in first-year engineering courses and the complexity of student interactions with generative AI tools. Ref. [7] paved the way for a hands-on hardware AI experience and personally/culturally relevant learning interventions can influence student learning and career choice. Ref. [8] explored AI influence on "gateway" courses, such as calculus, in first-year engineering programs. Within these, the students' AI use ranges from "use AI as much as possible" to curated in-house course-specific AI tools. Overall, the finding was that AI use can be beneficial, but careful consideration of individualized learning due to diverse student backgrounds and experiences are needed. For programming courses, specifically, AI use can be difficult to track, evaluate and measure. If a student submits a piece of code for grading, how can instructors assess the students' true understanding of the codes? How to address a wide range of student prior experience and expertise in programming? How can the instructional team and the course curriculum support students in the age of AI? In the following sections, we aim to answer these questions by describing four approaches we implemented in a first-year introductory programming course. By using real-world robotic projects as a medium for programming practice and assessment, and by leveraging peer-assisted learning and teaching and accessible course materials and staff, we observed positive student performance and feedback in terms of concept mastery and positive learning experience. In the following sections, we first provide a brief overview of the course summary. Then, we describe each approach with

their results in detail. We conclude by providing discussions and reflection on the benefits and challenges of the approaches. We provide additional information about the course structure and learner profile in the Appendix.

Course Summary

This study closely examines a first-year introduction to programming course at a large Midwestern University as a case study. The course was designed for first-year learners and provides an introduction to programming, artificial intelligence algorithms, and computational thinking. The course was designed as part of the robotics curriculum and is currently being accepted in many college of engineering disciplines as part of the prerequisite courses, such as biomedical engineering, aerospace engineering, and computer science. This course currently fulfills the engineering first-year core requirement, and serves as one of the prerequisites for sophomore-level data structure and advanced programming courses.

The course consists of lecture content in programming concepts in C++ and Python, including syntax and structure, variables and data types, branching and iteration, operators and functions, and relevant programming libraries and packages. The course also covers autonomous navigation and search algorithms (e.g., breadth first search), introduction to models of computing through graphs and graph algorithms (e.g., finite state machine) and neural networks and machine learning fundamentals. The performance of the students are primarily evaluated in five hands-on, real-robot-based projects and five in-person quizzes (each testing a portion of the programming and algorithmic content) throughout the semester. In the Appendix, we provide additional information on the course structure, logistics, instructor and learner profile. We also express our gratitude to the course instructors, staff, and students.

Approach 1: Classroom Strategy

Pre-Learning

A central element of the course design is a "semi"-flipped classroom model. Core instructional content, including programming concepts and artificial intelligence algorithms, are pre-recorded and segmented into bite-size videos. Readings and reference materials are also provided when needed. The PrairieLearn[1] online assessment and learning system is used as a platform to host the videos and the lecture review exercises. Figure 1 shows an example ("Intro to Python") of the video and exercise page from the student view. The course content was grouped as different homework sections, a brief introduction is provided, and the pre-recorded instructor video is embedded. The students have the ability to play back, re-watch, start or pause the videos, and can, thus, learn the materials at their own pace. Then, a collection of lecture review questions are generated as PrairieLearn questions. Figure 1 also shows an example of a set of exercises for

[1] https://us.prairielearn.com/

C++. A variety of question types were used, including but not limited to multiple choice and checkbox answers, text answers and symbolic (math or coding syntax) expressions, match and order, etc. Students were assigned PrairieLearn exercises each week with specified due dates (before relevant in-class meeting time), and were expected to pre-watch the lecture recordings and complete the lecture review exercises.

The flipped classroom model has been discussed as an alternative instructional method in the literature across various disciplines [9] [10]. Prior work suggests flipped classrooms resulted in higher averages in personalization, student cohesion, individualization and higher self-efficacy. Students are able to make their own decisions over the way they learn, which students prefer [11]. Other work [12] also suggested the flipped classroom model offers lecture content and learning material for just-in-time learning tailored to the student's particular needs. For this course, the reason for adopting the PrairieLearn online learning model was inspired by the wide range of student experience. As seen in the "Learner Profile" section in the Appendix, students enrolled in this class often come in with different learning experiences and backgrounds. As a simple example, some may already know how to code "Hello World" in Python yet some may have never opened a ".py" (Python) file. Thus, the pre-recorded lectures and lecture review questions aim to provide a flexible model for the students. For those students who already have experience with certain portions of the concepts, they can fast forward or skip the lecture and use the lecture review questions as a refresher and review. For those students who may need additional time to master a concept, they are welcome to watch and re-watch the lecture, and work through the exercises at their convenience and own pacing. PrairieLearn, as a platform, provides the ability to generate and grade unique variants of the same base question and concept. The lecture review exercises were pre-designed with a set of concepts to assess (the learning goals/concept inventory of the course), then generated with variations. As an example, if the concept to be assessed is the "print" functionality in C++ or Python, the base question may be "print 'Hello World'", but the variations may be "print [Any String]". Additionally, as we will discuss soon in the next section "In-Class Robotics Project-based Learning", the normal lecture was designed to facilitate hands-on student robotic projects and active learning supported by teaching staff. Pre-learning enables additional project work time with staff help. We did explore additional adaptations, such as coming up with review exercises and in-class activities to review the pre-learning content and check in with individual students on their learning trajectories.

Figure 2 shows the average duration students spent on PrairieLearn (the pre-recorded lecture and completing lecture review exercises). For this course, we provided 16 required pre-recorded full-length lectures (segmented into 5-10min "bite-size" lecture segments) and two additional optional supplementary videos ("helper function", etc.) over the period of a semester (typically, around 14 to 16 weeks). As shown, the student workload on the pre-learning is, on average, 30min to 1hr per week.

