RESEARCH & INNOVATION DIVISION
IJIST

Pedagogical Suitability: A Software Metrics-Based Analysis of Java and Python

Muhammad Shumail Naveed[1]
[1]Department of Computer Science & Information Technology, University of Balochistan, Quetta, Pakistan.
DOI: https://doi.org/10.33411/IJIST/1123
*Correspondence: dr.shumail.cs@um.uob.edu.pk
Citation | Naveed. M. S, "Pedagogical Suitability: A Software Metrics-Based Analysis of Java and Python", IJIST, Vol. 06 Issue. 04 pp 1956-1967, Dec 2024
Received | Oct 15, 2024 **Revised |** Nov 23, 2024 **Accepted |** Nov 28, 2024 **Published |** Dec 01, 2024.

Programming is one of the foundational skills essential for computer science professionals, yet attaining proficiency in this skill is widely acknowledged as a formidable challenge. The intrinsic complexity of programming is often cited as the primary factor contributing to its difficulty. The choice of programming language for IP courses typically relies on past experiences and empirical evidence, rather than on a quantitative basis, which can affect its effectiveness and suitability for novice learners. The study presented in this article conducted a quantitative analysis of Java and Python to assess their suitability for use in IP courses. The analysis involved evaluating programs based on a total of 210 elementary programming algorithms using HCM. The results of the study indicated that Python programs, compared to Java programs, have a reduced reliance on lexical elements, are less complex, and have a smaller code size. Additionally, Python was found to produce less complex programs and required less effort and time for development and maintenance. Moreover, Python programs tend to have fewer bugs. Overall, the study concluded that Python is better suited for IP courses than Java. The novelty of this study lies in its quantitative comparison of Java and Python using HCM, revealing that Python is more appropriate for IP courses due to its lower complexity, reduced development effort, and fewer bugs.

Key to Read

Science, Technology, Engineering, and Mathematics	STEM
Introductory Programming	IP
Halstead Complexity Metrics	HCM

Keywords: IP; Programming Language Comparison, HCM; Java; Python.

Introduction:

Programming constitutes the core of computer science, and acquiring proficiency in programming is crucial for students in information science, computer science, and electrical engineering [1]. The significance of programming languages is underscored by the fact that programming skills are essential not only for computer science majors but also desirable for students pursuing disciplines beyond computer science [2].

The significance and global popularity of programming education have grown significantly. The cognition of programming logical reasoning, algebra, and vocabulary skills—predicts programming performance following an introductory computing course [3]. Due to the rising significance of programming in the professional realm, there has been a surge in the demand for programming education.

An IP course, often referred to as CS1 [4], is critical for students pursuing STEM degrees [5]. In a foundational programming course, novice students are introduced to a range of concepts that typically encompass problem-solving skills, fundamental programming principles, as well as the syntax and semantics of a programming language. Additionally, they learn how to employ this programming language to craft solutions.

In pedagogical environments, it is widely recognized that the programming environment exerts a significant influence on learners' interactions with programming, as well as on the initial learning outcomes in computer science classrooms. Teaching programming has become increasingly demanding, particularly in introductory courses that often have large enrolments. Developing proficiency in writing functional programs is a cognitive skill that many novice programmers struggle to acquire. Equally challenging is the task of assessing students' abilities in this area. Studies have indicated that students must first learn to read and comprehend programs before they can effectively learn to write them [6].

Acquiring programming skills is a rewarding career pursuit, yet mastering them proves challenging [7] and demanding, as substantiated by numerous studies. In the process of learning to program, beginners should emphasize the development of problem-solving abilities alongside gaining a deep understanding of programming syntax and semantics [8]. For the majority of students, embarking on their initial journey into programming proves to be a formidable challenge [9]. The abstract structure, logic, negative perceptions, and anxiety associated with programming are viewed as challenges for novice programmers [10]. Anxiety and frustration frequently manifest when learners encounter challenges in writing programs [11].

A multitude of solutions have been developed to assist students in acquiring fundamental programming skills. Nevertheless, there remains a shortage of solutions addressing the challenges students encounter while learning programming, primarily due to the abstract nature of programming, misconceptions about programming concepts, and a lack of motivation [12].

The primary objective of teaching programming is to help students become proficient in writing computer programs that solve problems. These problems may stem from several factors, including the abstract concepts implied by programming, the problem-solving and problem-decomposition skills required, the fact that many students have never had the opportunity to practice computational thinking or programming, and the requirement for students to quickly learn the syntax, semantics, and structure of a new, unfamiliar language [13]. Significant advancements have been made in tools, methods, and approaches designed for teaching and learning IP [14], but the rates of dropout and failure in these courses remain notably high [15][16].

The initial choice of a programming language exerts a lasting influence on a programmer's development capabilities. The selection of a programming language for teaching IP has been a contentious issue within the computer science and information systems communities. This decision is guided by several criteria, including compatibility with educational

objectives, ease of learning, support for fundamental concepts, industry relevance, institutional preferences, resource availability, support for more advanced courses, and the ability to inspire students. While these criteria are undoubtedly useful, they may lead to oversimplification, limiting exposure to complex concepts and reducing the breadth of students' programming skills. A principally quantitative approach that formally analyses the suitability of programming languages for a first programming course would be more beneficial.

Java is a high-level, multi-platform, and object-oriented programming language, widely used in computing education [17]. Python, on the other hand, is a powerful, interactive, object-oriented, and interpreted scripting language [18]. Various studies have been conducted to analyze the pedagogical significance of Java and Python in IP from different perspectives. However, none has quantitatively examined and compared these languages within the context of IP.

