Learning Python through AI-Assisted Code Generation and Classroom Discussion: Developing Computational Thinking and Problem-Solving Skills among BSc Data Science Students

Dr. Paul Joseph Lawrance
Assistant Professor, Department of Computer Science, Mary Matha College of Arts and Science, Periyakulam, Tamil Nadu, India

***Abstract*—The increasing availability of generative artificial intelligence (AI) tools has created new opportunities for teaching and learning programming. However, using AI-generated code effectively requires learners to develop a strong understanding of programming concepts, logical reasoning, and problem-solving skills. This study examines an AI-assisted code-generation and classroom-discussion-based instructional approach to teaching Python programming among 11 second-year Bachelor of Science (BSc) Data Science and Analytics students at Mary Matha Arts and Science College, Periyakulam, Tamil Nadu, India. Adopting a mixed-method research design, the study investigated students' learning outcomes, particularly in Python programming knowledge, computational thinking, problem-solving skills, and practical programming competence, while also exploring their perceptions of the institutional teaching-learning facilities. Data were collected through classroom observations, student interactions, six practical assessments, class tests, internal examinations, model examinations, practical examinations, and an end-of-semester feedback questionnaire. The instructional approach integrated AI-powered tools, including ChatGPT and Google Gemini, with teacher-guided classroom discussions, code interpretation, program execution, and debugging activities. The findings revealed an overall mean score of 4.48 out of 5 across 66 observations from six AI-assisted practical assessments, indicating a high level of correctness in program outputs. Furthermore, all students scored above 80% in both the model and main practical examinations, demonstrating their ability to apply Python programming concepts to practical tasks. However, theoretical examination performance varied, with only five students achieving above 80%, highlighting the need for continued support in conceptual understanding, independent reasoning, and theoretical knowledge. Student feedback was unanimously positive, with 100% of respondents strongly agreeing that AI-assisted code generation, classroom discussions, individualized attention, and the available teaching-learning facilities supported their learning. The study highlights the educational potential of integrating generative AI with interactive teaching strategies and hands-on programming practice. It emphasizes that AI-generated code should serve as a learning resource rather than a substitute for programming knowledge, independent reasoning, and critical evaluation.**

***Index Terms*—Artificial Intelligence in Education; AI-Assisted Programming; Python Programming; Computational Thinking; Problem-Solving Skills; Generative AI; Classroom Discussion; Practical Programming Competence; Higher Education.**

I. INTRODUCTION

In the modern era of artificial intelligence (AI), code can be generated within seconds using AI-powered tools. The paper argues that educators must act quickly to shape the opportunities and challenges of AI code generation, and it flags over-reliance and the need for critical evaluation of generated code(Becker et al., 2023). However, understanding programming concepts remains essential for evaluating whether AI-generated code is accurate, secure, efficient, and appropriate for a given problem. Programming knowledge enables learners to critically examine generated solutions, identify logical errors, assess their suitability, and make informed decisions about their implementation. Although AI can support code

generation and computational tasks, it cannot replace the human qualities of wisdom, ethical judgement, empathy, and moral discernment required to evaluate the broader implications of technological solutions. Learning programming gives us the power to solve complex problems, automate boring tasks, and makes us to build digital tools from scratch. It teaches us to break a complex problem into smaller steps. Someone who understands programming can give AI better instructions, spot errors, modify generated code, and combine different AI-generated components. In Data Science and AI, programming is fundamental for Machine learning, Data analytics, Generative AI, Natural language processing, Computer vision, and Robotics and IoT.

The study was carried out on the teaching and learning of Python programming for eleven second-year students of the Bachelor of Computer Science in Data Science and Analytics at Mary Matha Arts and Science College, Periyakulam, Tamil Nadu, India. The institution is a self-financing, co-educational institution affiliated with Madurai Kamaraj University, Madurai, Tamil Nadu, India. Python is an open-source, excellent, easy, high-level, interpreted, interactive, object-oriented, and reliable language that uses English-like words. Python is one of the most consistent and clear computer languages. It has some special features that work better than adapting styles from other languages, and it balances ease of learning, ease of use, and expressive power(Lubanovic, 2023). The strength of Python can be understood from the fact that this programming language is the most preferred language of companies like Nokia, Google, YouTube, and even NASA for its easy syntax (Thareja, 2024).

The Python Programming course, offered as a core subject in the BSc Data Science and Analytics programme, was designed to develop students' foundational and applied programming competencies. The learning objectives focused on developing an understanding of fundamental Python programming concepts, applying object-oriented programming (OOP) principles, acquiring knowledge of programming best practices, and exploring the application of Python to economic concepts, particularly demand and supply, cost analysis, and profit maximization. Upon successful completion of the course, students were expected to demonstrate an understanding of Python syntax and semantics; identify problems and develop solutions using appropriate Python programming techniques; select suitable programming constructs for problem-solving; analyse Python language concepts to develop efficient solutions; and design, implement, and test Python programs to address given problems. Accordingly, the course emphasised a practice-oriented approach that integrated conceptual learning, code writing, program execution, and debugging through the development of practical Python applications. This approach was intended to provide students with in-depth knowledge of the language and progressively strengthen their confidence, independent problem-solving abilities, and practical programming competence.