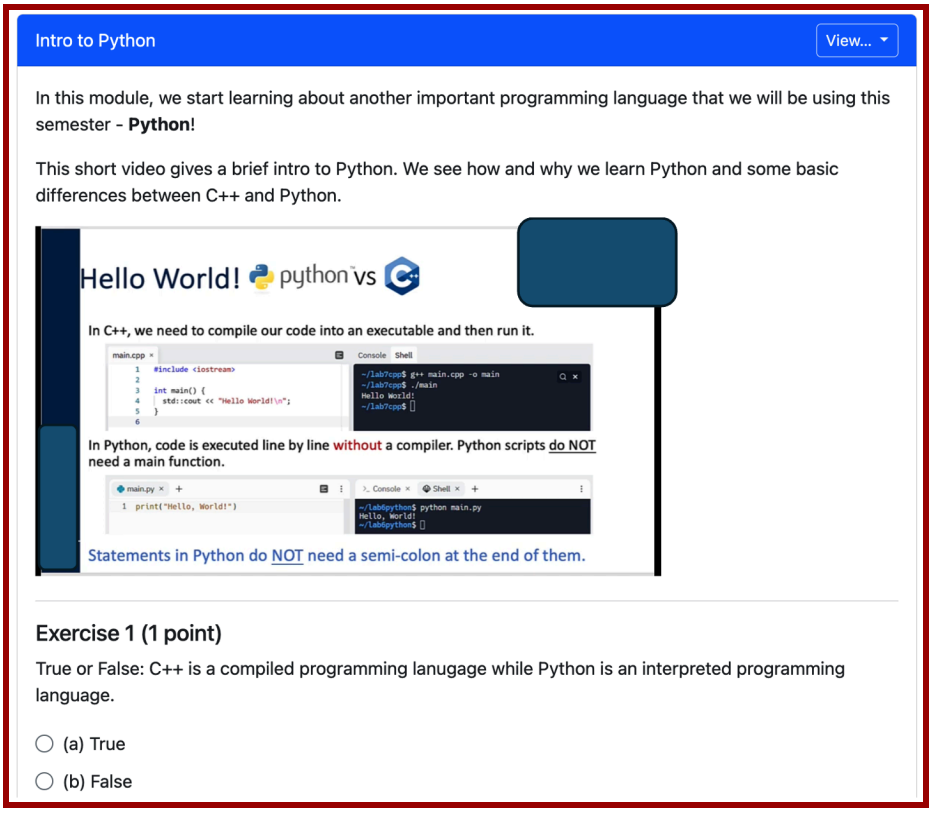

Figure 1: Screenshot of the PrairieLearn homework assignment (lecture review question) example. The example given is from an introduction to Python lecture, and the testing concept is C++ is typically a compiled language, while Python is usually an interpreted language. Typically, the student view shows the title of the lecture segment, a brief description, a lecture video or visuals, and some exercises. The instructor and school logo was redacted.

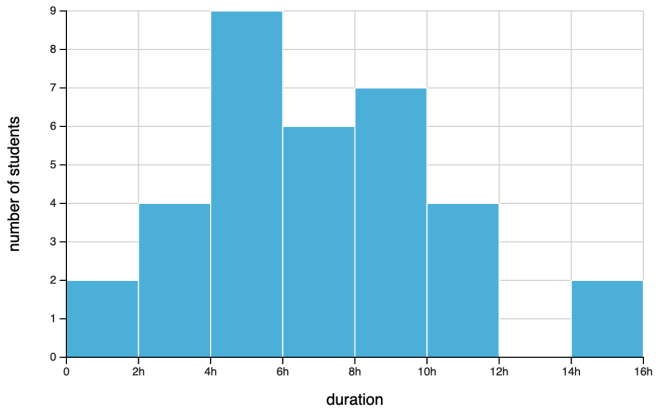

Figure 2: Duration students spent on PrairieLearn (the pre-recorded lecture and completing lecture review exercises).

In-Class Robotics Project-based Learning

After completing the pre-learning, students work hands-on on real robotic projects and activities in class. This way, students are able to utilize the concepts in real-world robotic applications and obtain personalized support from the instructional team. Some example robotic projects include programming a pocket calculator (practicing operators and programming basics), implementing a finite state machine (state transitions, iterations and branching), implementing bug navigation, obstacle avoidance, and path planning on a real-world two-wheeled robot, and using machine learning algorithms to analyze and recognize images and objects. The robots we use in this class are equipped with camera and LiDAR sensors, and part of the students' activity include collecting, loading, and processing sensor data, implementing graph search algorithms, printing out the robot pose (location and orientation) in real time, and driving the robot autonomously. In this case, students have a "use case" to apply the programming concepts. For example, students learned "what is a print statement" and "how to write print in C++ [std::cout] and Python [print]" in pre-learning, but in class they transitioned to "where can we use this print statement in achieving what we want the robot to do" and how to fit the code within the context of the rest of the project code (e.g., the students may choose to print out the current robot pose to help debug path planners, for example). Figure 3 shows an example student project using the wheeled robot autonomously navigating a maze environment. The students apply C++ and Python knowledge to collect camera and LiDAR data and implement navigation and path planning algorithms.

Current student feedback came from the anonymous university-wide teaching evaluations at midterm and end-of-the-semester. Students answer open-ended questions including "Which aspects of this course were most valuable?" "Which aspects of this course were least valuable?" "Please comment on the quality of the course as a whole." "How can [Course Instructors] improve the teaching of this course?" We received positive qualitative feedback from the class on the project-based learning aspect, such as "honestly most of the projects did well supplementing my learning", "Lots of lab time to work on the robots", "I liked the parts about learning how to code and how to do all of the coding aspects", and "I think that the most valuable aspect of the course was the fact that we used the robot to get hands on learning throughout the whole course". Interestingly, there were also additional comments on "I thought learning git hub and repository things were very helpful" [version control software]. Note that the git knowledge is not necessarily part of the programming learning objective, but is a common knowledge required for future higher-level courses and by programmers who are developing software collaboratively. The hands-on projects not only provide a platform to practice the programming languages themselves, but also offer an opportunity to introduce relevant tools such as git version control, which we hope helps the students in their future studies and career.

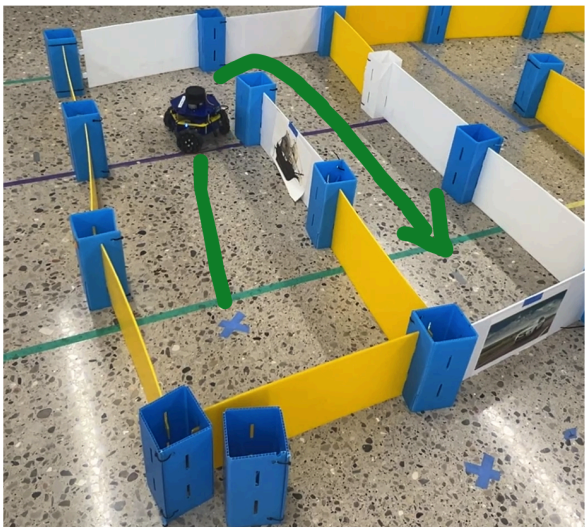

Figure 3: An example of the student project in the wheeled robot autonomously navigating a maze environment. The students apply C++ and Python knowledge to collect camera and LiDAR data and implement navigation and path planning algorithms.