The primary objective of the study presented in this article is to conduct a quantitative analysis of Java and Python to assess their suitability for use in IP courses. Additionally, the study aims to provide evidence-based recommendations for educators on the most appropriate programming language for teaching fundamental programming concepts to beginners, thereby enhancing the effectiveness of IP education.

The novelty of this study lies in its data-driven quantitative analysis of programming languages for pedagogical purposes, which is uncommon in existing literature. This approach offers a fresh perspective on language selection, paving the way for more data-driven decision-making in computer science education.

Related Work:

Numerous studies have been undertaken to examine programming languages from diverse angles and determine their appropriateness for IP courses. Hijón-Neira et al. [19] analysed the effectiveness of introducing foundational programming concepts to first-time programming students using a Java-based visual execution environment. In the study, sixty-three students participated. According to the findings, students' comprehension of basic programming concepts significantly improved when using the visual execution environment, compared to a control group that did not receive instruction with the visual tool.

Ling et al. [20] compared and analyzed the effects of learning programming with Python and Java. Two groups of students participated in the study: one group was taught Java, while the other group learned Python. During the analysis, learning performance, motivation, and maladaptive cognition were compared to evaluate the differences. The results indicated that motivation, computer programming self-efficacy, and the impact of maladaptive cognition on learning performance were significantly higher in the Python group.

A study on the selection of programming languages [21] analysed the implementation of novice algorithms in IP courses using C++ and Java. For the study, 200 algorithms were selected, and their implementation in C++ and Java was evaluated based on difficulty, effort, and time. The results indicated that Java is more complex than C++ and requires more effort to implement novice programming algorithms. Furthermore, the time needed to translate the selected algorithms into C++ was lower than for Java. The study concluded that C++ is more suitable than Java for implementing IP algorithms and is generally more appropriate for IP courses.

Balreira et al. [22] analysed the impact of incorporating the C and Python programming languages in IP courses tailored for engineering students. The study involved the university's IP course and organized students into two groups based on their use of either C or Python. Student data were collected and assessed, incorporating information from questionnaires, assignment scores, and inquiries posted by students in the online class forums. This experimental approach was replicated over two consecutive semesters. The primary findings indicated a preference for Python in terms of students' confidence in learning, motivation, and their overall decisions on programming design. In contrast, C was favored for its ease in understanding data structures.

Xu and Frydenberg [23] presented the implementation of an introductory Python course and enhanced its significance by incorporating topics related to data analytics. The research encompassed a survey of 64 undergraduate students enrolled in the course, to understand their perceived importance of acquiring Python programming skills as they enter the workforce. The study also aimed to explore how course design and various student characteristics influenced their perceptions of learning and performance. The findings revealed a high level of motivation among students to enroll in Python programming courses, driven by the desire to enhance their prospects in the expanding field of data analytics. Additionally, the study uncovered that students with no prior programming experience outperformed those with some background in programming. This suggests that Python proves to be a suitable choice as a first programming language in the context of Information Systems.

Naveed [24] analysed C and C++ by comparing functionally equivalent programs. For the study, 225 algorithms were considered, and their equivalent programs were evaluated by comparing difficulty, effort, time, and delivered bugs. The study found that C required less effort, and time, and resulted in fewer bugs than C++. However, C++ was found to be less difficult than C when developing programs based on the sample algorithms.

Laura-Ochoa and Bedregal-Alpaca [25] conducted a study aimed at enhancing learning in an introductory Python programming course by leveraging programming tools, including PSeInt, CodingBat, and the turtle graphic library. Employing a quasi-experimental methodological design, the study strategically placed the experimental and control groups in separate academic semesters. The control group comprised 41 students, while the experimental group consisted of 36 students. Results demonstrated that the integration of supportive programming tools, such as PSeInt, CodingBat, and the Python turtle graphic library, along with the infusion of computational thinking practices, led to superior learning outcomes for the experimental group. The study's findings underscored that utilizing appropriate tools, which facilitate the comprehension of programming concepts and foster skills development related to computational thinking (such as abstraction and algorithmic thinking), could enhance both student performance and motivation in programming courses.

Viduka et al. [26] conducted a study comparing Java and Python to determine which language is more suitable for mastering programming by examining their characteristics. The study suggested that Python, with its less steep learning curve, is the optimal choice for mastering programming. Additionally, the study indicated that Java is more convenient when used as a second language.

Material and Methods:

The main objective of this study is the quantitative analysis and comparison of the suitability of Java and Python for teaching elementary programming concepts in an introductory course. To achieve this objective, the study employs the research methodology outlined in Figure 1.

The study began with the search for novice programming algorithms. Initially, 297 algorithms were identified, and 210 of them were selected for the study. The programming code in Java and Python for these chosen algorithms was generated using a high-level code generator [18], with consultation from two programming experts. The extracted code was then organized as the programming corpus. Afterward, it underwent refinement by removing redundant code, resulting in an updated programming corpus repository. The updated code for Java and Python was evaluated using (HCM) following the confirmation of Turing equivalence in terms of functionality for the comparable programs.