The institutional computer laboratory was equipped with systems featuring Intel Core i5 processors operating at 2.90 GHz and running the Windows 11 operating system. All systems were pre-installed with the required educational software and the Veyon Master application. Veyon is free and open-source classroom management software designed to monitor and control computers across multiple platforms. It supports teaching and learning in digital environments by enabling instructors or system administrators to monitor, control, and interact with multiple client computers in real time through a centralized dashboard (Veyon, 2026). The application also facilitates screen sharing, allowing the instructor's screen to be broadcast to all student computers while simultaneously enabling the monitoring or locking of individual computers.

The institution provides a well-equipped library that supports teaching and learning. Students and faculty members have access to a wide range of print resources and may borrow books for a specified loan period. In addition, the library offers internet-enabled computer facilities, allowing users to access digital resources and retrieve relevant academic information. To further enhance the teaching–learning process, the institution has recently established a smart classroom equipped with Information and Communication Technology (ICT) facilities. The classroom features a 52-inch Android-based interactive smart display that enables faculty members to deliver lectures, demonstrate digital content, and conduct interactive learning activities, thereby fostering greater student engagement and participation.

The study focused on two research questions designed based on the subject learning objectives and the institutional facilities.

1. How does an AI-assisted, code-generation and classroom-discussion-based instructional approach influence the development of computational thinking, problem-solving skills, Python programming knowledge, and practical programming competence among second-year BSc Data Science students?
2. What are the perceptions of second-year undergraduate students regarding the effectiveness of the institutional learning environment (traditional classrooms, computer laboratories, smart classroom facilities, library resources, and internet access) in supporting their learning of Python programming?

The research objectives

1. To examine students' learning outcomes following the implementation of an AI-assisted, code-generation and classroom-discussion-based instructional approach, with particular emphasis on Python programming knowledge, computational thinking, problem-solving skills, and practical programming competence.
2. To explore the students' perceptions of the available teaching-learning facilities, which fits well with a mixed-method research design through surveys and interviews/focus group discussions.

This study adopted a mixed-method approach to examine the teaching and learning of Python programming through AI-assisted code generation and classroom discussions, while also exploring students' perceptions of the teaching-learning facilities provided by the institution. Quantitative and qualitative data were collected through classroom observations, student interactions, and assessments of learning, including classroom activities, class tests, internal examinations, and the model examination. Flexible instructional strategies aligned with the course learning objectives were designed and implemented during regular teaching sessions to address students' varying learning needs. At the end of the semester, a questionnaire was administered to evaluate students' perceptions of the AI-assisted code-generation and classroom-discussion approach, as well as the adequacy of the available institutional teaching-learning facilities. The findings indicate positive learning outcomes, particularly in practical programming performance, and suggest that integrating AI-assisted code generation with guided classroom discussions can support the development of computational thinking, problem-solving, and practical programming competence among students.

II. METHODOLOGY

The initial study focused on a self-introduction session; it was observed that many students experienced difficulty in expressing their ideas in English, although they demonstrated proficiency in their respective mother tongues. To gain insight into their perceptions of education, an inquiry-based approach was adopted by posing the question, "Why is education important?" Students provided a wide range of responses, including earning money, securing employment, acquiring knowledge, developing self-confidence, enhancing communication skills, cultivating hard work, and improving overall competencies. To deepen students' understanding of the significance of education, each of these responses was systematically discussed in the classroom and linked to broader educational goals. These discussions enabled students to reflect on the multifaceted role of education in personal, social, and professional development.

Furthermore, it was observed that the students belonged to two linguistic backgrounds, namely Kerala and Tamil Nadu, where Malayalam and Tamil are the predominant mother tongues. This linguistic diversity created valuable opportunities to facilitate understanding through the use of students' native languages, particularly when explaining complex technical concepts in English. In this context, the strategic use of the mother tongue served as an effective pedagogical tool to enhance comprehension and engagement. Supporting this approach, (Yadav, 2014) argues that the target language should remain the primary medium of instruction whenever possible, while the learners' mother tongue may be used selectively when genuinely necessary to facilitate understanding. This perspective aligns with Nelson Mandela's well-known observation: "If you talk to a man in a language he understands, that goes to his head. If you talk to him in his own language, that goes to his heart." The use of the mother tongue can

therefore support students in constructing accurate meanings and developing a deeper understanding of educational concepts.