Approach 2: Concept Mastery and Assessment

Frequent mastery based assessments were designed to assess students' knowledge mastery. In addition to PrairieLearn lecture review questions and general participation, the assessment of the course mainly consists of two parts, a set of frequent, low-stakes, mastery based quizzes and hands-on robotic projects.

Table 1 shows a concept inventory of five quizzes throughout the semester. The quizzes were in-person and designed to be low-stake - Each quiz only takes up 4 points (~3.6%) of the overall grades in the semester. The concepts tested on each quiz are available to students before each quiz and closely follow the learning objectives of the course and the course content design. As shown, each quiz assesses both the programming concepts as well as robotic and AI (machine learning) concepts and algorithms. In some cases, bonus points/extra credit questions are provided as an advanced challenge, as well as provides an opportunity for students to earn additional points. The quizzes are cumulative based on the fundamental concept (for example, Quiz 1 will use basic C++ syntax concepts in Quiz 0), but each quiz focuses on assessing unique concepts in programming language and algorithms. Quiz 0 to Quiz 3 are in C++ and Quiz 4 is in Python. Note that the in-person quizzes include both deterministic question types (e.g., multiple choice with a definite correct answer) and code implementation questions (e.g., the students' implementations were graded as correct as long as it followed the instructions and achieved the desired outcome - this may lead to multiple versions of correct codes).

Table 1: A concept inventory of the five quizzes (Quiz 0 - Quiz 4) throughout the semester.

	Concept Inventory	**Max score**
Quiz 0	C++ fundamentals: ● Reading and writing basic C++ code (syntax) ● Operators and variables ● Precedence ● Branching and iterations ● Pre- and Post- increment Basic Robot control: ● Bang-bang control ● Basic PID (P-) feedback control Basic robot use: ● Basic robot coordinate system (right hand rule) ● Polar and cartesian coordinate frames ● Basic git commands for version control/repository management	4.0 + 0.5 bonus Bonus task example: ● Fibonacci Number ● Integer reverse
Quiz 1	C++ fundamentals: ● C++ functions ● C++ vectors (indexing, initialization) ● Adding and Removing (de-registering) elements Robot functions and algorithms: ● Find minimum value in a vector of LiDAR array ● Computing the angles required for robot to drive a pattern ● Wall-following algorithm (cross product)	4.0 + 0.5 bonus Bonus task example: ● Two sum
Quiz 2	C++ fundamentals: ● C++ struct ● C++ string (initialization, concatenation) ● switch-case Robot functions and algorithms: ● Finite State Machine ● Bug navigation algorithm (The robot moves towards the goal in an environment with obstacles)	4.0 + 0.5 bonus Bonus task example: ● 2D coordinate transform (e.g., world frame to robot frame)
Quiz 3	C++ fundamentals: ● C++ queue Robot functions and algorithms:	4.0 + 0.5 bonus Bonus task example:

	• Euclidean distance on grid cell map • 4-connectivity and 8-connectivity neighbors • Nodes and graph struct • Breadth first search • Path planning • Grid search	• Vector reverse (e.g., trace a path from begin to end)
Quiz 4	Python fundamentals: • Reading and writing basic Python code (syntax) • Operators and variables • Branching and iterations Robot functions and Machine Learning algorithms: • Understanding supervised, unsupervised and reinforcement learning • Basic classifier (e.g., K Nearest Neighbors) • Basic neural network (e.g., run a pre-trained model on a test image)	4.0 + 0.7 bonus Bonus task example: • Variable swap • String manipulation

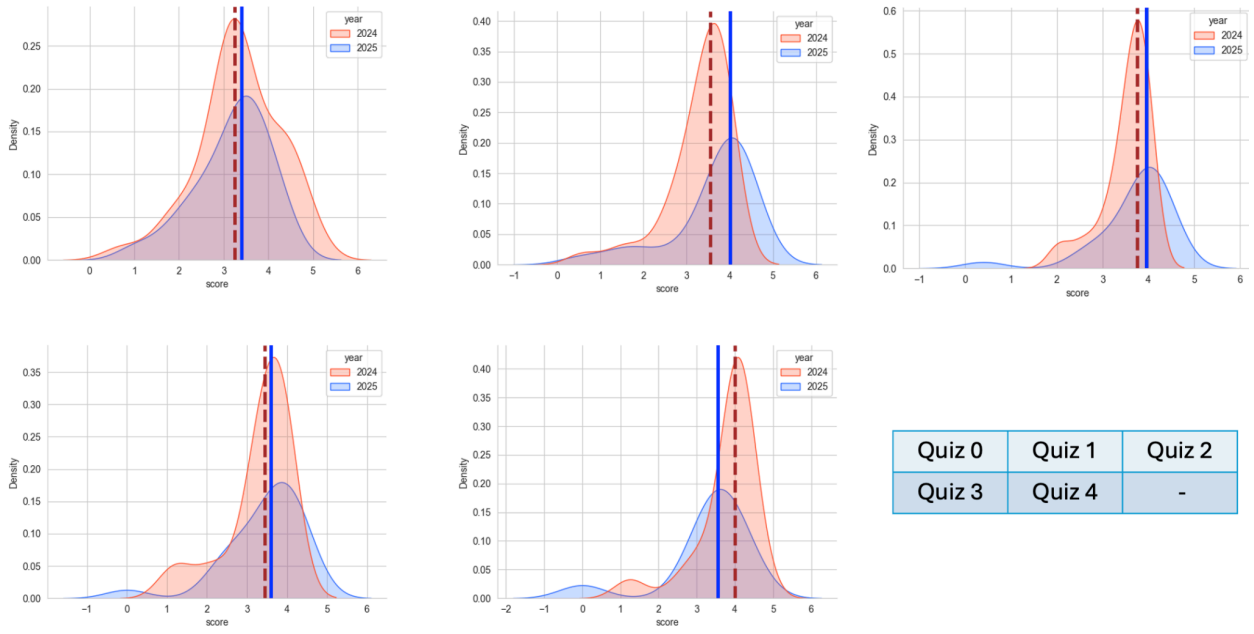

Figure 4: A kernel density estimate (KDE) plot visualizing the grade distribution of the five quizzes (Quiz 0 - Quiz 4) throughout the semester, for both 2024 and 2025 offerings. The brown dashed line is the median for Year 2024 and the blue solid line is the median for Year 2025.