The HCM is specifically designed to quantify program complexity from source code by utilizing the details of operators and operands [27] as elementary information. The elementary information needed to be gathered from the source program to calculate Halstead complexity is as follows [28]:

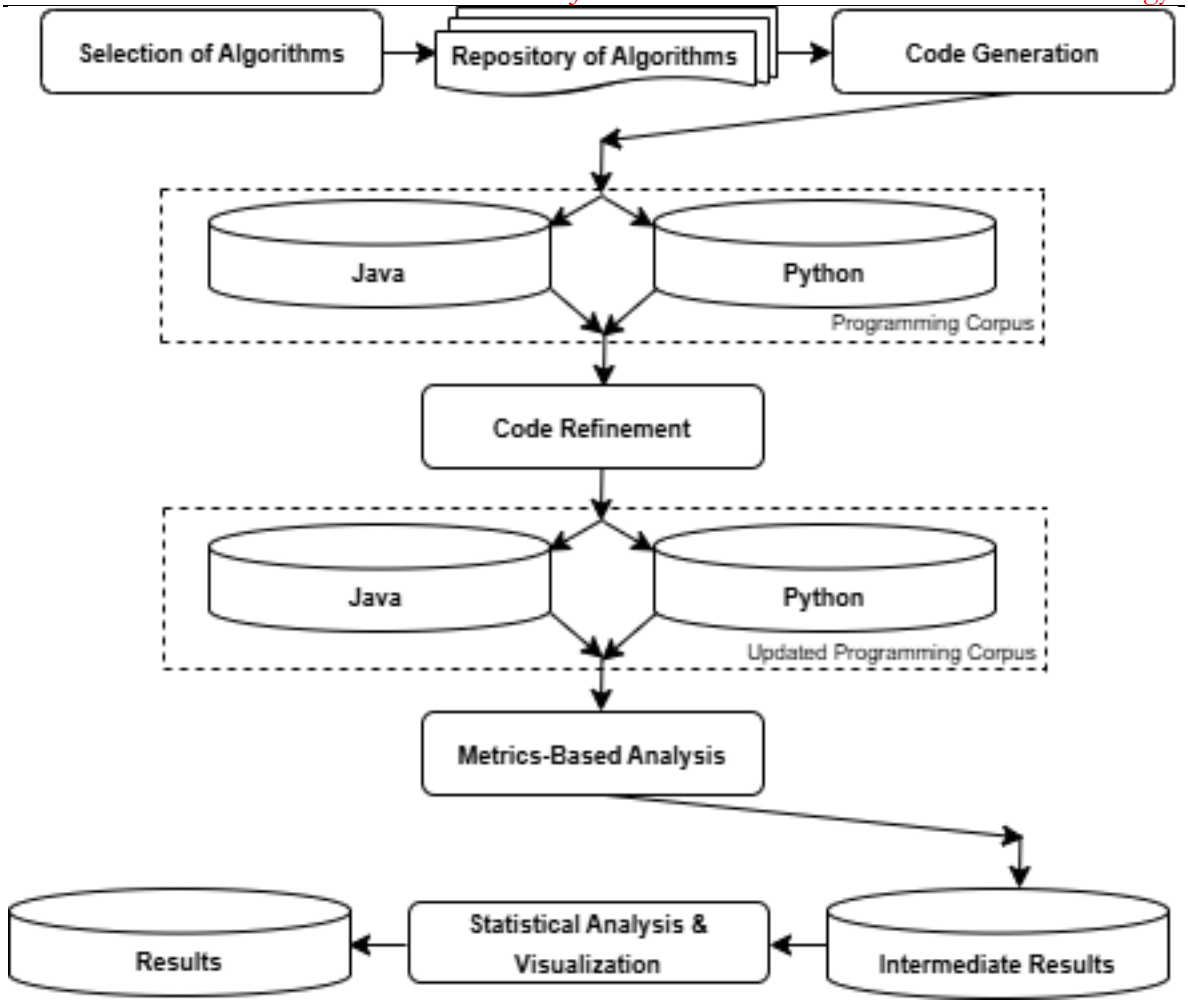


Figure 1. Research Methodology

N_1 = Total number of operators

N_2 = Total number of operands

η_1 = number of unique (distinct) operators

η_2 = number of unique (distinct) operands

The HCM was employed to calculate volume, difficulty, effort, time, and the number of bugs associated with the implementation of the algorithms in Java and Python, using elementary information. Volume represents the information content of the program and defines the scale of algorithm implementation. The following formula is used to calculate the volume

$$Volume = (N_1 + N_2) * log_2 (\eta_1 + \eta_2)$$

The program's level of difficulty is directly related to the count of distinct operators within it, serving as an indicator of the challenge involved in writing or comprehending the program. To compute the difficulty, the following formula is employed

$$Difficulty = \left(\frac{\eta_1}{2}\right) \times \left(\frac{N_2}{\eta_2}\right)$$

Effort quantifies the degree of cognitive activity required **to** transform an existing algorithm into an implementation in a specified programming language. This effort measurement corresponds to the actual coding time and can be calculated using the following formula

$$Efforts = Difficulty \times \text{Volume} = \left(\frac{\eta_1}{2}\right) \times \left(\frac{N_2}{\eta_2}\right) \times \left((N_1 + N_2) \times log_2(\eta_1 + \eta_2)\right)$$

The time required for implementing or comprehending a program is directly related **to** the effort involved. Empirical experiments can be employed to calibrate this relationship. Halstead discovered that dividing the effort by 18 provides an approximate estimation of time in seconds. The following formula is used to calculate the time.

$$Time = \frac{Efforts}{18} \; seconds \; = \frac{\left(\frac{\eta_1}{2}\right) \times \left(\frac{N_2}{\eta_2}\right) \times \left((N_1 + N_2) \times log_2(\eta_1 + \eta_2)\right)}{18}$$

The number of delivered bugs is associated with the overall complexity of the software and serves as an estimate of the number of errors in the implementation. The following formula is used to calculate the time.

$$Bugs = \frac{Efforts^{\frac{2}{3}}}{3000} = \frac{\left(\left(\frac{\eta_1}{2}\right) \times \left(\frac{N_2}{\eta_2}\right) \times \left((N_1 + N_2) \times log_2(\eta_1 + \eta_2)\right)\right)^{\frac{2}{3}}}{3000}$$

Python (3.4) was used to analyze the Halstead complexity of the sampled programs using the formulas mentioned above. Lizard, a simple, straightforward, and ready-to-use Python library, was used to analyze Java programs, while Radon, a popular Python library, was utilized for Python programs. The findings acquired from the analysis of generated programs in Java and Python using HCM were subsequently subjected to statistical analysis performed in SPSS (version: 25), and the resulting outcomes were visualized using the R package (4.2.3).

