International Journal of Technology in Education and Science

www.ijtes.net

Selecting Suitable Programming Languages for Beginner-Level Instruction

Adaiti Allen Kadams [1] *, Solomon Sunday Oyelere [2]

[1] Department of Computer Science, Faculty of Computing, Modibbo Adama University, Nigeria, 0000-0002-6381-4226
[2] Dept. of Computer Science, Fac. of Environment, Science and Economy, University of Exeter, United Kingdom, 0000-0001-9895-6796
* Corresponding author: Adaiti Allen Kadams (adaiti@mau.edu.ng)

Article Info

Article History

Received:
11 June 2025

Revised:
27 October 2025

Accepted:
29 November 2025

Published:
1 January 2026

Keywords

UTAUT2
TAM2
Generative AI (GenAI)
Pedagogical approaches
Curriculum design

Abstract

This study examines factors influencing the preference for Python and Java as introductory programming languages in a Nigerian higher education institution. Using an integrated framework combining the Extended Unified Theory of Acceptance and Use of Technology (UTAUT2) and the Technology Acceptance Model (TAM2), key constructs such as perceived usefulness, ease of learning, social influence, and industry relevance were identified as crucial in shaping students' preferences. A survey of 308 second-year students revealed Python as the preferred beginner-level language, with 75.6% favoring it over Java. Python's perceived ease of learning (M = 4.09), usefulness (M = 4.41), and alignment with industry demands (M = 4.34) were significantly higher than Java's (M = 3.31, 3.74, and 3.78 respectively). Additionally, 70 students (over 22%) selected C++ as the best alternative, appreciating its ability to provide a deeper understanding of system-level programming. Regression analysis showed perceived usefulness (β = 0.24), ease of learning (β = 0.22), and industry relevance (β = 0.21) as strong predictors of language preference, especially for Python. Students' perceptions of future use and social influence also significantly predicted preferences, highlighting Python's applicability to emerging technologies and career goals. The study recommends prioritizing Python for introductory courses, retaining Java for advanced topics, and integrating Generative AI tools to enhance programming education outcomes.

Citation: Kadams, A. A. & Oyelere, S. S. (2026). Selecting suitable programming languages for beginner-level instruction. *International Journal of Technology in Education and Science (IJTES), 10*(1), 133-161. https://doi.org/10.46328/ijtes.5061

Introduction

The choice of an introductory programming language for beginner-level instruction is a crucial decision that significantly impacts students' learning experiences and outcomes (Ishaq & Alvi, 2023). While numerous programming languages have dedicated resources for beginners, institutions must balance pedagogical effectiveness with industry relevance when choosing a programming language. However, despite the growing emphasis on programming education, there is a lack of consensus on the most suitable introductory language. Many universities, particularly in Europe, have gravitated towards Java, Python, C++, and C, with Python emerging as a preferred option due to its readability and growing industry demand as shown in a study by Siegfried et al., (2021). He observed that Java, Python, C++, and C are the predominant languages employed in introductory programming courses across European institutions of higher education. This trend is further corroborated by a global survey conducted by Mason et al., (2024), which revealed that Python and Java are jointly the leading programming language for teaching programming globally. Furthermore, a multi-group analysis conducted in a study by Ling et al., (2021) comparing Python and Java in programming courses revealed that students demonstrated significantly higher learning motivation, self-efficacy, and overall effectiveness in Python. The study attributes this preference to Python's simpler data and programming structure and shorter syntax, making it more suitable for beginner-level programming. Collectively, these findings underscore the growing recognition of Python's suitability as an entry-level programming language, driven by both educational benefits and labor market relevance.