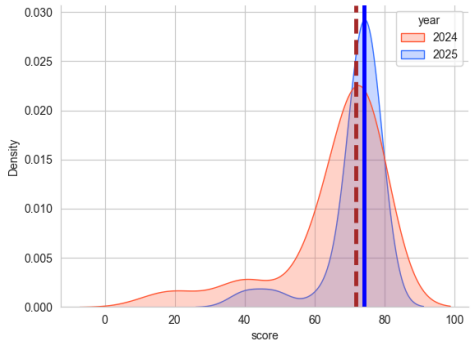

Figure 5: A kernel density estimate (KDE) plot visualizing the grade distribution of the sum of the projects throughout the semester, for both 2024 and 2025 offerings. The brown dashed line is the median for Year 2024 and the blue solid line is the median for Year 2025.

Figure 4 shows a kernel density estimate (KDE) plot visualizing the grade distribution of the five quizzes (Quiz 0 - Quiz 4) throughout the semester, for both 2024 and 2025 academic years. The brown dashed line is the median for Year 2024 and the blue solid line is the median for Year 2025. As shown, on average, the students achieve in general above 3 out of 4 (+bonus) scores, indicating generally a good mastery of the learning concepts.

The hands-on robotic projects are the main part of the course, taking up approximately 65-68% of the overall grades (72-75 points out of 110 total points). As the course is an introduction to programming and AI algorithms, the projects aim to evaluate the programming and algorithm mastery. Students submit their implementation codes as well as a project demo video to show their working robotic system. As an example, for an autonomous navigation project, students submit a video of their robot driving in a maze from a marked “start” point to a “goal” point, similar to those shown in Figure 3. Unit tests were developed to evaluate the codes, and the projects were graded as full points if both the code unit testing and the project demo videos satisfy all required instructions. Figure 5 shows a kernel density estimate (KDE) plot visualizing the grade distribution of the total score across five projects throughout the semester, for both 2024 and 2025 academic years. The brown dashed line is the median for Year 2024 and the blue solid line is the median for Year 2025. As shown, on average, the students achieved in general above 70 out of a total of 75 points possible, indicating generally a successful implementation of the learning concepts. We, as an instructional team, observed that if the students were actively engaged and continued to be present during in-class time as well as sought office hour help if needed, the students were in general capable of completing the required tasks and implementations, regardless of their initial background or experience (some may take longer time, but in general achievable before final project due dates). When asked (anonymously, as part of midterm evaluation) about the projects, the students reported “The course seems to be laid out very logically, with each project leading into the next one. Each project was still a challenge, but never to the extent where anything felt impossible to complete in the time given”, and

"Working with the robot and programming it to do basic practical activities has really advanced my programming knowledge on how a robot can navigate. I especially relied on the PrairieLearn videos which were a life saver in my opinion."

Approach 3: Peer Collaboration and Peer Assisted Teaching

A valuable component and goal of the course is providing a collaborative platform for the students in the course to work within the class, as well as with the instructional team. Part of the goal for the semi-flipped classroom (Approach 1), as discussed above, was to increase the engagement and time used to complete hands-on projects during actual class time. In class, students were placed in teams and encouraged to work with their partner to complete the robotic projects in class. Online question and answer platforms, such as Piazza, and live office hour queues, were used to provide a platform for students to ask questions and receive support (anonymously, if they so wished). Additionally, interactive presentation tools and online platforms, such as live polls and in-class activities (e.g., exercise variations based on the pre-learning videos), are incorporated during class sessions to promote engagement, gauge student understanding, and encourage participation in real time.

For an engineering course, graduate students were typically employed as part of the instructional staff (e.g., as graduate student instructor). For this course, we intentionally increased the number of undergraduate students as part of the peer assisted teaching team (for 2024 offering, 9 undergraduate instructional aides; and for 2025 offering, 5 undergraduate instructional aides[2].) The majority of the undergraduate instructional aides are students who have taken the course, or are familiar with the programming and robotic algorithms used. The undergraduate staff range from sophomore (one year after taking this introductory course) to seniors (soon-to-graduate). This way, the students taking the course gain opportunities to not only ask conceptual questions with the instructional team, but also can network with peers who may be a few years ahead. The undergraduate instructional staff, on the other hand, can also offer their experience and expertise in course materials, project help, as well as life and career as a university student (e.g., course enrollment, student project team selection, internship, jobs, and graduate school applications, etc.) There was overwhelmingly positive feedback from the class that strongly favored the undergraduate instructional aides (IA), such as "The IAs…were so valuable I CANNOT understate that fact," "... Also the IAs are life– savers."

The undergraduate instructional aides also were capable of providing support for the faculty and the entire teaching team, in addition to the students enrolled in the course. As the IAs have taken

[2] Note some IAs may be part-time. A "full-time" undergraduate instructional aide typically can work up to 10 hours per week. A "part-time" undergraduate IA can request to work less than 10 hours. For example, a "0.5"-appointment IA would work up to 5 hours per week. IAs are compensated based on hours worked. The specific IA schedules are generally determined based on the funding available, class needs, and IA availability.

the course before, they can bring in fresh ideas and provide feedback about future iterations of the course and further improving the course materials. The course, given the high robotic project content, naturally relies heavily on the robotic hardware system. The IAs were able to help tremendously with robot testing, making sure batteries are charged, swapping out broken motors, and in general helping out with maintaining course hardware demands. As for grading, for the in-person assessments, the IAs also play an important role for providing customized feedback for each student together with the course faculty.

Approach 4: Accessibility and Beyond

Accessible Course Materials

The course aims to serve as a resource and reference for educators seeking programming and algorithms-related concept materials. The course materials are openly available online at the university website, and it has been offered as part of the distributed teaching collaborative at other programs and universities. Some improvements have been made on the digital course materials, including adding and revising the pre-learning videos with transcript texts, upgrading the course materials with better contrasts, and so on, so that they may be more easily accessible to students and educators.