To evaluate students' learning potential and presentation skills, a classroom activity was conducted in which students were assigned to collect information on the Python programming language, including its history, working principles, advantages, disadvantages, and limitations. A two-hour computer laboratory session was provided to enable students to browse online resources, gather relevant information, and gain initial hands-on exposure. Students were subsequently encouraged to continue their learning and practice independently outside the classroom using AI tools. On the following day, each student delivered an oral presentation on the assigned topic.

Student presentation performance was assessed across five dimensions—content, preparation, language, creativity, and enthusiasm—using a five-point rating scale: Excellent (5), Good (4), Satisfactory (3), Needs Improvement (2), and Poor (1). The performance outcomes of the students are summarized in Table 1 below.

Students	Content	Preparation	Language	Creativity	Enthusiasm
Student 1	2	2	2	2	3
Student 2	2	2	2	1	3
Student 3	2	2	2	1	2
Student 4	2	3	2	1	3
Student 5	3	2	3	1	3
Student 6	4	4	2	3	3
Student 7	4	4	3	3	3
Student 8	3	3	2	1	2
Student 9	2	2	3	2	3
Student 10	2	2	2	1	2
Student 11	2	2	2	1	3

Table 1: Students' Presentation Performance

Dimension	Mean Score	Interpretation
Content	2.55	Needs Improvement–Satisfactory
Preparation	2.55	Needs Improvement–Satisfactory
Language	2.27	Needs Improvement
Creativity	1.55	Poor–Needs Improvement
Enthusiasm	2.73	Needs Improvement–Satisfactory
Overall	2.33	Needs Improvement

Table 2: Students' Presentation Performance in Mean Score

The ratings of the 11 students indicate an overall mean score of 2.33/5, suggesting that the students' presentation skills were generally at the "Needs Improvement" to "Satisfactory" level.

The highest mean score was observed for enthusiasm (M = 2.73), indicating that students generally demonstrated a reasonable level of interest and willingness to participate in the presentation activity. Content and preparation had identical mean scores (M = 2.55), suggesting that students demonstrated a basic understanding of the subject matter and prepared for their presentations, although further development is needed to reach a satisfactory or good level. Language received a lower mean score (M = 2.27), indicating difficulties in verbal expression and effective communication. Creativity received the lowest score (M = 1.55), with most students rated at the "Poor" level, suggesting limited originality in presentation approaches and a need for greater encouragement to use creative, engaging methods.

In terms of individual performance, Student 6 and 7 demonstrated relatively stronger performance, particularly in content and preparation, with ratings of 4 (Good) in both areas and 3 (Satisfactory) in creativity. However, no student received a rating of 5 (Excellent) in any of the assessed dimensions, indicating that there is considerable scope for further improvement. The relatively low ratings in language and creativity are consistent with the classroom observation that several students found it difficult to

present confidently without referring to their notes. Consequently, allowing students to use notebooks during the presentations served as a supportive pedagogical strategy to reduce presentation anxiety and facilitate communication, while also providing students with an opportunity to gradually develop confidence and presentation fluency.

Overall, the findings suggest that the presentation activity provided students with an initial opportunity to develop communication, preparation, content organization, creativity, and confidence. The results particularly highlight the need for continued instructional support in oral communication, spontaneous presentation, creative expression, and confidence-building activities. Repeated presentation opportunities, peer feedback, guided speaking activities, and gradual reduction of dependence on notes may help students progress from the "Needs Improvement" level toward Satisfactory, Good, and eventually Excellent performance.

As students learnt C++ in the first-year program, they gained experience with structural building blocks and data management concepts. This was discussed in the classroom, and students were asked to review Python programming structure, data types, data structures, input/output, and conditional and loop statements in the computer lab during two-hour practical periods using the website "www.w3schools.com". This allows students to learn the basics of Python by using a Python tutorial option, and it even helps them to understand the code with an example by clicking the "Try it Yourself " button. It was instructed that the learning in the computer lab should be recorded in their notebooks.

The first execution of a Python program in an IDLE Python interface was explained in the computer lab using Veyon software. It allows the screen to be shared with students and helped them to write their first Python program by accepting students' details (name, registration number, and course registered) as input and printing the same in a formatted output. All the students were able to get the formatted output on their systems. According to the class timetable, students have two combined classes as lab sessions in a week, which allows them to practice Python programs that they have learnt in the classroom sessions.

After completing every topic, a 5-mark activity was conducted to test their understanding of Python syntax and hands-on abilities. The following Table 3 shows P1, P2, P3, P4, P5, and P6, which are the performance of students in writing Python programs.