Nevertheless, little research explicitly addresses how students perceive these choices and how evolving technological trends influence programming pedagogy. The rapid advancements in educational technology, including the rise of GenAI, have introduced new dynamics into programming education (Zastudil et al., 2023). AI-driven coding assistants, automated debugging tools, and intelligent tutoring systems are transforming how students learn to code. Recent studies indicate that GenAI can influence programming language preferences by lowering the barriers to learning complex languages and providing personalized learning experiences (Phung et al., 2023). This emerging trend highlights the need for a fresh evaluation of programming language selection criteria, ensuring that introductory programming aligns with contemporary technological advancements, industry demands and students' motivation to acquiring coding skills.

A critical question remains: What criteria guide academic institutions in selecting, maintaining, or transitioning to specific programming languages for instructing first-year students? Identifying and understanding these criteria is essential for informing future decisions and ensuring the effective dissemination of foundational programming concepts to the next generation of programmers. Existing research suggests that factors such as ease of learning, community support, industry relevance and pedagogical effectiveness play a role in language selection (Chakraborty et al., 2021; Dela Rosa, 2023). Recent studies emphasize the importance of aligning programming education with real-world applications and industry expectations (Romao et al., 2024). A comprehensive analysis by Dobslaw et al., (2023) identified "market demand" as the most influential factors in language selection. This underscores the need for a balanced approach that considers both students' learning experiences and long-term professional applicability.

Programming language selection is often predominantly guided by institutional policies or instructor preferences, with minimal input from students (Asgari et al., 2024). However, with the increasing impact of GenAI on programming practices and the evolving needs of the tech industry, it is increasingly crucial to incorporate students' perspectives into curriculum decision-making processes. Recent studies emphasize that aligning educational practices with student experiences and expectations is essential for effective integration of GenAI tools in computing education (Keuning et al., 2024; Zastudil et al., 2023). A holistic approach that combines insights from educators, industry professionals, and students can develop a more informed, practical, and widely embraced choice of programming languages. Although prior research has primarily focused on the cognitive and pedagogical aspects of programming education (Singh & Rajendran, 2024), there is limited exploration of how students' perceptions, motivations, and acceptance of programming languages shape their learning experiences. To address this gap, this study investigates students' perspectives on Python and Java as introductory programming languages within a Nigerian higher education institution (HEI). By employing an integrated framework that synthesizes the Extended Unified Theory of Acceptance and Use of Technology (UTAUT2) and the Technology Acceptance Model (TAM2) (Rudhumbu, 2022; Venkatesh et al., 2012), this research identifies key factors influencing programming language preferences. The findings provide valuable empirical insights to inform a balanced and inclusive approach to language selection, one that aligns the viewpoints of educators, industry professionals, and students with the context of emerging technologies such as GenAI and Educational Technology (EdTech) (Haroud & Saqri, 2025).

Alternative Programming Languages based on Learners Perspective

It is worth noting that learners in any field often lack the comprehensive understanding required to identify the most suitable materials for effectively grasping a subject. This is particularly true in the context of programming, where novice learners may not possess the necessary insights to determine which programming language would be suitable for their initial exposure to coding concepts. Furthermore, many university students are primarily driven by the goal of getting good grades or passing courses, rather than pursuing an in-depth comprehension of the subject matter (Abbas et al., 2023). As a result, relying solely on the learner's perspective when selecting an introductory programming language may not be the most prudent approach. Nonetheless, as illustrated by Asgari et al. (2024) research, incorporating students' perspectives and feedback into the selection of pedagogical approaches and programming languages for teaching coding concepts can substantially enhance learning outcomes while developing professional growth among instructors.

In contrast, programming textbooks aimed at beginners are typically designed with the learner's perspective in mind, aiming to foster an engaging and accessible learning experience while promoting language adoption, which in turn boosts the book's sales. Authors of these introductory programming textbooks often base their choices on a programming language they are intimately familiar with and comfortable using, which can potentially introduce biases or limitations.