Results:

To achieve the study's objective, 200 algorithms were implemented in both Java and Python over two phases. In the first phase, elementary analysis was performed, including lexical analysis to identify operators, distinct operators, operands, and distinct operands in each program. The results of the elementary analysis are presented in Table 1.

Table 1. Result of Elementary Analysis

Elements	Language	Mean	Median	Std. Dev	Min	Max	Skewness	Kurtosis
Operators	Java	86.58	81.00	33.84	33.00	244.00	1.34	3.20
	Python	47.79	41.00	26.52	9.00	172.00	1.44	3.31
Distinct Operators	Java	30.55	31.00	4.48	20.00	43.00	0.05	-0.38
	Python	16.34	16.00	4.93	5.00	28.00	0.32	-0.58
Operands	Java	37.78	34.00	21.21	8.00	156.00	2.02	7.56
	Python	35.99	33.00	20.93	4.00	139.00	1.43	3.34
Distinct Operands	Java	12.70	12.00	4.07	4.00	26.00	0.65	0.54
	Python	11.78	11.00	4.44	2.00	27.00	0.62	0.42

The descriptive statistics presented in Table 1 indicate that, in each statistical measure, Python programs were significantly smaller than their equivalent Java programs. This observation strongly suggests that Python supports concise programming. For a more illustrative representation, the results of the elementary analysis are depicted in line charts and can be seen in Figure 2.

The line charts reveal a clear contrast between Java and Python programs in terms of operator and operand usage. Java programs exhibit a broader range of operator and operand counts with significant variability and several extreme peaks, indicating a more complex and diverse set of operations and operands. In contrast, Python programs show smoother, more consistent trends with lower counts and less variability, reflecting a minimalistic and efficient approach to both operators and operands. This pattern highlights the complexity of Java compared to Python's streamlined coding style. The results obtained from the elementary analysis are used in the second phase of the analysis, in which the volume, difficulty, effort, time, and bugs were calculated for the programs in the updated programming corpus with the HCM. The results obtained from the analysis are shown in Table 2.

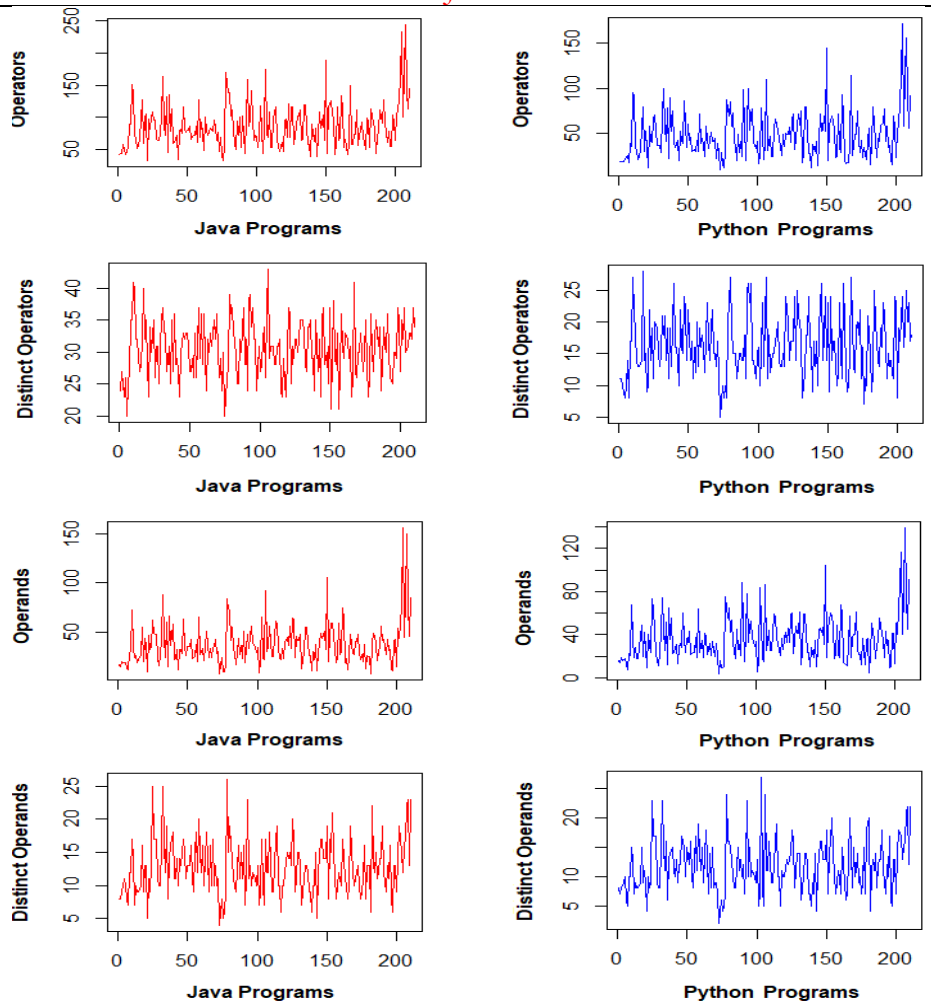


Figure 2. Line Charts of Elementary Analysis

Table 2. Result of Detailed Analysis

Elements	Language	Mean	Median	Std. Dev	Min	Max	Skewness	Kurtosis
Volume	Java	683.71	622.67	322.17	206.72	2277.86	1.52	4.05
	Python	414.31	354.63	259.51	36.50	1580.48	1.37	2.67
Difficulty	Java	44.93	41.89	20.69	13.80	195.00	2.50	13.26
	Python	25.18	22.67	13.61	4.67	71.38	1.03	0.94
Efforts	Java	16195.92	10238.66	20378.30	1033.58	170839.18	4.55	28.67
	Python	10098.09	5930.41	13472.03	72.99	109843.21	3.64	18.96
Time	Java	899.77	568.82	1132.13	57.42	9491.07	4.55	28.67
	Python	561.00	329.47	748.45	4.06	6102.40	3.64	18.96
Bugs	Java	0.19	0.16	0.14	0.03	1.03	2.61	11.25
	Python	0.14	0.11	0.11	0.01	0.76	2.07	6.47

