TEACHING JAVA AND PYTHON PROGRAMMING: A COMPARATIVE PEDAGOGICAL FRAMEWORK FOR DEVELOPING COMPUTATIONAL THINKING AND PROBLEM-SOLVING SKILLS

Authors:
Teacher at G'ijduvon District Technical College No. 4
Nematova Nafisa Fazliddinovna

ABSTRACT

Programming education is a fundamental component of computer science curricula because it develops computational thinking, logical reasoning, algorithmic problem-solving, and software development skills. However, novice learners frequently experience difficulties understanding programming syntax, control structures, data types, functions, and object-oriented concepts. Java and Python represent two widely used programming languages with different syntactic characteristics, execution models, and pedagogical affordances. This article presents a conceptual review of approaches to teaching Java and Python and proposes an integrated pedagogical framework for their use in programming education. The study synthesizes selected peer-reviewed research on introductory programming, programming language selection, novice learning, and curriculum design. The analysis identifies five interconnected instructional dimensions: foundational algorithmic thinking, progressive syntax acquisition, comparative programming practice, object-oriented design, and project-based assessment. The proposed framework recommends introducing fundamental computational concepts through accessible programming tasks and subsequently developing more structured software engineering competencies through Java and Python. Future research should investigate its effectiveness through controlled or quasi-experimental studies, programming performance assessments, and longitudinal evaluation of students' problem-solving skills.

Keywords: programming education, Java, Python, computational thinking, algorithmic problem-solving, computer science education, object-oriented programming, project-based learning.

INTRODUCTION

Programming education has become an essential element of contemporary computer science curricula. As digital technologies continue to influence industry, scientific research, business, and education, learners increasingly need the ability to design algorithms, develop software, process information, and solve problems through computational methods.

Programming is not limited to learning the syntax of a particular language. It requires students to formulate problems, identify relevant information, develop logical procedures, translate solutions into executable instructions, test program behavior, and correct errors. These activities involve computational thinking, abstraction, decomposition, algorithmic reasoning, and systematic problem-solving.

Despite its importance, learning to program remains challenging for many beginners. Students may struggle to understand variables, conditional statements, loops, functions, data structures, and the relationship between program code and program execution. They may also experience difficulties distinguishing syntax errors from logical errors or applying previously learned concepts to unfamiliar problems.

The choice of programming language can influence the way these concepts are introduced. Python provides relatively concise syntax and supports rapid experimentation, making it suitable for illustrating many fundamental programming concepts. Java, in contrast, offers explicit type declarations, a structured object-oriented programming model, and a well-defined class-based approach that can support the teaching of software design and abstraction.

Research comparing introductory programming approaches has explored whether beginning with Python before transitioning to Java may support the acquisition of fundamental programming concepts (Jayal et al., 2011). Lo et al. (2015) also examined considerations involved in choosing between Java and Python for introductory programming instruction. More recent research has investigated changes in university programming curricula and the role of Python and Java in successive introductory courses (Kaila et al., 2024; Gomes et al., 2026).

However, a curriculum that includes both Python and Java needs a coherent instructional strategy. Without careful sequencing, students may confuse language-specific syntax or memorize examples without developing transferable programming knowledge.

This article addresses this challenge by proposing a comparative pedagogical framework for teaching Java and Python within a course entitled Working with Programming Languages (Java and Python).

1.1. RESEARCH PROBLEM

The central problem concerns how to organize Java and Python instruction so that learners develop foundational programming knowledge while progressively acquiring more advanced software development competencies.

Teaching only one language may limit opportunities to compare alternative approaches to representing and solving computational problems. Conversely, introducing two languages without a structured progression may increase cognitive demands and create unnecessary confusion for beginners.

A coherent teaching model should therefore distinguish transferable programming concepts from language-specific features and provide opportunities for students to implement, compare, test, and explain their solutions.

1.2. AIM AND OBJECTIVES

The aim of this article is to develop a conceptual pedagogical framework for integrating Java and Python instruction in computer science education.

The specific objectives are:

1. To examine selected research on introductory programming and programming language selection.
2. To compare the pedagogical characteristics of Java and Python.
3. To identify instructional principles that support the progressive development of programming competence.
4. To propose a structured model integrating algorithmic thinking, comparative coding, object-oriented programming, and project-based learning.
5. To formulate recommendations and research propositions for future empirical evaluation.

2. LITERATURE REVIEW AND THEORETICAL BACKGROUND

2.1. PROGRAMMING EDUCATION AND COMPUTATIONAL THINKING

Computational thinking involves approaching problems through processes such as decomposition, abstraction, algorithm design, pattern recognition, and systematic evaluation of solutions.