Therefore, no single category of criteria for choosing a programming language in instructing first-year students and novice programmers is foolproof. While the university curriculum committee or tutors' criteria for selecting a programming language to teach basic programming concepts to new learners may seem more valid, it is also imperative to consider the learners' perspective, as this approach has been predominantly adopted by authors of introductory programming textbooks. A balanced and holistic approach that blends both the teachers' and students' perspectives on programming language selection criteria would enable institutions to make more informed and widely acceptable choices. By incorporating insights from both stakeholders, the chosen programming language would better align with the pedagogical goals of instructors while catering to the learning needs and preferences of students, ultimately enhancing learners' understanding and performance in programming.

Related Works and Theoretical Framework

Related Works

Prior research incorporating both students' perspectives and university committee or tutor viewpoints in selecting a programming language for teaching basic concepts to first-year students has not been extensively explored. However, several studies have attempted to address this challenge of choosing an appropriate programming language for introducing coding to new or early beginners (Perera et al., 2021). Recent studies have expanded our understanding of programming language selection for beginners, by emphasizing the psychological, practical and contextual implications for learners and not only the technical and pedagogical criteria. Building upon earlier research by Kruglyk et al., (2012), and Sobral, (2021), which primarily focused on educators' perspectives, these works incorporate learner-centric factors such as programming anxiety, cognitive load, real-world applicability, and infrastructural limitations. Demir, (2022) investigated the impact of integrating educational programming languages into both theoretical and practical components of programming courses. The study found that such integration significantly reduced programming anxiety and enhanced academic achievement among students. This suggests that a holistic approach to teaching programming, which combines theory with hands-on practice, can alleviate common psychological barriers faced by novices. Jain et al., (2024) conducted a comparative analysis of Python and C to determine their suitability for beginners. The study highlighted Python's simplicity, readability, and extensive library support as key factors that facilitate a smoother learning curve for novices. In contrast, while C offers insights into low-level programming and memory management, its complexity may pose challenges for beginners. Therefore, Python was recommended as a more accessible entry point for those new to programming. Fulton et al., (2021) explored the adoption of Rust, a secure programming language, through interviews and surveys with professional developers. While Rust offers benefits like enhanced security and performance, the study identified challenges such as a steep learning curve and limited library support. These findings underscore the importance of balancing language features with learner accessibility when selecting a programming language for educational purposes. Eteng et al., (2022) added a critical dimension to this discourse by focusing on the challenges faced by undergraduate learners in developing countries. Through a systematic review, they proposed a model for effective programming instruction that leverages mobile and online compilers to improve accessibility, especially in resource-constrained environments. Their work underscores the importance of considering infrastructural realities alongside pedagogical and psychological factors, reinforcing the need for practical, inclusive, and context-aware approaches to programming education.

Collectively, these studies advocate for a learner-centered approach in choosing introductory programming languages. They emphasize the need to consider psychological, technical, and contextual factors to ensure that the selected language not only imparts programming concepts effectively but also supports learner confidence, accessibility, and overall engagement.

Theoretical Framework

The selection of an introductory programming language significantly influences students' learning experiences and their long-term proficiency in software development. Traditional educational theories offer valuable insights into the cognitive and pedagogical dimensions of programming instruction. However, they often overlook crucial factors such as student perceptions, motivations, and technology adoption behaviours. To bridge this gap, this study adopts an integrated framework that combines the Extended Unified Theory of Acceptance and Use of Technology (UTAUT2) and the Technology Acceptance Model (TAM2) to evaluate the factors shaping students' programming language preferences. In the proposed structural model, Programming Language Preference (PLP) serves as the central dependent construct. All constructs derived from UTAUT2 and TAM2, such as perceived ease of use, habitual use, social influence, facilitating conditions, and perceived usefulness, are hypothesized to influence students' PLP. Both TAM2 and UTAUT2 were originally developed to predict technology adoption in organizational contexts but have been successfully applied in educational settings to understand technology acceptance among students and educators (Rudhumbu, 2022; Venkatesh et al., 2012). To address the limitations of each model, this study proposes a synthesized framework that leverages the cognitive focus of TAM2 and the contextual breadth of UTAUT2, supplemented by constructs related to intrinsic motivation, habitual behaviours, and industry relevance. TAM2 focuses on individual cognitive factors such as Perceived Usefulness (PU) and Ease of Learning (EL) (adapted from Perceived Ease of Use (PEU)), which are central to understanding students' academic performance and career goals. However, it neglects social and contextual influences, such as peer input and resource availability, that are often vital in educational settings. Conversely, UTAUT2 incorporates constructs like Social Influence (SI), Facilitating Conditions (FC), and Habitual Use (HT), providing a broader perspective on the social and contextual dimensions of technology adoption. Despite its broader scope, UTAUT2 underemphasizes intrinsic motivation and the relevance of specific technologies to professional applications.