Table 3 presents the performance of 11 students across six assessment points (P1–P6) in the form of a class test out of 5 marks. Questions were based on writing a program to find a value for the given formula. The descriptive findings indicate a substantial improvement in student performance over the assessment period. The mean score increased progressively from 0.18 (SD = 0.40) at P1 to 1.00 (SD = 0.77) at P2 and subsequently reached 3.18 (SD = 0.98) at P3. Although a decline was observed at P4 (M = 2.36, SD = 0.67) and P5 (M = 1.64, SD = 1.21), performance increased again at P6, with a mean score of 3.18 (SD = 0.98). The median score similarly increased from 0 at P1 to 3 at both P3 and P6. The range of scores at P6 (2–5) also indicates that all students achieved at least a score of 2 in the final assessment, compared with a range of 0–1 at baseline. Overall, the results demonstrate a marked improvement in student performance between the initial and final assessment points, although fluctuations across the intermediate assessments suggest that the learning path was not strictly linear.

Students	P1	P2	P3	P4	P5	P6	E1	E2	E3	E4	E5	E6
Student 1	0	1	3	2	1	3	5	3	4	5	4	5
Student 2	1	2	5	3	2	3	5	5	5	5	5	5
Student 3	0	0	2	2	1	2	5	3	2	2	4	5
Student 4	1	2	5	3	3	3	5	5	5	5	4	5
Student 5	0	1	3	2	2	3	5	4	4	4	4	5
Student 6	0	0	3	2	1	3	5	3	5	3	4	5
Student 7	0	1	3	3	2	3	5	5	5	5	4	5
Student 8	0	1	3	3	2	3	5	5	5	5	5	5
Student 9	0	2	3	3	4	5	5	5	5	5	5	5

Student 10	0	1	2	1	0	5	5	4	5	4	4	5
Student 11	0	0	3	2	0	2	5	3	3	3	4	5

Table 3: Students' Classroom and Practical Activities

Table 3 presents the performance of 11 students across six assessment points (P1–P6) in the form of a class test out of 5 marks. Questions were based on writing a program to find a value for the given formula. The descriptive findings indicate a substantial improvement in student performance over the assessment period. The mean score increased progressively from 0.18 (SD = 0.40) at P1 to 1.00 (SD = 0.77) at P2 and subsequently reached 3.18 (SD = 0.98) at P3. Although a decline was observed at P4 (M = 2.36, SD = 0.67) and P5 (M = 1.64, SD = 1.21), performance increased again at P6, with a mean score of 3.18 (SD = 0.98). The median score similarly increased from 0 at P1 to 3 at both P3 and P6. The range of scores at P6 (2–5) also indicates that all students achieved at least a score of 2 in the final assessment, compared with a range of 0–1 at baseline. Overall, the results demonstrate a marked improvement in student performance between the initial and final assessment points, although fluctuations across the intermediate assessments suggest that the learning path was not strictly linear.

During classroom instruction and practical sessions, students demonstrated interactive engagement and performed tasks with reasonable proficiency. However, classroom activities revealed difficulties in recalling Python syntax and applying programming logic independently. These observations highlighted the need for differentiated instructional strategies that accommodate students' varying proficiency levels and support the progressive development of programming and logical reasoning skills. In response, an AI-assisted, discussion-based learning strategy was introduced. Students were permitted to use their mobile phones to generate Python code for given programming tasks using generative AI tools, such as ChatGPT and Google Gemini. With students' participation, the generated code was reproduced on the blackboard and examined through step-by-step explanations and discussions. According to (Zhang & Zhou, 2025) the collaborative use of student grouping, tailored resource recommendations, and dynamic feedback mechanisms has greatly increased students' programming abilities and learning motivation. In a controlled study with 69 novices learning Python, access to an AI code generator improved progression and correctness and reduced errors and completion time. The authors also note that over-reliance is a possible drawback, which fits your "learning resource, not a substitute" argument (Kazemitabaar et al., 2023). Students were encouraged to identify the purpose of individual statements, understand the underlying programming logic, and interpret the expected output before executing the code using the Programiz online Python compiler. Student performance was assessed on a five-point scale based on the correctness of the program output. This instructional strategy was implemented across six practical assessments (E1–E6), and the results are presented in Table 3.

The results presented in Table 3 indicate an overall mean score of 4.48 out of 5 across the 66 student-assessment observations, reflecting generally strong performance in producing correct Python program outputs. In the first assessment (E1), all 11 students obtained the maximum score of 5. The mean scores for the subsequent assessments were 4.09 in E2, 4.36 in E3, 4.18 in E4, and 4.27 in E5, indicating some variation in performance across the assessment sequence. In the final assessment (E6), all students achieved the maximum score of 5. Individual results revealed that students 2, 8, and 9 consistently scored 5 across all six assessments, while students 4 and 7 maintained scores of 4 or 5 throughout. Students 3 and 11 demonstrated improvement in the later assessments, progressing from lower scores in the earlier assessments to 5 in E6. Overall, the findings suggest a favourable pattern of performance following the implementation of the AI-assisted, discussion-based instructional strategy, particularly in students' ability to produce correct program outputs. However, as the assessments primarily measured output correctness, the results should not be interpreted as definitive evidence of improved programming logic or independent problem-solving skills. Furthermore, without a comparison group or additional measures, the observed improvements cannot be attributed solely to the instructional strategy.