The analysis results obtained using HCM reveal that, across all statistical measures, Python programs exhibit lower volume, difficulty, effort, time, and bugs compared to Java programs. To provide a clearer illustration, the results are visualized using bean plots generated with the R package (version 4.2.3) and presented in Figure 3.

The results of both phases of the study undergo further statistical analysis. In the statistical analysis, the results are initially assessed for normality using the Kolmogorov-Smirnov and Shapiro-Wilk tests, and the findings are presented in Table 3.

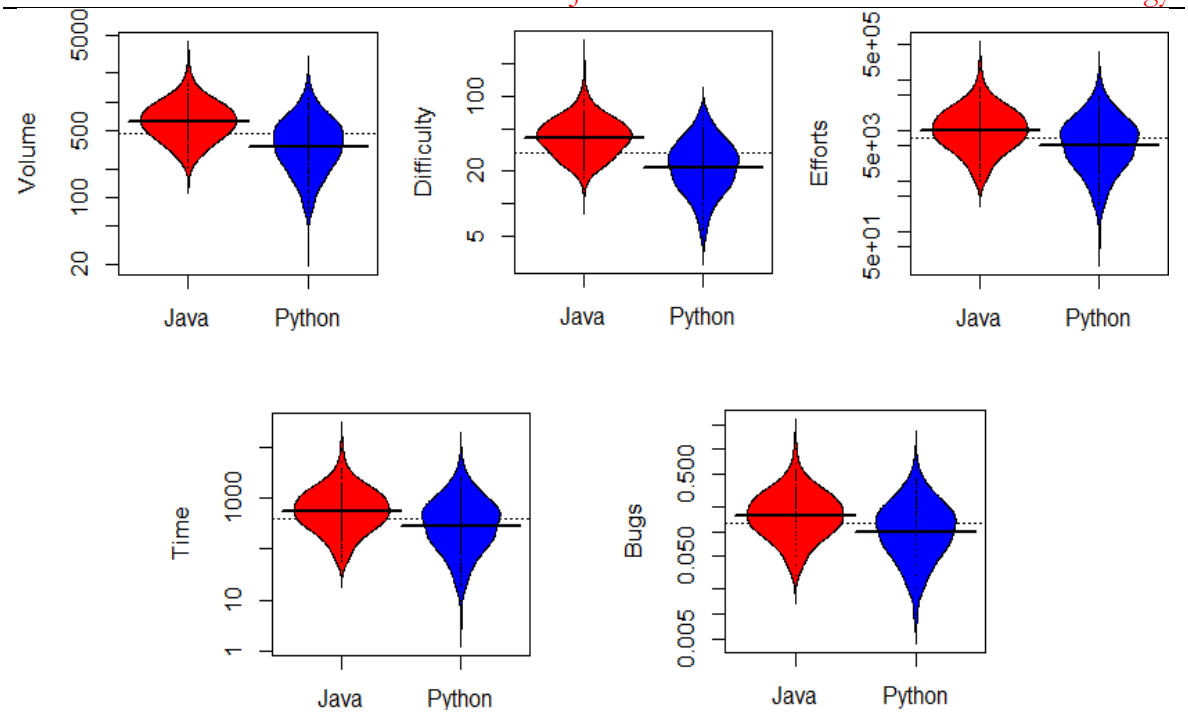


Figure 3. Bean Plots of Detailed Results

Table 3. Result of Normality Test

Elements	Language	Kolmogorov-Smirnov			Shapiro-Wilk		
		Statistic	df	Sig.	Statistic	Df	Sig.
Operators	Java	0.09	210.00	< .05	0.92	210.00	< .05
	Python	0.12	210.00	< .05	0.90	210.00	< .05
Distinct	Java	0.06	210.00	0.08	0.99	210.00	0.18
Operators	Python	0.10	210.00	< .05	0.98	210.00	< .05
Operands	Java	0.11	210.00	< .05	0.86	210.00	< .05
	Python	0.10	210.00	< .05	0.91	210.00	< .05
Distinct	Java	0.10	210.00	< .05	0.97	210.00	< .05
Operands	Python	0.09	210.00	< .05	0.97	210.00	< .05
Volume	Java	0.09	210.00	< .05	0.90	210.00	< .05
	Python	0.11	210.00	< .05	0.90	210.00	< .05
Difficulty	Java	0.11	210.00	< .05	0.86	210.00	< .05
	Python	0.10	210.00	< .05	0.91	210.00	< .05
Effort	Java	0.23	210.00	< .05	0.59	210.00	< .05
	Python	0.23	210.00	< .05	0.64	210.00	< .05
Time	Java	0.23	210.00	< .05	0.59	210.00	< .05
	Python	0.23	210.00	< .05	0.64	210.00	< .05
Bugs	Java	0.13	210.00	< .05	0.79	210.00	< .05
	Python	0.14	210.00	< .05	0.83	210.00	< .05