Programming education provides a practical environment in which learners can develop these processes. When students create a program, they must translate a problem into precise instructions, consider alternative solution strategies, and evaluate whether the resulting code satisfies the requirements.

For example, a task requiring the calculation of a student's average score can be approached through several stages:

1. Identify the required input values.
2. Determine the calculation procedure.
3. Define the conditions for valid input.
4. Implement the algorithm.
5. Test the program with representative examples.
6. Correct errors and explain the solution.

This approach emphasizes the reasoning process rather than the memorization of programming commands.

Introductory programming courses should therefore provide opportunities for learners to formulate problems independently, explain algorithmic decisions, and test solutions systematically.

The selection of a programming language should support these objectives without unnecessarily obscuring the underlying concepts.

2.2. CHALLENGES FACED BY NOVICE PROGRAMMERS

Novice programmers must learn several interrelated concepts simultaneously. These include the syntax of the language, program execution, variable management, control flow, data representation, and debugging.

Difficulties may arise when students concentrate excessively on syntax and fail to understand the logic of their programs. For example, a student may know how to write a for loop but may not understand why the loop terminates or how the loop variable changes.

Similarly, learners may produce syntactically correct programs that generate incorrect results because their algorithms are incomplete or based on incorrect assumptions.

Research on novice use of Java has investigated how beginners use different language features and the challenges associated with learning the language (Novice Use of the Java Programming Language, 2023). Such work highlights the importance of understanding students' actual programming behavior rather than relying exclusively on final assignment scores.

From a pedagogical perspective, these difficulties suggest the need for short exercises, explicit explanations, worked examples, immediate feedback, and progressively more demanding tasks.

2.3. PYTHON AS AN INTRODUCTORY PROGRAMMING LANGUAGE

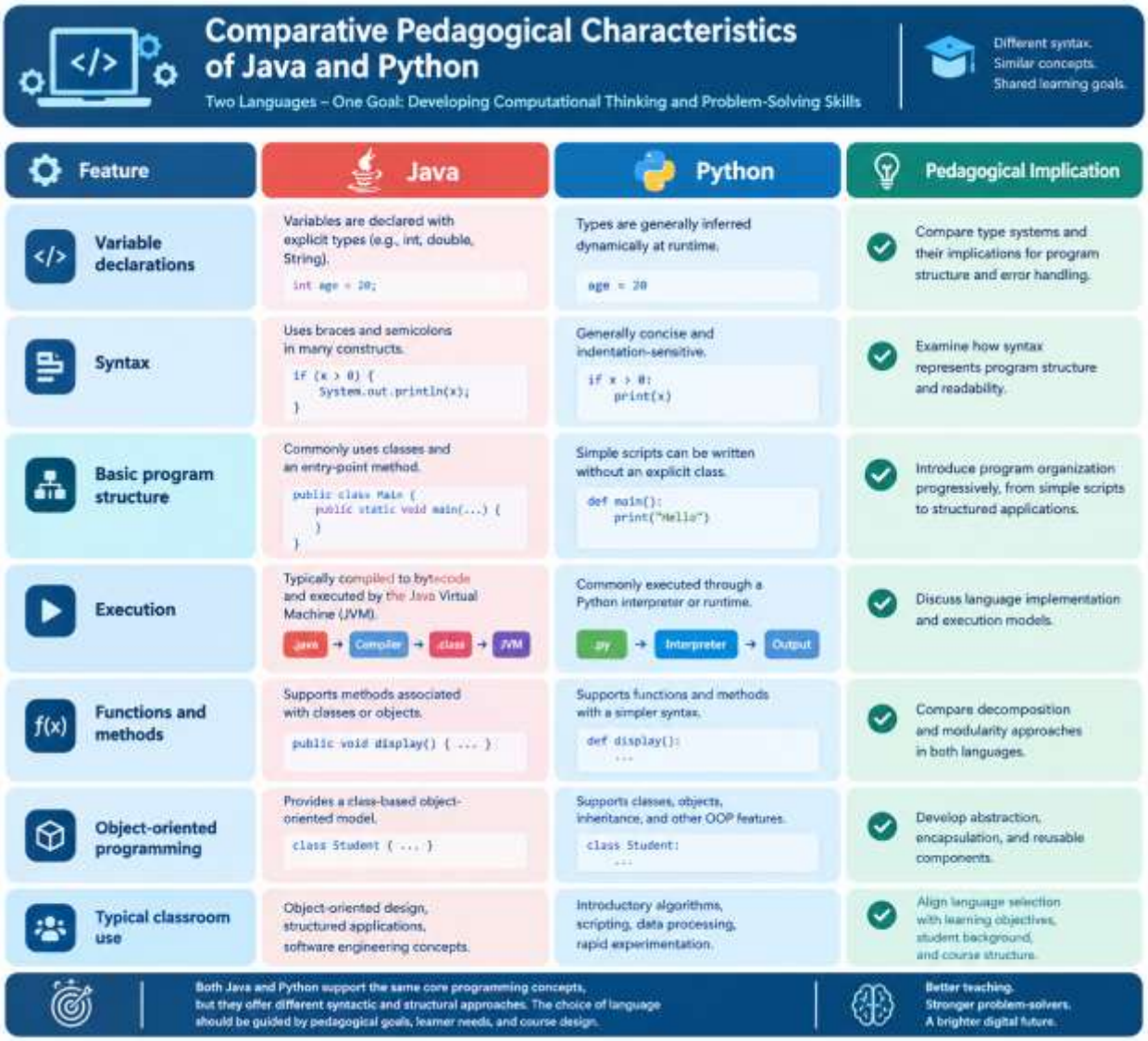
Python is a high-level programming language characterized by relatively concise syntax, dynamic typing, and extensive support for libraries and different programming paradigms.