A class test carrying a maximum of 25 marks was conducted to assess students' understanding of fundamental Python programming concepts through

practical problems involving the calculation of the areas of mathematical shapes. The assessment focused on students' ability to declare and use variables and appropriate data types, such as integers and floating-point numbers, to represent dimensions including radius, length, and base. It also evaluated their proficiency in using input () and print () statements to obtain user input and display computational results, respectively, and applying arithmetic operators, including multiplication (*), exponentiation (**), and division (/), to implement mathematical formulas. Furthermore, the test assessed students' ability to use conditional statements (if, elif, and else) to select the required shape and execute the corresponding calculation, define reusable functions using the def keyword to structure area-calculation procedures, and import Python's built-in math module, particularly math.pi, for accurate calculations involving circles. Overall, the assessment was designed to evaluate students' understanding of basic Python syntax and their ability to apply fundamental programming constructs to solve mathematical problems.

Students	Class Test Out of 25 marks	Internal Exam 1 Out of 50 marks	Internal Exam 2 Out of 50 marks	Model Practical Exam Out of 75 marks	Main Practical Exam Out of 75 marks	Model Theory Exam Out of 75 marks
Student 1	18	25	32	60	65	30
Student 2	18	30	24	75	75	60
Student 3	3	19	23	66	62	19
Student 4	25	48	50	75	75	69
Student 5	20	32	31	64	75	33
Student 6	4	24	23	65	65	23
Student 7	21	36	50	75	75	69
Student 8	22	45	50	71	75	61
Student 9	20	48	41	68	75	70
Student 10	20	15	30	61	62	18
Student 11	15	36	28	72	75	16

Table 4: Students' Performance in Class Test, Internal Exams, and Model Exams

The class test results in the above Table 4 indicate varying levels of student achievement in fundamental Python programming concepts, with scores ranging from 3 to 25 out of a maximum of 25 marks. The 11 students obtained a mean score of 16.91 (SD = 7.10), equivalent to 67.64% of the total marks. Five students (45.45%) scored 20 marks or above, demonstrating a relatively good understanding of basic Python syntax, variables and data types, input and output operations, arithmetic operators, conditional statements, function definition, and the use of the math module. One student achieved the maximum score of 25, indicating strong performance in applying programming constructs to mathematical problems. Three students scored between 15 and 18 marks, suggesting a developing understanding of the concepts assessed, while the remaining three students scored below 15, including two students who obtained fewer than 5 marks. These lower scores indicate that some students may require additional instructional support and guided practice to strengthen their understanding of Python syntax, logical reasoning, and the implementation of mathematical formulas through code. The complexity of programs used during their practice could be tailored more accurately to their level of proficiency (Jayal et al., 2011). Overall, although the majority of students demonstrated a satisfactory to good level of achievement, the variation in scores highlights differences in programming proficiency and the need for differentiated instruction, scaffolded practical exercises, and targeted remedial activities to support students with lower levels of attainment.

In accordance with the course curriculum, students were scheduled to undertake two internal examinations and one model examination. The first internal examination covered Units 1 and 2, the second covered Units 3 and 4, and the model examination assessed learning across all five units. The first and second internal examinations were conducted to evaluate students' academic performance and progress. Each examination was designed for a maximum of 50 marks and comprised nine multiple-choice questions, three 7-mark questions with internal

choices, and two 10-mark questions to be answered from a choice of three. Before the first internal examination, observations of students' presentations, classroom activities, and class-test performance indicated a need for continued instructional support in oral communication, spontaneous presentation, creative expression, and confidence-building. Although classroom activities revealed a favourable pattern of performance following the implementation of an AI-assisted, discussion-based instructional strategy, particularly in students' ability to produce correct program outputs, the class-test results highlighted difficulties among some students in applying Python syntax, logical reasoning, and mathematical formulas in programming tasks. In response to these observations, scaffolded practical exercises were introduced to reinforce programming concepts and support progressive skill development. Furthermore, classroom interactions and discussions were conducted entirely in English to provide students with sustained opportunities to practise oral communication, articulate their reasoning, and develop confidence in expressing their understanding of programming concepts.