Non-normality is observed in most of the results. Therefore, to conduct further analysis and comparisons, the non-parametric Mann–Whitney U test is employed. A Mann-Whitney U test has revealed a statistically significant difference between Java and Python in terms of operators, distinct operators, distinct operands, volume, effort, time, and bugs within comparable programs. However, no significant difference was observed in operands between comparable Java and Python programs. Specifically, for operators ($U = 6842.50$; $Z = -12.23$; $p < .05$), distinct operators ($U = 800.50$; $Z = -17.10$; $p < .05$), operands ($U = 20593.50$; $Z = -1.17$; $p = .24$), distinct operands ($U = 19089.50$; $Z = -2.39$; $p = 0.02$), volume ($U = 10182.50$; $Z = -$

9.54; $p < 0.05$), difficulty ($U = 7999.50$; $Z = -11.30$; $p < 0.05$), effort ($U = 15135.00$; $Z = -5.56$; $p < 0.05$), time ($U = 15135.50$; $Z = -5.56$; $p < 0.05$), and bugs ($U = 15175.50$; $Z = -5.53$; $p < 0.05$).

Discussion:

Programming has become of paramount significance and garnered widespread popularity on a global scale in the realm of education [29]. Delivering core programming principles to novices poses a formidable challenge, prompting the creation of various IP languages and platforms tailored to facilitate this learning journey [30]. Educators teaching programming subjects frequently grapple with the question of which programming language to introduce first [31]. This study was conducted to quantitatively analyze the suitability of Java and Python **as** IP languages. It examined the implementation of novice programming algorithms in both languages. The study commenced by identifying operators, distinct operators, operands, and distinct operands within programs developed in Java and Python. The average number of total operators in Java programs was 86.58, compared to 47.79 in Python programs, indicating a percentage difference of 57.74% in operators between Java and Python, demonstrating that Python requires fewer operators for implementing the same algorithms. Additionally, there was a 60.61% difference in distinct operators, a 4.85% difference in operands, and a 7.52% difference in distinct operands, further emphasizing that Python requires less code in terms of distinct operators, operands, and distinct operands compared to Java for novice programming algorithm implementations. A Mann-Whitney U test revealed a statistical difference in operators, distinct operators, and distinct operands, indicating that Java and Python differ significantly in these metrics, with Python being superior for novice programming algorithm implementations. The average volume of Java programs was 683.71, while it was 414.31 for Python programs, resulting in a percentage difference of 40.07. A Mann-Whitney U test identified a statistically significant difference in program volume between Java and Python, indicating that Python programs are less complex and have a smaller code size compared to their Java counterparts. Regarding program difficulty, the average for Java was 44.93, whereas it was 25.18 for Python, resulting in a percentage difference of 56.34. The Mann-Whitney U test found a significant statistical difference in program difficulty, signifying that Python programs are less challenging, more comprehensible, and easier to manage. For program effort, Java programs had an average effort of 16195.92, while Python programs had an average of 10098.09, with a percentage difference of 46.38. The Mann-Whitney U test detected a statistical difference in effort, indicating that Python programs require less effort for development and maintenance. In terms of time, Java programs had an average time of 899.77, while Python programs required 561, resulting in a percentage difference of 46.38. The Mann-Whitney U test demonstrated a statistical difference in the time required for development and maintenance, with Python programs being more time-efficient. Regarding the number of bugs, Java programs had an average of 0.19, while Python programs had an average of 0.14, resulting in a percentage difference of 30.30. The Mann-Whitney U test identified a statistical difference in the number of bugs, suggesting that Python programs tend to have fewer bugs. The study's findings have practical implications, providing institutions and instructors with evidence-based recommendations for selecting a programming language for introductory courses. By quantitatively comparing Python and Java using HCM, the study highlights Python's advantages, including reduced code complexity, less development effort, and fewer bugs. These results suggest that Python is better suited for teaching novices, as it can simplify the learning process, reduce cognitive load, and improve student performance in IP classes. Consequently, adopting Python may enhance student engagement, decrease dropout rates, and increase the overall effectiveness of programming education. The overall analysis clearly indicates that Python is superior to Java for implementing novice programming algorithms. Python's versatility, extensive libraries, and simple, readable syntax contribute **to** its suitability for IP courses. While many studies have explored programming language selection

for novice courses, this study stands out for its novel approach of comparing Java and Python using quantitative metrics. However, this study has several limitations and validity constraints: (i) it is limited **to** comparing only Java and Python, excluding other languages like C, C++, and JavaScript that might be more effective for beginners; (ii) the 210 algorithms used **as** the sample may not fully represent all possible programming scenarios or outcomes; (iii) not all programming concepts were covered by the methods used, which may limit the generalizability of the results; (iv) the study did not consider how the varying skill levels and backgrounds of novice programmers might influence their ability to learn a particular language; (v) relying solely on HCM may not provide a comprehensive analysis, and additional metrics, qualitative assessments, or expert opinions could be necessary; (vi) the conclusions drawn may not be applicable to other programming languages or paradigms; (vii) variations in learners' backgrounds, such as age, prior programming experience, or cognitive abilities, were not considered, which could impact the effectiveness of a language for beginners; (viii) the results may only apply to the controlled setting of the study and might not reflect real-world programming environments; (ix) the specific context of the programming course was not taken into account, which could affect the appropriateness of a programming language for novice learners; and (x) the study does not address how different teaching approaches, such as pair programming or project-based learning, might influence the effectiveness of a programming language in an educational setting. The study recommends that institutions adopt Python **as** the primary language for IP courses due to its lower complexity, reduced development effort, and fewer bugs compared to Java. Python can increase student engagement, reduce dropout rates, and improve learning outcomes, as it is beginner-friendly and facilitates a better understanding of programming concepts. Curricula should focus on languages with lower learning curves, such as Python, to build students' confidence and ease their transition to more advanced topics. As part of future work, i) more diverse and extensive algorithms will be added to ensure a broader coverage of the programming tasks, ii) more programming shall be added in the study, iii) validation of findings with human judgment will be added for deep analysis, iv) additional metrics shall be incorporated to provide accurate assessment of programming languages, and v) a comprehensive and extended longitudinal study will be undertaken to track the evolution of programming languages over time and to assess their suitability for beginners.