Instructional Team Office Hours

In connection with the peer-assisted teaching (Approach 3) discussed above, the instructional team, including faculty, graduate and undergraduate instructional aides, all provide office hours for all students as needed. An editable office hour calendar is available to all instructional staff as well as all students so that the office hours and project help remains flexible, accessible, and efficient. From the students' perspective, the office hour calendar can be easily added to their school/personal account, and thus it is easily viewable and enables them to easily select the times/days that would work best. From the student instructional staff point of view, the student instructional aides have their individual schedule and other classes to attend to, and an editable office hour calendar enables flexibility in case something changes or updates. An office hour queue with both in person and remote videoconferencing options was used to make sure students are able to receive one-on-one meeting and project help. Multiple student instructional staff can also coordinate and keep track of who in the class may still need help. In the recent two years that we offered it, all students at least attended office hours once, and many have attended multiple office hours, especially before major project deadlines and getting hands-on help to fully integrate the algorithms on the robot.

Real-World Application and Impact

To support in-class robotics project-based learning (Approach 1) and assessment (Approach 2), real-world applications were discussed and encouraged in the course for the students to demonstrate their learning and mastery of the programming concepts and robotic system. As an

example, at the end of the semester when we discuss fundamental AI and machine learning algorithms, we encourage the students to create their own unique customized datasets for inference. Figure 6 shows an example of the student testing their robot in a maze-like environment, with their own hand-written digits stuck on the wall. Imaging the scenario of a robot "tour guide" navigating in the maze, where it automatically detects the handwritten digits (e.g., going from "3" to "0") and uses these as navigation waypoints to then autonomously navigate inside the maze. This way, students enjoyed the excitement and novelty of incorporating their own unique datasets and methods in the project in a "real-world" scenario. Other such real-world examples include collecting their own digits and images for running basic machine learning detection and segmentation algorithms (e.g., YOLO [13]). Another advantage of using real-world examples and encouraging students to customize their datasets and inputs is that this provides an opportunity for discussing real-world AI and algorithmic challenges, such as how to address bias in data collection and labeling, how an existing algorithm performs at inference when testing on a different test data, etc. It moves the course to beyond "simply coding", and towards responsible, thoughtful engagement with modern-day AI technology.

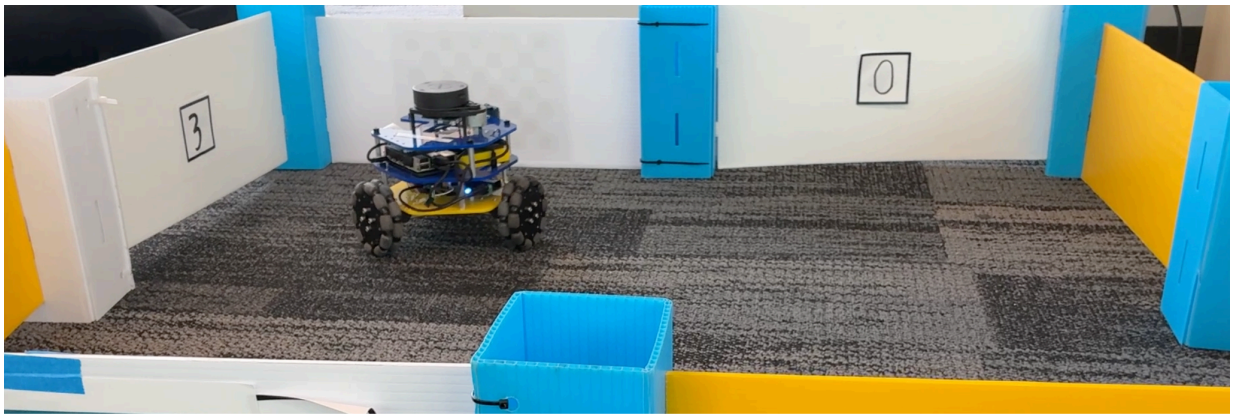

Figure 6: An illustration of a project maze set up by the students at the end of the semester, completing the robot navigation and hand-written digit recognition tasks.

Reflection, Discussion, and Limitations

The strategies discussed above have been successfully incorporated in the past two offerings and in general, the class feedback has been positive. The course received an average 4.8 out of 5.0 rating for "Overall, this was an excellent course" and positive student comments include a sense of well-preparedness for more advanced programming courses in the undergraduate curriculum sequence (e.g., "I learned much more than I would ever need to! EECS 2XX [second-year programming and data structure course] is going to be much easier thanks to this!")

We acknowledge a few limitations in the work. First, we acknowledge that some of the methods may have already been implemented (in some way) in prior work. For example, regarding Approach 1, the concept of active learning and flipped classrooms is not entirely new [9]-[12].

However, this course implements this revised flipped classroom model using the PrairieLearn platform, which was developed in the most recent decade with continued state-of-the-art features that may benefit future curriculum design. The authors also have not seen many other prior works that incorporate robotic projects in first-year programming courses, which we believe is a unique aspect of this work. Second, regarding Approach 2, there has been discussion on the disadvantage of quizzes of using memorization as a way of measuring learning gains. Instead, alternative methods such as collaborative quizzes or oral assessments may be given [14] [15]. We argue that the written ("pencil-and-paper") quizzes are a fitting tool for concept mastery assessments (and an effective one, as evidenced by the student performance data shown earlier). This addresses the concern for teaching programming in the age of generative AI by directly assessing student programming abilities on paper without AI assistants or vibe coding. We will clarify that for this course, our quizzes are not focused on "memorization". Instead, the problems on the quizzes are practical applications of the programming concept. For example, the quiz will include an example of "please fill in the C++ code for moving the robotic arm", and a skeleton template code is provided with specific "TO-DOs" for students to fill in. The concepts tested may be C++ vectors (for storing robotic arm joint angles) or branching (if- conditions). The students will need to really understand how the programming works (not just memorizing the language syntax, but to choose the correct variables and operators to implement the programming "applications"). We argue this type of quizzes is similar to what students may encounter realistically in future studies and career. Students need to know how and when to use the programming knowledge to complete the tasks given stakeholder requirements and specifications. For future offerings, we are open to explore alternative assessment strategies, such as oral exams, peer programming, whiteboard coding, etc. Besides, in the robotic projects, as the project codes have components that are specifically tailored to this specific mobile robot, even if students tried to use generative AI tools to help debug, it does not return a good answer for the programming assignments and parameter tuning. By using real robots, students are mandated to work hands-on with both the hardware and the software, learning programming concepts and computational thinking in the process. Additionally, current data in this study were from anonymous teaching evaluations. If resources allow, individualized questions and surveys may be conducted to further analyze student response and impact of the proposed approaches, such as major retention, follow-on course performance, etc.