The results of the two internal examinations revealed variations in students' academic performance and progress across the assessed units. In the first internal examination, students' scores ranged from 15 to 48 out of 50, with a mean score of 32.55. In the second internal examination, scores ranged from 23 to 50, and the mean increased to 34.73, representing an improvement of 2.18 marks. Six of the eleven students improved their scores, whereas five recorded lower marks in the second examination. The most notable improvements were observed in Student 10, whose score increased from 15 to 30, and Student 7, whose score increased from 36 to 50. Student 1 and Student 8 also demonstrated improvements, with scores increasing from 25 to 32 and from 45 to 50, respectively. Furthermore, three students—Student 4, Student 7, and Student 8—achieved the maximum score of 50 in the second examination. However, the decline in scores among five students indicates that learning progress was not uniform across the cohort. These findings suggest a modest overall improvement in academic performance, alongside individual differences in learning outcomes. Consequently, continued formative assessment, targeted feedback, and scaffolded practical activities may be beneficial in addressing students' varying levels of understanding and strengthening their ability to apply Python programming concepts

The Python practical exam is designed to test students' Problem-solving Logic and Control Flow, Ability to manipulate built-in data structures, Modular programming and functions, and File handling and exception management. The institution organizes a model practical to prepare students for the main practical organized by Madurai Kamaraj University. Both practicals follow the same procedures; out of 14 practical questions, two questions are randomly selected by the students. The questions are designed according to the university norms.

The Model and Main Practical Examinations assessed students' practical programming competencies through a set of 14 Python programming tasks covering the fundamental concepts and applications addressed in the course curriculum. The tasks required students to write Python programs to accept student details; calculate simple interest, compound interest, and equated monthly instalments (EMI); determine grades based on marks using conditional statements (if-elif-else); calculate the sum of digits and reverse a number using a while loop; display odd and even numbers between 1 and 100; perform basic arithmetic operations; calculate the factorial of a number using recursion; identify the largest and smallest values and calculate the average of elements in an array; determine whether a given string is a palindrome; create and utilise a user-defined calculator module; perform matrix addition and multiplication using nested lists; identify the maximum and minimum values in a tuple; calculate word frequencies in a sentence using a dictionary; and count the number of lines, words, and characters in a text file.

These tasks were designed to evaluate students' ability to apply Python programming concepts, construct logical solutions, implement appropriate programming constructs, and execute programs to obtain the expected outputs.

Before participating in the practical examinations, students were required to submit a practical record book documenting each exercise under the following headings: Aim, Algorithm, Python Program, Expected Output, and Result obtained after program execution. Students developed their Python programs with help from AI-based code-generation tools. During classroom sessions, they discussed each program and

explained its execution step by step, enabling the instructor to assess their understanding of program logic and the underlying theoretical concepts. Subsequently, in the computer laboratory, students executed and debugged the programs in Python IDLE to verify the actual outputs against their expected results. This instructional approach encouraged students to move beyond the generation of code towards understanding, testing, and evaluating its functionality. By combining AI-assisted code generation, classroom discussion, and hands-on programming practice, the approach provided opportunities for students to develop computational thinking through systematic problem decomposition, logical reasoning, and iterative debugging. Furthermore, the reflective discussion and verification process encouraged students to critically examine AI-generated solutions, question underlying assumptions, evaluate the correctness of program logic, and consider alternative approaches to problem-solving.

The results of the model and final practical examinations indicated that all students scored above 80% of the total marks, as presented in Table 4. During the practical examinations, students were required to randomly select two questions from a set of 14 and document each solution by presenting the Aim, Algorithm, Python Program, Expected Output, and Result obtained after program execution. Students who achieved the maximum score of 75 out of 75 successfully executed both programs, obtained the expected outputs, and answered all oral questions posed by the external examiner. These results suggest that students were able to interpret the given problems, recall relevant Python concepts, apply their knowledge to develop programs, execute the code, and identify and correct errors through debugging. This process reflects computational thinking, particularly the systematic decomposition of problems into manageable steps to develop appropriate solutions (Wing, 2006). The successful completion of these tasks demonstrates the application of programming knowledge and practical problem-solving skills. The ability to recall Python concepts reflects the remembering level, while interpreting problems, developing programs, executing code, and debugging solutions correspond to higher cognitive processes, including Understanding, Applying, Analyzing, and Evaluating (Anderson & Krathwohl, 2002). Thus, the practical examination results provide evidence of students' ability to transfer theoretical knowledge into practical programming tasks.