To enhance the explanatory power of the integrated framework, this study incorporates two adapted constructs: Relevance to Industry (RI) and Likelihood of Future Use (LFU). These constructs extend the traditional dimensions of TAM2 and UTAUT2 to better fit educational contexts. RI can be seen as an extension from TAM2 and the Task-Technology Fit (TTF) model, emphasizing how students perceive the value of a programming language in relation to industry trends and employability. LFU, meanwhile, aligns with and deepens the Behavioural Intention component of both models by capturing students' expectations about the long-term applicability and relevance of a programming language across academic and professional contexts. For instance, Python's simplified syntax and intuitive learning curve may align with TAM2 constructs, while UTAUT2 explains how social and institutional factors influence its popularity. The framework is further built by insights from Cognitive Load Theory and Expectancy-Value Theory. Cognitive Load Theory posits that reducing cognitive barriers, such as complex syntax, facilitates efficient learning by minimizing extraneous cognitive load (Quintero-Manes & Vieira, 2024; Sandoval-Medina et al., 2024). Expectancy-Value Theory, on the other hand, highlights that students' motivation to engage with a programming language is shaped by their expectations of success, the perceived value of the language, and its alignment with their future goals (Schoeffel et al., 2021). By combining the strengths of TAM2 and UTAUT2 while integrating overlooked factors related to motivation and industry relevance, this study establishes a strong framework for understanding programming language adoption among novice learners. This multifaceted approach not only strengthens the analytical depth of the study but also yields practical implications for educators and curriculum designers. Specifically, aligning instructional strategies with students' cognitive capacities, motivational drivers, and social-contextual influences allows for more informed and inclusive decisions regarding the selection of introductory programming languages. Ultimately, this framework aims to enhance student engagement, support meaningful learning outcomes, and ensure curricular relevance to evolving industry demands.

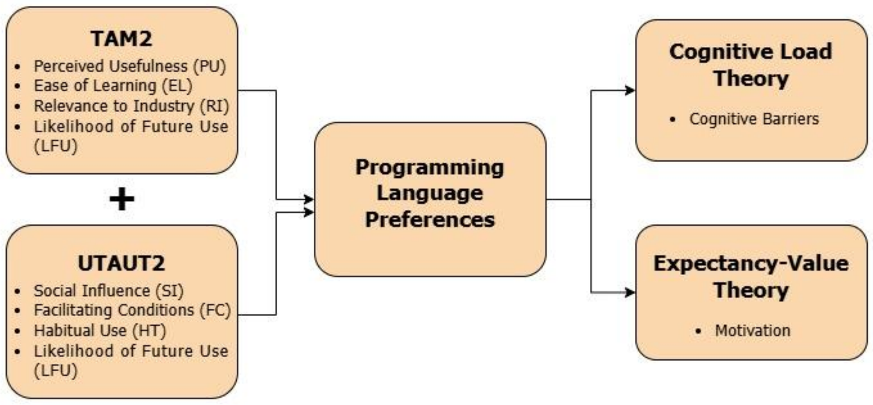
TAM2
Perceived Usefulness (PU)
Ease of Learning (EL)
Relevance to Industry (RI)
Likelihood of Future Use (LFU)
+
UTAUT2
Social Influence (SI)
Facilitating Conditions (FC)
Habitual Use (HT)
Likelihood of Future Use (LFU)
Programming Language Preferences
Cognitive Load Theory
Cognitive Barriers
Expectancy-Value Theory
Motivation