We would also like to recognize a few complexities in the process. For Approach 1, the PrairieLearn-based pre-learning requires instructors to pre-record videos and design exercises outside of the usual lecture time, whose feasibility can be dependent on the workflow and availability of the instructors. The in-class robotics project-based learning is one of the unique innovations of this course, yet we do recognize that not all first-year programming courses may have the resources (e.g., hardware systems, debugging efforts) for an individualized robot, especially if the class size is significantly larger. Some adaptations may be functional, such as working with a simulated dataset or platform (e.g., Issac Sim [16]). There has been some

feedback about trying to debug the robot, such as “my robot was very buggy and so I spent an excessive amount of time inside and outside of class trying to debug issues outside the scope of this course. While I did learn extra skills from these issues, it was frustrating to have to do extra work to get the same results as classmates with working robots.” We recognize that real-world robots do break down sometimes, and it does take time and effort to debug as part of the educational experience. For Approach 2, the in-person frequent low-stake quizzes with hand-written coding questions are another novelty in the course that largely sidestepped the concern for AI-assisted coding in tests. Some students gave high praise to the in-person quizzes, such as “The aspect of this course that was most valuable was having to do hand–written code of the quizzes”. However, we acknowledge that it also takes time and effort to design and grade the pencil-and-paper quizzes, as well as providing customized feedback for hand-written codes. In relation to Approach 3, we were fortunate to work with a wonderful group of instructional staff that made it possible, but grading efforts may be a consideration for other courses to adopt the in-person quiz approach, given availability and size of the instructional staff. Some possible alternatives may include having whiteboard challenges and pair programming assessments in class, peer oral exams [17], blocking certain generative AI coding tools during exams so a Computer-Based Testing Facility (CBTF) (e.g., [18]) may be used, etc. As for Approach 4, the open source materials and real-world examples are easily scalable as the class sizes grow.

A potential avenue for future work includes examining multi-year offerings of this course with possible pre-post surveys to better investigate the instructional strategies effectiveness. A more detailed breakdown and analysis (e.g., [19]) of post-course student development (e.g., majors declared, future performance, success and retention in higher-level programming and engineering courses) may be evaluated if such longitudinal data are available.

Conclusion

This paper presents four approaches in a first-year algorithms and programming course, including incorporating real-world robotic projects, in-person mastery-based assessments, leveraging peer-assisted learning and teaching, and flexible and accessible course materials and individualized test cases. We provide and analyze the student feedback examples and assessment data in 2024 and 2025 (most recent two academic year offerings). As the class size continues to grow and as more AI technology and tools are being developed, we expect the continued adaptation and advancement in the course materials and pedagogical method. We hope the approaches presented help provide some insights, practical recommendations, and lessons learned for instructors seeking to adapt introductory computer science and programming education to the demands and opportunities presented by advances in AI, with the goal of better preparing first-year learners for future study and careers in computational and multidisciplinary engineering fields.

References

[1] Geddis, D., & Aufderheide, B., & Colquhoun, H. W., "*Work in Progress: Project and Design-Based Introductory Engineering Course using Arduino Kits*", ASEE Virtual Annual Conference Content Access, 2020.

[2] Anastasio, D. D., & Chwatko, M., & Burkey, D. D., & McCutcheon, J. R., "*A First-year Project-based Design Course with Management Simulation and Game-based Learning*", ASEE Annual Conference & Exposition, Seattle, Washington, 2015.

[3] James-Byrnes, C. R., & Holdhusen, M. H., "*Online Delivery of a Project-based Introductory Engineering Course*", ASEE Annual Conference & Exposition, San Antonio, Texas, 2012.

[4] Tahmina, Q., "*Does Peer Mentoring Help Students be Successful in an Introductory Engineering Course?*", ASEE Annual Conference & Exposition , Tampa, Florida, 2019.

[5] Huett, K. C., & Kawulich, B. B., & Raju, P., & Sankar, C. S., "*Use of Multimedia Case Studies in an Introductory Engineering Course at Two Southeastern Universities: A Qualitative Evaluation Study*", ASEE Annual Conference & Exposition, Atlanta, Georgia, 2013.

[6] Vora, M. K., & Danahy, E. E. (2025, June), A Student Classification and Characterization Model of Generative AI Use in First-Year Engineering Design Paper presented at 2025 ASEE Annual Conference & Exposition , Montreal, Quebec, Canada . 10.18260/1-2--55398

[7] Hwang, W., & Ramirez-Salgado, A., & Kalavakonda, R. R., & Ambarwati, Y. E., & Antonenko, P., & Bhunia, S. (2025, June), WIP: Empowering First-Year Engineering Students for Career Choices through Hands-On AI Hardware Experiences Paper presented at 2025 ASEE Annual Conference & Exposition , Montreal, Quebec, Canada . 10.18260/1-2--57398

[8] Stransky, J., & Agrawal, A., & Eastman, M. (2025, June), WIP: Efficacy of Connecting Engineering and Calculus through AI Problem Generation Paper presented at 2025 ASEE Annual Conference & Exposition , Montreal, Quebec, Canada . 10.18260/1-2--57397

[9] Garrick, R. (2018, June), Flipped Classroom Video Analytics Paper presented at 2018 ASEE Annual Conference & Exposition , Salt Lake City, Utah. 10.18260/1-2--30526

[10] Ayasoufi, A., & Williams, R. (2018, June), Revising the Flipped Classroom Paper presented at 2018 ASEE Annual Conference & Exposition , Salt Lake City, Utah. 10.18260/1-2--30940

[11] Velegol, S. B., & Zappe, S. E. (2016, June), How Does a Flipped Classroom Impact Classroom Climate? Paper presented at 2016 ASEE Annual Conference & Exposition, New Orleans, Louisiana. 10.18260/p.25479

[12] Howard, A. K. T. (2019, June), Flipped Classroom – Ten Years Later Paper presented at 2019 ASEE Annual Conference & Exposition , Tampa, Florida. 10.18260/1-2--32849

[13] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A, "*You only look once: Unified, real-time object detection*," In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 779-788, 2016.