In the final model theory examination, worth 75 marks, only five students (Students 2, 4, 7, 8, and 9) scored above 80% of the total marks as shown in Table 3. The study reports that GenAI can give novices complete solutions and proposes ways to scaffold the novice GenAI experience. It also points to teaching metacognitive skills explicitly (Prather et al., 2024). An analysis of their answer scripts indicated a sound understanding of Python concepts and strong problem-solving abilities, as evidenced by their ability to explain concepts in their own words and develop original examples. These responses suggest the development of computational and critical thinking skills. Furthermore, the students demonstrated effective learning habits by preparing and maintaining well-organised learning materials and making productive use of the time allocated for note preparation during scheduled classroom, laboratory, and library sessions. Their active participation and sustained engagement in classroom teaching, computer laboratory activities, and smart classroom sessions further reflected their commitment to learning. They also interacted confidently with the instructor and sought clarification whenever necessary, indicating a proactive approach to addressing learning difficulties and achieving academic goals. In addition, all five students-maintained attendance rates above 85%, which may have contributed to their consistent participation in instructional activities and continuity of learning. Taken together, these observations suggest an association between regular attendance, active engagement, and effective learning practices, enabling students to perform better in Python programming;

Students 1 and 5 scored 30 and 33 marks, respectively, out of 75 in the model examination. An analysis of their answer scripts revealed that several responses were irrelevant to the questions or demonstrated limited conceptual alignment with the Python programming content. Some answers appeared to draw on C++ programming concepts encountered in previous semesters, suggesting difficulties in distinguishing and applying the concepts specific to Python. During the distribution of the evaluated answer scripts, the instructor asked both students about their examination performance. They responded, "It is a model exam; we will focus more on

the main exam." This response suggested that they perceived the model examination as less consequential and may not have prepared for it with sufficient seriousness. Their performance indicated a need for continued scaffolded practical exercises, targeted revision, and structured guidance to strengthen their understanding of Python concepts and progressively develop their programming skills.

Student 10 was absent for about the first six weeks of the semester and had an attendance rate of only 54%. Despite this limited attendance, classroom observations indicated that she participated actively and responded appropriately to questions. She also scored 20 out of 25 in a class test involving mathematical programming problems, demonstrating a capacity to apply relevant concepts in a structured task. However, her performance in the model examination was considerably lower, with a score of 18 out of 75. An examination of her answer script suggested that insufficient preparation for the model examination may have contributed to this result. She scored 7 out of 10 in the multiple-choice section, indicating that she could recall or recognise several relevant concepts when provided with answer options. The contrast between her performance in the class test and the model examination suggests a possible gap between her ability to solve focused programming tasks and her ability to demonstrate broader theoretical knowledge under examination conditions. Her limited attendance may also have restricted her exposure to instructional activities.

The model examination results of Students 3, 6, and 11 revealed specific learning challenges that required individual attention, structured guidance, and additional academic support. Although these students occasionally displayed playful behaviour during classroom sessions, they became more attentive and engaged when provided with firm and structured instructions. Student 3 was particularly regular and active in classroom interactions and demonstrated the ability to respond correctly to questions. However, the student's academic performance did not reflect this level of classroom engagement, as evidenced by a score of 3 out of 25 in the initial class test, 19 out of 50 in the first internal examination, and 19 out of 75 in the model examination. Further observation revealed that the student attended college without carrying personal textbooks or notebooks and relied on borrowed rough notes from a peer to follow classroom sessions. The model examination answer script also contained only limited responses, suggesting a gap between classroom participation and independent preparation. Student 6 scored 23 out of 75 in the model examination. Although the student demonstrated the ability to solve mathematical problems independently, difficulties were observed in translating mathematical logic into appropriate Python programming statements. The answer script showed a relatively better performance in the multiple-choice section, with 7 out of 10 marks, indicating the ability to recall or recognise relevant concepts when provided with answer options. However, the student attempted only a limited number of descriptive and programming-oriented questions carrying 7 and 10 marks, respectively. Student 11 scored 16 out of 75 in the model examination. An examination of the answer script indicated that the student's handwriting was difficult to read, which may have affected the examiner's ability to interpret the responses and consequently influenced the marks awarded. The student's performance across the class test, internal examinations, and model examination also indicated the need for further academic support. Overall, the performance of these three students highlighted the importance of developing greater responsibility towards independent learning, regular preparation, and active engagement with programming activities. Accordingly, continued scaffolded practical exercises, targeted revision of fundamental Python concepts, individualised feedback, and structured classroom guidance were considered necessary to address their respective learning difficulties. These interventions were intended to strengthen conceptual understanding, improve the ability to translate problem-solving logic into Python code, and progressively develop independent programming competence.

Feedback collected at the end of the semester through the Google Forms questionnaire revealed that 100% of students strongly agreed with six statements about the effectiveness of the AI-assisted Python instructional approach and the adequacy of the available teaching-learning facilities. Students unanimously reported that AI-generated Python code, followed by classroom discussion, helped them understand Python programming concepts more clearly. They also strongly agreed that mobile phones and access to AI tools were adequate for participating in AI-assisted programming activities. Furthermore, all students

indicated that classroom and computer laboratory discussions helped them identify logical errors and understand alternative approaches to solving programming problems. The students also unanimously reported that individualized attention increased their confidence in writing Python programs independently. Regarding practical learning facilities, 100% strongly agreed that the available computers and the IDLE Python 3.14 (64-bit) programming environment supported their practical learning needs. Similarly, all students considered the classroom facilities, including mobile phones and blackboard-based learning resources, helpful for following Python demonstrations and participating in discussions. These findings indicate a uniformly positive student perception of the instructional approach, individualised support, and the technological and classroom resources provided.