Figure 1. Theoretical Framework Integrating TAM2, UTAUT2, and Pedagogical Theories

Hypotheses Development

Perceived Usefulness (PU)

Perceived Usefulness (PU), derived from the TAM2, corresponds to the concept of performance expectancy. It refers to the degree to which an individual believes that using a particular tool, system, or technology will enhance their performance or help them achieve specific goals (Penney et al., 2021). In this context, students' beliefs about how Python or Java might contribute to achieving academic or professional goals are crucial. These beliefs often emphasize practical benefits such as enhancing academic performance, improving understanding of programming concepts, and supporting future projects. Accordingly, the following hypothesis is proposed:

> *H_1:* There is a significant relationship between students' perceived usefulness of Python and Java and their preference for either language in beginner-level programming.

Ease of Learning (EL)

According to Nguyen et al. (2024), perceived ease of learning (or use) derived from the TAM2 model, denotes the extent to which an individual believes that using a certain technology requires minimal effort. Lin (2022) stressed this factor as a positive influence on students' learning attitudes. This study aims to explore how students' perceived ease of learning influences their preferences for either Python or Java. The hypothesis is stated as:

H_2: Students' perceived ease of learning Python and Java significantly influences their preference for either language in beginner-level programming.

Habitual Use (HT)

Habitual Use (HT), as conceptualized in the UTAUT2 framework, reflects the extent to which individuals use a technology automatically due to repeated exposure and experience over time (Venkatesh et al., 2012). Although HT may appear less relevant at the early stages of learning, Iftikhar et al., (2022) argue that students who regularly engaged with these tools developed better problem-solving skills and a deeper understanding of programming concepts. This leads to the following hypothesis:

H_3: Students who regularly engaged with a programming languages significantly shape their habitual use of the languages, thereby influencing their preference for either Python or Java in beginner-level programming.

Social Influence (SI)

In UTAUT2, Social Influence (SI) refers to the degree to which individuals perceive advice or views from important others, such as peers, family members, or instructors who believe they should use a particular technology. In a study by Shahzad et al., (2023) they highlighted how such social expectations such as social media and peers can powerfully affect a student's technology adoption. Therefore, the study proposes the following hypothesis:

H_4: Social influence significantly affects students' choice between Python and Java for beginner-level programming.

Facilitating Conditions (FC)

Facilitating Conditions (FC) is a construct within the UTAUT2 extended model, referring to individuals' perceptions of the resources and support necessary for utilizing a technology. It reflects the extent to which individuals believe that the required infrastructure, tools, and assistance are accessible and adequate to enable effective use of the technology. ENUDI & Umoeshiet E. Akpan, (2023) noted a significant positive correlation between students' accessibility to instructional materials and greater student engagement and meaningful contribution to the system. A learner is more likely to be influenced by the number of facilities and resources available to use in learning and using a programming language, thus the following hypothesis was proposed.

H_5: Institutional support and resources significantly impact students' preference for Python or Java in beginner-level programming.

Relevance to Industry (RI)

This study connects the concept of Relevance to Industry (RI) with the Job Relevance construct from TAM2 and the Task-Technology Fit (TTF) model. In TAM2, RI refers to the extent to which a technology supports users' tasks and goals in ways that are applicable to professional contexts. Romao et al., (2024) similarly highlight the importance of aligning educational programs with industry requirements, noting that such alignment enhances student engagement and learning through the integration of real-world applications. This connection not only bridges the gap between academic theory and practical skills but also prepares graduates to meet the evolving demands of the job market. RI also corresponds to the utility value dimension of Expectancy-Value Theory, especially in contexts where employability and career advancement are key motivators. Programming languages perceived as widely adopted in the industry, whether in enterprise systems, startup ecosystems, or trending tech stacks, are more likely to be valued by students. Those with clear career goals may prioritize languages that align with market trends and perceived job opportunities.