[14] Baghdadchi, S., & Schurgers, C., & Qi, H., & Alajeel, H. (2024, June), Using Oral Assessments to Improve Student Learning Gains Paper presented at 2024 ASEE Annual Conference & Exposition, Portland, Oregon. 10.18260/1-2--48238
[15] Oishi, M., & Svihla, V., & Law, V. (2017, June), Improved Learning Through Collaborative, Scenario-based Quizzes in an Undergraduate Control Theory Course Paper presented at 2017 ASEE Annual Conference & Exposition, Columbus, Ohio. 10.18260/1-2--28485
[16] NVIDIA. Isaac Sim (Version 5.1.0) [Computer software]. https://github.com/isaac-sim/IsaacSim
[17] Lubarda, M. V., & Phan, A. M., & Ghazinejad, M., & Delson, N., & Baghdadchi, S., & Schurgers, C., & Kim, M., & Relaford-Doyle, J., & Sandoval, C. L., & Qi, H., "*Peer oral exams: A learner-centered authentic assessment approach scalable to large classe*s", ASEE Annual Conference & Exposition, Baltimore, Maryland, June 2023.
[18] Craig Zilles, "*Computer-Based Testing Facilities as a Means for Enabling Better Assessment Pedagogy*," In Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2 (SIGCSE 2023). Association for Computing Machinery, New York, NY, USA, pp. 1258, 2023.
[19] Nazempour, R., & Darabi, H., & Revelo, R. A., & Nelson, P. C., & Felder, A. E., & Ozevin, D., & Abiade, J. T., "*Implementation of an Introductory Engineering Course and its Impact on Students' Academic Success and Retention*", ASEE Virtual Annual Conference Content Access, 2020.
[20] Quezada, B., & Moore, T., & Douglas, K., & Johnston, A., & Haluschak, E., & Bidna, G., "*Analyzing First-Year Students' Motivation and Exposure Towards an Advanced Topic During an Introductory Coding Course*", ASEE Annual Conference & Exposition, Minneapolis, MN., Aug. 2022.

APPENDIX

Learner Profile

Programming Experience

For the most recent two academic years (2024 Fall, 2025 Fall), the course included 35 and 23 students, respectively. This course is offered through the Robotics department and open to all engineering students. Figure A1 and Figure A2 shows the first-year learner profile regarding their self-rated prior programming experience before taking the course. As shown, the majority of students taking the class have some exposure/experience with some form of programming, with only 27.3% (2024, 9 students) and 16.7% (2025, 3 students) reported having little or no programming experience. For those who have some programming experience, 61.5% reported gaining these experiences from completing a pre-college introductory programming course and/or taking Advance Placement courses in high school, and 15.3% reported informal or

non-coursework experience to gain familiarity with coding (e.g., from practicing by themselves at home, personal projects, gaming kit such as Lego sets, project teams and competitions such as FIRST tech challenge, etc.).

Compared to Figure A1 (2024), Figure A2 (2025) shows less people rating themselves as having little or no programming experience, while the number of people who rated "significant programming experience" has increased. We also observed the students wrote more occurrences of affirmative language in describing programming experience/abilities, using phrases such as "Lots of experience with Python", "quite a bit of experience in Java", "Yes, I have coded in HTML, CSS, JavaScript, Java, and Python." This could be due to the increased exposure and opportunity for the incoming first year students in the age of AI, where there are more and more tutorials, courses, project teams (challenges, competitions) readily available to students who are interested in exploring programming and AI concepts before enrolling in a college-level introductory course. A small percentage of students (12.1%) also reported they have taken data structure, data analytics, and data science courses that had some programming, which shows the wide variety of resources now available to students for learning programming.

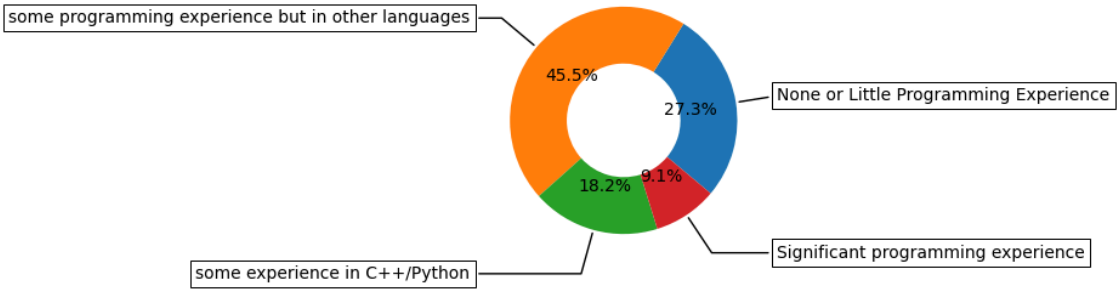

Figure A1: Pie chart showing the first-year learner programming experience profile for 2024. Best viewed in color.

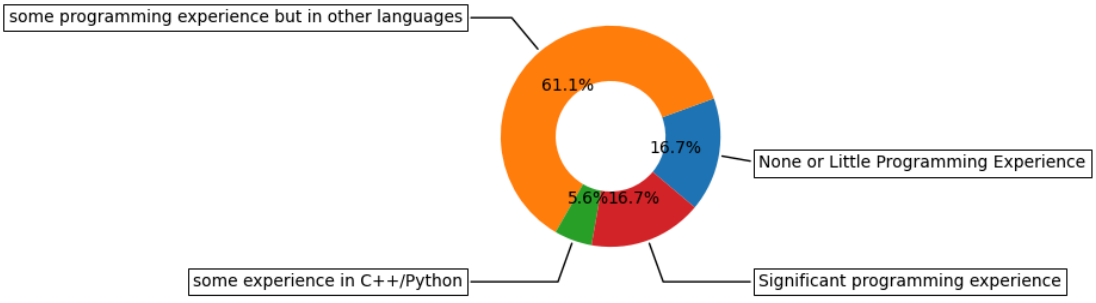

Figure A2: Pie chart showing the first-year learner programming experience profile for 2025. Best viewed in color.

Programming Languages

The students also reported familiarity with a wide range of different programming languages. Figure A3 shows the ranking of the languages students have stated they have some experience in. As shown, over 25% of the students stated that they have at least some experience with Python. As a high-level, general-purpose programming language, Python has been widely used in many computational and robotic applications due to its simplicity, readability, and versatility. This result shows that Python is gaining popularity in pre-college programming experience as well (currently ranked 1st). Java ranked closely second at around 20%. Many reported learning Java as part of the high school curriculum (e.g., AP CSA - Advanced Placement Computer Science A and CSP - Computer Science Principles classes), or in FIRST tech challenge (FTC) and FIRST robotic competition (FRC) project teams. C/C++ ranked the third, showing that C/C++ is still highly relevant in the current age despite its long history and focus on "traditional" software engineering (lower-level control, less readable but highly customizable). Students also reported familiarity with web development (12.9% using HTML/CSS and 11.1% using Javascript). A few students reported using C#, MATLAB, and other programming languages/interfaces such as Processing programming language, Unix, Swift, and LEGO Mindstorms.