III. CONCLUSION

This study examined the implementation of an AI-assisted code-generation and classroom-discussion-based instructional approach to teaching Python programming to second-year BSc Data Science and Analytics students. The findings indicate that integrating AI tools such as ChatGPT and Google Gemini with teacher-guided discussions, practical exercises, program execution, and debugging provided opportunities for students to engage with Python concepts and develop practical programming skills.

The six AI-assisted practical assessments recorded an overall mean score of 4.48 out of 5 across 66 student-assessment observations, indicating a high level of correctness in program outputs. Furthermore, all students secured more than 80% in the model and main practical examinations, demonstrating their ability to apply Python concepts to practical programming tasks. However, performance in the model theory examination varied, with only five students scoring above 80%. This difference highlights the importance of strengthening conceptual understanding, independent reasoning, and the ability to express programming knowledge beyond successful program execution. The findings also emphasise the need for continued support through structured practice, revision, individual attention, and targeted feedback.

Students' feedback was consistently positive, with all respondents strongly agreeing that AI-assisted code generation, classroom discussions, individual attention, and the available teaching-learning facilities supported their learning. These responses suggest that the combination of AI tools and interactive instructional practices was perceived as beneficial to the learning experience.

Overall, the study highlights the potential of AI-assisted learning when integrated with meaningful classroom interaction and hands-on programming activities. AI-generated code should be treated as a learning resource rather than a substitute for programming knowledge or independent problem-solving. Students need to understand, examine, modify, test, and evaluate generated code to develop genuine programming competence and use AI responsibly.

The findings should be interpreted in light of the study's limitations, particularly its small sample of 11 students, single-institution setting, and absence of a comparison group. Further research involving larger student cohorts, multiple institutions, and comparative or longitudinal designs is recommended to examine the sustained impact of AI-assisted programming instruction on independent computational thinking and problem-solving skills.

REFERENCES

[1] L. W. Anderson and D. R. Krathwohl, *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. New York, NY, USA: Longman, 2001. [Online]. Available: https://quincycollege.edu/wp-content/uploads/Anderson-and-Krathwohl_Revised-Blooms-Taxonomy.pdf

[2] B. Becker, P. Denny, J. Finnie-Ansley, A. L. Reilly, J. Prather, and E. A. Santos, "Programming is hard—or at least it used to be: Educational opportunities and challenges of AI code generation," in *Proc. ACM Conf. Int. Comput. Educ. Res. (ICER)*, 2023. doi: 10.1145/3545945.3569759.

[3] Jayal, S. Lauria, A. Tucker, and S. Swift, "Python for teaching introductory programming: A quantitative evaluation," *Innovation in Teaching and Learning in Information and Computer Sciences*, vol. 10,

no. 1, pp. 86–90, 2011, doi: 10.11120/ital.2011.10010086.

[4] M. Kazemitabaar, J. Chow, C. T. Ma, B. J. Ericson, D. Weintrop, and T. Grossman, "Studying the effect of AI code generators on supporting novice learners in introductory programming," in *Proc. 2023 CHI Conf. Human Factors in Computing Systems*, 2023, doi: 10.1145/3544548.3580919.

[5] B. Lubanovic, *Introducing Python: Modern Computing in Simple Packages*, 3rd ed. Sebastopol, CA, USA: O'Reilly Media, 2023.

[6] J. Prather, B. Reeves, J. Leinonen, S. MacNeil, and A. Randrianasolo, "The widening gap: The benefits and harms of generative AI for novice programmers," in *Proc. 2024 ACM Conf. Int. Comput. Educ. Res. (ICER)*, 2024, doi: 10.1145/3632620.3671116.

[7] R. Thareja, *Python Programming Using Problem Solving Approach*. Oxford, UK: Oxford University Press, 2024.

[8] Veyon, "Home," 2026. [Online]. Available: https://veyon.io/en/

[9] J. M. Wing, "Computational thinking," *Communications of the ACM*, vol. 49, no. 3, pp. 33–35, Mar. 2006, doi: 10.1145/1118178.1118215.

[10] M. K. Yadav, "Role of mother tongue in second language learning," *International Journal of Research*, vol. 1, no. 11, pp. 572–582, 2014.

[11] W. Zhang and Y. Zhou, "Research on differentiated teaching model of Python programming based on learning data analysis," *Journal of Computer Technology and Electronic Research*, vol. 1, no. 3, 2025, doi: 10.70767/jcter. v1i3.405.