H_6: Students' perceived need to learn Python or Java is significantly influenced by their perceptions of the languages' relevance to current industry demands and practices.

Likelihood of Future Use (LFU)

Likelihood of Future Use (LFU) serves as a proxy for the Behavioral Intention (BI) construct in UTAUT2 and the various iterations of the TAM2. It refers to a student's intention or likelihood to continue using technology in the future. In a study by Parveen et al., (2024), playfulness emerged as a key factor influencing students' likelihood of using ChatGPT in the future, followed by perceived value and performance expectancy. In the context of this study, LFU will be assessed based on the constructs of Perceived Usefulness (PU) and Effort Expectancy/Ease of Learning (EL), which are recognized in the literature as significant predictors of BI (Tey & Moses, 2018).

LFU captures students' expectations regarding the future utility and frequency of use of a programming language across academic, professional, and personal contexts. This construct is grounded in Expectancy-Value Theory (Eccles & Wigfield, 2023), particularly the components of expectancy for success and utility value. Expectancy for success refers to students' beliefs about their ability to successfully use the language in the future, while utility value reflects their perception of its relevance for achieving long-term goals. In programming education, LFU is particularly salient. Students may evaluate a language not only for its immediate usefulness in coursework but also for its applicability in later modules, software projects, internships, and career aspirations. A language perceived as broadly applicable, such as being prevalent in emerging technologies, open-source communities, or industry-standard environments, is likely to be associated with higher LFU. Therefore, the following hypothesis is proposed.

H_7: Students' likelihood of future use of Python and Java is significantly influenced by their perceptions of the languages' usefulness and ease of learning.

Methodology

This study was conducted at a Nigerian higher education institution, targeting second-year students from various faculties and departments. The aim was to gather students' perspectives on selecting a suitable programming language for teaching fundamental programming concepts. A quantitative, survey-based approach was employed using Google Forms for data collection. Google Forms was selected as the data collection instrument due to its accessibility, cost-effectiveness, and widespread adoption among students, aligning with findings from White, (2015) research on students' familiarity with google applications for Education. Panchbudhe et al., (2024) highlighted the platform's capabilities for seamless distribution, real-time data collection, and user-friendly response tracking, characteristics that make it particularly suitable for large-scale academic surveys. Furthermore, this choice was especially appropriate for university students in developing countries, as research by Edeh et al., (2022) demonstrated that the majority of these students possess Android smartphones and regularly engage with Google applications, ensuring high participation rates and minimal technical barriers.

To distribute the questionnaire, a link to the Google Form was shared in the WhatsApp groups of the 799 enrolled second-year students. Python and Java were chosen as baseline languages due to their global popularity, as discussed in the literature review. Additionally, students were given the option to indicate alternative language preferences, ensuring a broader perspective on programming language selection.

Sample and Data Collection Process

The survey targeted second-year students who had taken or were currently enrolled in a programming course, with 308 students responding. The participants (n=308) varied in gender and age. The age range majority (72.4%) were between 18-24 years old. Males constitute 87.9% and female 8.8%. The respondents, 271 (87.9%) were computer science majors, with the remainder from Engineering, Information Technology, and other fields (see Table 1). The survey was based on the UTAUT2 and TAM2 integrated frameworks by incorporating constructs such as perceived usefulness, ease of learning, habitual use, social influence, facilitating conditions, industry relevance, and future use. Data collection spanned four weeks, with participants informed about the study's purpose and providing consent before participation. Given the predominantly male and computer science-major respondent pool, the generalizability of the findings is somewhat limited. However, very few females enroll in STEM related course in HEIs of developing countries (BusinessDay, 2024; Sosale et al., 2023). Future research could consider employing targeted sampling strategies to diversify the participant pool. This could include actively engaging female students through dedicated outreach efforts. Such an approach would provide a more representative dataset, capturing a wider range of perspectives on programming language selection. Notably, most students from non-Computer Science departments did not participate due to limited interest in programming courses, which contributed to non-responses.