Its syntax can make fundamental concepts such as variables, conditions, loops, functions, collections, and input/output accessible to beginners.

For example, the following Python program calculates the sum of two numbers:

```
a = 10
b = 20
result = a + b
print(result)
```

The short program allows students to focus on assignment, arithmetic operations, and output without first defining a class or declaring variable types.



Comparative pedagogical characteristics of Java and Python

Python can also support rapid experimentation. Learners can modify a program, execute it, observe the result, and refine their understanding through repeated practice.

Jayal et al. (2011) compared two approaches to introductory programming: beginning with Java and object-oriented concepts, or teaching basic programming concepts with Python before introducing object-oriented programming through Java. Their classroom-based analysis reported results favoring the latter sequence for introductory programming concepts in the context studied.

These findings provide a rationale for investigating staged instruction, but they should not be interpreted as proof that Python is universally superior. The study involved particular student groups, course arrangements, and assessment conditions.

Python also has pedagogical challenges. Its flexible syntax and dynamic typing may conceal some type-related errors until runtime, and beginners still need to develop an understanding of data structures, function behavior, program organization, and algorithmic correctness.

Consequently, Python should be taught as a structured programming language rather than merely as an easy language for writing short scripts.

2.4. JAVA AND THE DEVELOPMENT OF STRUCTURED PROGRAMMING COMPETENCE

Java is a class-based, statically typed programming language widely used in software development and computer science education.

Its explicit type declarations and class-based organization provide opportunities to teach abstraction, encapsulation, method design, object creation, and the relationships between program components.

A simple Java program illustrating arithmetic operations is shown below:

```
public class Main {
    public static void main(String[] args) {
        int a = 10;
        int b = 20;
        int result = a + b;
        System.out.println(result);
    }
}
```

Compared with the Python example, the Java program requires a class declaration, a main method, explicit integer types, and a more structured program entry point.

These additional elements can initially increase the amount of syntax that learners must understand. However, they also provide opportunities to discuss program structure, type declarations, compilation, and the organization of larger applications.

Java becomes particularly relevant when students progress from simple algorithms to object-oriented design, reusable components, and more structured software projects.

Nevertheless, explicit syntax does not automatically produce deeper understanding. Teachers must explain the purpose of classes, methods, types, and objects rather than requiring students to memorize standard program templates without understanding them.

2.5. COMPARATIVE TEACHING OF JAVA AND PYTHON

Teaching Java and Python together creates opportunities to distinguish general programming concepts from language-specific implementations.

For instance, both languages support variables, conditional statements, loops, functions or methods, collections, and object-oriented programming. However, they differ in syntax, type systems, program structure, and common development practices.

Lo et al. (2015) examined the question of which language students should learn first, comparing Java and Python from the perspective of novice programming

education. Their work identifies language characteristics, including syntax and variable declaration, as relevant considerations in introductory instruction.

Kaila et al. (2024) investigated a university curriculum transition from Java to Python. Their study considered student performance, course outcomes, and teaching assistant feedback during the transition. This work illustrates that changing the language used in a curriculum is a pedagogical and organizational decision requiring careful planning.

More broadly, Gomes et al. (2026) examined introductory programming courses at highly ranked universities and reported differences in language selection between first and second introductory courses. Their study provides contemporary evidence that Python and Java can occupy different positions within a broader programming curriculum.

Taken together, these studies support a flexible curriculum perspective. Python may be used to introduce fundamental computational concepts, while Java may provide a structured environment for developing object-oriented programming knowledge. The sequence, however, should be determined by the course objectives and learners' needs.

LIMITATIONS AND FUTURE RESEARCH

This article has several limitations.

First, the literature review is conceptual and purposive rather than systematic. It does not provide exhaustive coverage of research on programming language selection, novice programming, or comparative Java–Python instruction.