Conclusion:

Programming plays a pivotal role in the field of computing; however, learning programming is a complex intellectual undertaking. The selection of the initial programming language for an introductory course has a lasting impact on future dimensions and aptitudes. Novice programmers often grapple with understanding IP concepts, contributing to the challenges associated with learning to program. The decision to choose the first programming language is typically grounded in empirical evidence. The study presented in this article conducted an analysis comparing Java and Python to assess their suitability for use in IP courses. This assessment involved evaluating the programs designed for a total of 210 elementary programming algorithms using HCM. The study's findings indicate that Python is better suited for IP when compared to Java. This preference stems from Python's reduced reliance on lexical elements and its ability to minimize the effort, time, difficulty, and errors associated with novice program implementation. In conclusion, the study suggests that for IP courses, Python is more suitable than Java for teaching novice programming concepts. In the future, additional algorithms will be incorporated into the study, and various programming languages will be employed to facilitate a more comprehensive analysis.

Acknowledgment. The author extends gratitude to Muhammad Tahaam for invaluable assistance in selecting algorithms and developing programs. Additionally, the author acknowledges and appreciates the support and guidance provided by Muhammad Aayaan in the analysis and composition of the article.

Author's Contribution. The article is single-authored.
Conflict of interest. Publication of this research article has no conflict of interest.
Project details. Nill.

REFERENCES

[1] J. C.-C. and F. P. M. Tupac-Yupanqui, C. Vidal-Silva, L. Pavesi-Farriol, A. Sánchez Ortiz, "Exploiting Arduino Features to Develop Programming Competencies," *IEEE Access*, vol. 10, pp. 20602–20615, 2022, doi: 10.1109/ACCESS.2022.3150101.

[2] E. R. and S. Stenbom, "Engineering Students' Experiences of Assessment in Introductory Computer Science Courses," *IEEE Trans. Educ.*, vol. 66, no. 4, pp. 350–359, 2023, doi: 10.1109/TE.2023.3238895.

[3] M. R. I. L. Graafsma, Serje Robidoux, Lyndsey Nickels, Vince Polito, Judy D. Zhu, "The cognition of programming: logical reasoning, algebra and vocabulary skills predict programming performance following an introductory computing course," *J. Cogn. Psychol.*, vol. 35, no. 3, pp. 364–381, 2023, doi: https://doi.org/10.1080/20445911.2023.2166054.

[4] M. S. Naveed and M. Sarim, "Two-Phase CS0 for Introductory Programming: CS0 for CS1," *Proc. Pakistan Acad. Sci. A. Phys. Comput. Sci.*, vol. 59, no. 1, pp. 59–70, Jun. 2022, doi: 10.53560/PPASA(59-1)710.

[5] S. R. Sobral, "Teaching and Learning to Program: Umbrella Review of Introductory Programming in Higher Education," *Mathematics*, vol. 9, no. 15, p. 1737, 2021, doi: 10.3390/math9151737.

[6] A. M. B. Emil Stankov, Mile Jovanov, "Smart generation of code tracing questions for assessment in introductory programming," *Comput. Appl. Eng. Educ.*, vol. 31, no. 1, pp. 5–25, 2023, doi: https://doi.org/10.1002/cae.22567.

[7] Á. F. José Carlos Paiva, José Paulo Leal, "PROGpedia: Collection of source-code submitted to introductory programming assignments," *Data Br.*, vol. 46, p. 108887, 2023, doi: https://doi.org/10.1016/j.dib.2023.108887.

[8] J. J. Sohail Iqbal Malik, Roy Mathew, Abir Al Sideiri, "Enhancing problem-solving skills of novice programmers in an introductory programming course," *Comput. Appl. Eng. Educ.*, vol. 30, no. 1, pp. 174–194, 2022, doi: 10.1002/cae.22450.

[9] P. K. C. I. & C. David Wong-Aitken, Diana Cukierman, "'It Depends on Whether or Not I'm Lucky, How Students in an Introductory Programming Course Discover, Select, and Assess the Utility of Web-Based Resources," *ITiCSE '22 Proc. 27th ACM Conf. Innov. Technol. Comput. Sci. Educ.*, vol. 1, pp. 512–518, 2022, doi: https://doi.org/10.1145/3502718.3524751.

[10] F. Demir, "The effect of different usage of the educational programming language in programming education on the programming anxiety and achievement," *Educ. Inf. Technol.*, vol. 27, pp. 4171–4194, 2022, doi: https://doi.org/10.1007/s10639-021-10750-6.

[11] M. C. L. I. & C. Zahra Atiq, "A Qualitative Study of Emotions Experienced by First-year Engineering Students during Programming Tasks," *ACM Trans. Comput. Educ.*, vol. 22, no. 3, pp. 1–26, 2022, doi: https://doi.org/10.1145/350769.

[12] K. M. Y. and W. N. L. C. Wee, "iProgVR: Design of a Virtual Reality Environment to Improve Introductory Programming Learning," *IEEE Access*, vol. 10, pp. 100054–100078, 2022, doi: 10.1109/ACCESS.2022.3204392.

[13] J. F. and F. García-Peñalvo, "Teaching and Learning Tools for Introductory Programming in University Courses," *2021 Int. Symp. Comput. Educ. (SIIE), Malaga, Spain*, pp. 1–6, 2021, doi: 10.1109/SIIE53363.2021.9583623.