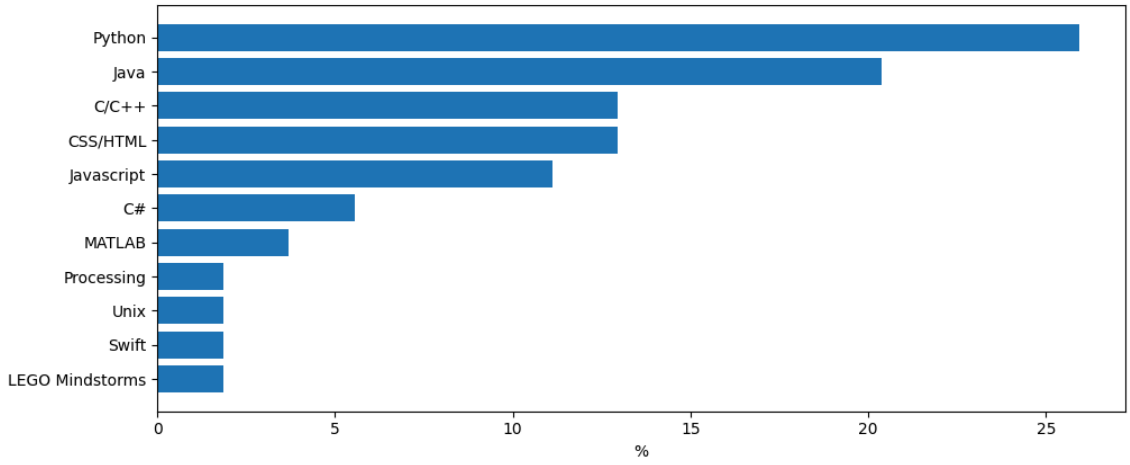

Figure A3: Bar chart showing the first-year learner programming experience (prior to enrolling in the course), ranked by programming languages.

Motivation and Course Outlook

The class expressed various motivations and outlooks for taking the course. Multiple students expressed the interest and desire to work with robots, such as "Programming interests me because it is what brings robots to life in the real world", "I am very interested in how to work with robots, in how to translate my code physically," "I'm interested because I love building robots from my experience on my high school robotics team so now I want to learn how to do all the processes involved in constructing the whole thing," and "What I'm interested in learning about programming is having the robot go through a maze without any bugs and being able to do most code in memory." From those who did not have a strong prior experience with

programming, there were some apprehension and yearning for more help and support, for example “I struggle a bit to understand coding,” “I have very little experience and a bit overwhelmed just like everyone else so some one on one help would be very helpful to me,” and “I can learn but at the start I'm gonna sound and look stupid by how many questions I ask [with skeleton and cry emojis as part of the message]”. One student stated that in addition to the interest in learning other programming languages, they would like to be “exploring more about AI, since it’s such a popular topic nowadays.” A prior work in exploring student motivation in microelectronics [20] also found students mentioning their motivation for learning the subject topic (in their case, microelectronics; in our case, robotics and AI) as either a career path of interest, or mentioning their awareness of cutting-edge technology and their role in society.

Thus, to support the students with varying programming experiences and motivation to enroll in the course, we implemented several approaches, including a flipped classroom strategy to encourage, frequent low-stake tests (quizzes) to assess knowledge mastery, incorporating peer collaboration and peer assisted teaching as part of the instructional team formation, and providing accessible, open-source tools and materials.

Course Instructor Profile

For the most recent two academic years (2024 Fall, 2025 Fall), one tenured faculty and one instructional faculty have been the primary instructors for this course, together with multiple graduate student instructors and undergraduate instructional aides. In a typical semester, the teaching staff includes 1-2 faculty members plus 1 graduate student instructor and 5-12 undergraduate instructional aides (IAs, note some may be part-time). The teaching faculty typically have experience in computer programming, mobile robotics, artificial intelligence and its algorithmic foundations, machine and deep learning, and robotic perception and reasoning methods. The graduate student instructor and undergraduate instructional aides are typically from students who have either taken the course before and/or have conducted research or projects in related areas (and, thus, familiar with the robotic platform, programming, and artificial intelligence algorithms). For the most recent two academic years (2024 Fall, 2025 Fall), around 60% of undergraduates IAs are recurring (have served as IA for this course in previous year) and the remaining are new IAs (never served IAs for the course before, but typically have taken the course. Usually, they were from outstanding students from the previous year’s cohort).

Course Structure and Logistics

This course was designed as a robotics-friendly pathway into computing disciplines and engineering more broadly. The course was offered through the Robotics department but open to all engineering students. This course serves as one of the prerequisites for EECS 2XX, a

second-year programming and Introductory Data Structures course which, in turn, serves as an enforced prerequisites for many of the 3XX and 4XX (junior- and senior-level) courses in many engineering departments. In other words, EECS 2XX serves as a “gateway” for many upper-level computationally intensive courses, and this course serves as one of the first-year introductory programming courses that leads up to EECS 2XX. This course also serves as part of the Engineering First Year Core Requirement for introductory programming for multiple engineering departments. The learning objectives of this course includes learning computer programming for the C++ and Python languages. This course also introduces programming pragmatics (e.g., version control and file navigation, as part of working on real robotic systems), as well as building an understanding of artificial intelligence and its algorithmic foundations and computational thinking. Table 1 “Concept Inventory” column lists the concepts covered in the course in more detail.

The course has two lectures per week (1.5hr per lecture) and one 2-hr lab session per week. As mentioned as part of the course design, we offer the course using a “semi-flipped classroom” format, where during lectures, the instructors typically take about 20-30min for a lecture view or an in-class activity (e.g., a programming exercise), and the remaining lecture time is for students to work on their robotic and programming projects. The lab time is also project time where staff will be available to answer questions. Sometimes, during lecture or lab, instructors may call a stop to student work, reiterate any common mistakes or reminders, and students continue their hands-on work. Multiple office hours (typically, at least 1-2hr office hours per week for each full IA) are offered outside of class for students who wish to continue working on their projects or have any questions.