Second, the proposed framework has not been evaluated in a classroom intervention. No original student assessments, interviews, surveys, or experimental measurements were collected. Consequently, the framework's effectiveness remains unverified.

Stage	Main focus	Learning activity	Assessment evidence
1	Computational thinking	Algorithm design and pseudocode	Correct problem decomposition and algorithm
2	Programming fundamentals	Short exercises in variables, conditions, loops, and functions	Correct implementation and explanation
3	Comparative programming	Implement the same problem in both languages	Comparison of syntax and shared logic
4	Object-oriented programming	Design classes and create objects	Working class-based implementation
5	Integrated projects	Develop and test a practical application	Source code, test cases, and project report

Proposed stages for teaching Java and Python

Third, the framework assumes that learners can benefit from connecting language-independent algorithmic concepts with language-specific implementation. The extent to which this approach improves transfer of learning has not been established by the present study.

Fourth, the proposed assessment rubric has not undergone formal validation. It should be reviewed by programming education specialists and piloted with the intended student population before being used to measure educational outcomes.

Fifth, the suitability of a Python-first, Java-first, or parallel teaching sequence may depend on student age, previous experience, institutional objectives, instructional time, and available teaching resources.

Future research should investigate these issues through carefully designed empirical studies.

A comparative study could examine students learning through a Python-first sequence and students learning through a Java-first sequence. The study should use comparable learning objectives, document prior programming experience, and employ assessments that measure algorithmic reasoning, code correctness, debugging, and transfer to unfamiliar tasks.

A separate study could investigate whether implementing the same algorithm in both languages helps learners explain the difference between conceptual logic and language-specific syntax.

Researchers could also examine the role of formative feedback, project-based instruction, and AI-assisted programming tools in the development of programming competence.

Such studies should report their sampling procedures, instructional conditions, assessment instruments, ethical safeguards, and limitations transparently. Their findings would help determine which elements of the proposed framework are most useful for different groups of learners.

CONCLUSION

Teaching Java and Python provides an opportunity to develop both foundational programming knowledge and more structured software development skills.

Python can support the introduction of fundamental programming concepts through concise syntax and rapid experimentation. Java can provide a structured environment for teaching explicit type declarations, class-based organization, object-oriented design, and larger program structures.

Nevertheless, neither language should be considered universally superior for all learners or educational objectives. Effective instruction depends on the relationship between language characteristics, learning objectives, students' prior knowledge, and assessment practices.

This article has proposed a five-stage pedagogical framework comprising computational thinking, programming fundamentals, comparative implementation, object-oriented programming, and integrated projects.

The framework emphasizes problem analysis, deliberate practice, systematic debugging, conceptual comparison, and reflective assessment. It offers a practical starting point for designing the course Working with Programming Languages (Java and Python).

The article's contribution is conceptual rather than empirical. The framework has not been validated through classroom experimentation, and its educational effectiveness should not be assumed. Future research should test its feasibility and investigate its effects on programming performance, computational thinking, and the transfer of learning between languages.

By organizing Java and Python instruction around transferable concepts and progressively demanding practical tasks, educators can create a structured learning environment for developing programming competence. Establishing the effectiveness of this approach will require systematic evidence from future educational research.

REFERENCES

1. Gomes, J. P., Monteiro, R., Moreira, A. S., Mendonça, V., Pinho, A., & Cunha, C. R. (2026). Teaching introductory programming in top universities: A global study of languages, paradigms, assessment, and AI. Journal of Computer Languages, 86, Article 101384. https://doi.org/10.1016/j.cola.2025.101384
2. Jayal, A., Lauria, S., Tucker, A., & Swift, S. (2011). Python for teaching introductory programming: A quantitative evaluation. Innovation in Teaching and Learning in Information and Computer Sciences, 10(1). https://doi.org/10.11120/ital.2011.10010086
3. Kaila, E. T., Luukkainen, M., Laaksonen, A., & Lemström, K. (2024). On changing the curriculum programming language from Java to Python. In A. Mühling & I. Jormanainen (Eds.), Proceedings of the 23rd Koli Calling International Conference on Computing Education Research (pp. 1–7). ACM. https://doi.org/10.1145/3631802.3631814
4. Lo, C. A., Lin, Y. T., & Wu, C. C. (2015). Which programming language should students learn first? A comparison of Java and Python. In Proceedings of the 2015 International Conference on Learning and Teaching in Computing and Engineering (pp. 225–226). IEEE. https://doi.org/10.1109/LaTiCE.2015.15
5. Novice use of the Java programming language. (2023). ACM Transactions on Computing Education, 23(1), Article 10, 1–24. https://doi.org/10.1145/3551393