[14] A. I. Idongesit Eteng, Sylvia Akpotuzor, Solomon O. Akinola, "A review on effective approach to teaching computer programming to undergraduates in developing countries," *Sci. African*, vol. 16, p. 01240, 2022, doi: https://doi.org/10.1016/j.sciaf.2022.e01240.

[15] F. J. G.-P. José Figueiredo, "Design science research applied to difficulties of teaching and learning initial programming," *Univers. Access Inf. Soc.*, vol. 23, pp. 1151–1161, 2024, doi: https://doi.org/10.1007/s10209-022-00941-4.

[16] M.-L. H. & W.-Y. C. Chia-Wen Tsai, Michael Yu-Ching Lin, Yih-Ping Cheng, Lynne Lee, Chih-Hsien Lin, Jian-Wei Lin, "Integrating online partial pair programming and socially shared metacognitive regulation for the improvement of students' learning," *Univers. Access Inf. Soc.*,

2024, doi: https://doi.org/10.1007/s10209-024-01127-w.
[17] M. K. I. & C. Neil C. C. Brown, Pierre Weill-Tessier, Maksymilian Sekula, Alexandra-Lucia Costache, "Novice Use of the Java Programming Language," *ACM Trans. Comput. Educ.*, vol. 23, no. 1, pp. 1–24, 2023, doi: https://doi.org/10.1145/3551393.
[18] M. S. N. Kashif Munawar, "The Impact of Language Syntax on the Complexity of Programs: A Case Study of Java and Python," *Int. J. Innov. Sci. Technol.*, vol. 4, no. 3, pp. 683–695, 2022, doi: 10.33411/IJIST/2022040310.
[19] M. D. Raquel Hijón-Neira, Celeste Pizarro, John French, Pedro Paredes-Barragán, "Improving CS1 Programming Learning with Visual Execution Environments," *Information*, vol. 14, no. 10, p. 579, 2023, doi: 10.3390/info14100579.
[20] W.-C. H. Hsiao-Chi Ling , Kuo-Lun Hsiao, "Can Students' Computer Programming Learning Motivation and Effectiveness Be Enhanced by Learning Python Language? A Multi-Group Analysis," *Front. Psychol.*, vol. 11, p. 600814, 2021, doi: 10.3389/fpsyg.2020.600814.
[21] M. S. Naveed, "Comparison of C++ and Java in Implementing Introductory Programming Algorithms," *Quaid-E-Awam Univ. Res. J. Eng. Sci. Technol. Nawabshah.*, vol. 19, no. 1, pp. 95–103, Jun. 2021, doi: 10.52584/QRJ.1901.14.
[22] J. A. W. Silveira, Thiago L. T. da, Dennis Balreira, "Investigating the impact of adopting Python and C languages for introductory engineering programming courses," *Comput. Appl. Eng. Educ.*, vol. 31, no. 1, 2022, doi: 10.1002/cae.22570.
[23] J. X. and M. Frydenberg, "Python Programming in an IS Curriculum: Perceived Relevance and Outcomes," *Inf. Syst. Educ. J.*, vol. 19, no. 4, pp. 37–54, 2021, [Online]. Available: https://files.eric.ed.gov/fulltext/EJ1310052.pdf
[24] M. S. Naveed, "Measuring the Programming Complexity of C and C++ using Halstead Metrics," *Univ. Sindh J. Inf. Commun. Technol.*, vol. 5, no. 4, pp. 158–165, 2021.
[25] L. L.-O. Norka Bedregal-Alpaca, "Incorporation of Computational Thinking Practices to Enhance Learning in a Programming Course," *Int. J. Adv. Comput. Sci. Appl.*, vol. 13, no. 2, pp. 194–200, 2022, doi: http://dx.doi.org/10.14569/IJACSA.2022.0130224.
[26] V. K. and B. L. D. Viduka, "A Comparative Analysis of the Benefits of Python and Java for Beginners," *QUAESTUS Multidiscip. Res. J.*, vol. 1, no. 19, pp. 318–327, 2021.
[27] N. A. Khan B, "Evaluating the effectiveness of decomposed Halstead Metrics in software fault prediction," *PeerJ Comput Sci*, vol. 27, no. 9, p. 1647, 2023, doi: 10.7717/peerj-cs.1647.
[28] K. A. Onyango, "A comparative study of the lexicographical complexity of Java, Python and C languages based on program characteristics," *J. Innov. Technol. Sustain.*, vol. 1, no. 1, pp. 42–67, 2023, [Online]. Available: https://www.academia.edu/122346019/A_comparative_study_of_the_lexicographical_complexity_of_Java_Python_and_C_languages_based_on_program_characteristics
[29] I. L. Graafsma, "The cognition of programming: logical reasoning, algebra and vocabulary skills predict programming performance following an introductory computing course," *J. Cogn. Psychol.*, vol. 35, no. 3, pp. 364–381, 2023, doi: https://doi.org/10.1080/20445911.2023.2166054.
[30] R. P. and B. C. P. Perera, G. Tennakoon, S. Ahangama, "A Systematic Mapping of Introductory Programming Languages for Novice Learners," *IEEE Access*, vol. 9, pp. 88121–88136, 2021, doi: 10.1109/ACCESS.2021.3089560.
[31] A. C. and M. L. E. Lokkila, "A Data-Driven Approach to Compare the Syntactic Difficulty of Programming Languages," *J. Inf. Syst. Educ.*, vol. 34, no. 1, pp. 84–93, 2023, [Online]. Available: https://aisel.aisnet.org/jise/vol34/iss1/7/



Copyright © by authors and 50Sea. This work is licensed under Creative Commons Attribution 4.0 International License.