Table 1. Demographic Profile of the Participants

Demography Variable	Demography Classification	Frequency (N=308)	Percentage (%)
Age Range	< 18	11	3.6
	18-24	223	72.4
	25-34	71	23.1
	35-44	3	1.0
Gender	Male	281	87.9
	Female	27	8.8
Field of Study	Computer Science	271	87.9

Demography Variable	Demography Classification	Frequency (N=308)	Percentage (%)
	Physical Sciences	16	5.2
	Engineering	10	3.2
	Environmental Sciences	10	3.2
	Life Sciences	1	0.3

Data Analysis

In this study, survey responses were transformed into a standardized 5-point Likert scale, ranging from 1 (Strongly Disagree) to 5 (Strongly Agree). We selected Python's statistical libraries for their analytical capabilities and robust ecosystem. These libraries provide exceptional functionality across data manipulation, visualization, machine learning, and statistical analysis, encompassing descriptive statistics, inferential testing, advanced regression modeling, and natural language processing. Moreover, Python's open-source nature ensures broad accessibility and benefits from continuous community-driven improvements and rigorous peer review (Joshi & Tiwari, 2023; Mahalaxmi et al., 2023).

Data analysis was conducted using both specialized software and programming tools. SmartPLS 4 (Ringle et al., 2024) was used to perform Partial Least Squares Structural Equation Modeling (PLS-SEM), which was selected for its ability to handle complex, multi-construct models in exploratory research settings where theoretical frameworks are still developing (Hair & Alamer, 2022). This method enabled the examination of both direct and indirect relationships between constructs, that aligns with the study's objective of understanding the key factors influencing programming language preference (PLP). Although SmartPLS reported an SRMR value of 0.000 due to the deterministic nature of the PLP score, which was computed as a direct average of its predictor constructs, this value did not reflect meaningful model fit. To address this, CB-SEM-style fit indices, including Chi-Square (χ^2), RMSEA, CFI, TLI, and SRMR, were approximated using matrix-based computations in Python. These calculations utilized libraries such as Pandas, NumPy, SciPy, and scikit-learn (PCA) to estimate the model-implied covariance structure and corresponding fit statistics. The resulting SRMR of 0.049, along with perfect fit values for CFI (1.000), TLI (1.000), and RMSEA (0.000), confirmed the model's strong structural validity and robustness. This hybrid approach ensured that the model was assessed both in terms of predictive performance (via PLS-SEM) and overall structural quality (via CB-SEM-style fit evaluation).

The constructs used in this study include Perceived Usefulness, Ease of Learning, Habitual Use, Social Influence, Facilitating Conditions, Relevance to Industry, and Likelihood of Future Use. Each construct was operationalized to align with the specific context of programming language selection:

Table 2. Conceptual Framework Constructs and Descriptions

Construct	Description	Definition of Hypothesis	Hypothesis
PU	Does usefulness affect PLP?	Students' belief that learning a programming language will enhance their academic or career prospects.	H_1

EL	Does learning ease affect PLP?	Students' perception of how intuitive, accessible and easy a language is for beginners.	H_2
HT	Does routine use affect PLP?	The extent to which students have formed a routine or preference for a programming language based on prior experience, such as exposure during K-12 education or personal coding projects.	H_3
SI	Do peers influence PLP?	The impact of recommendations from peers, educators, or industry trends on students' programming language choices.	H_4
FC	Does support or environment affect PLP?	The availability of institutional support, including access to learning materials, programming tools, and faculty guidance, which can influence students' ease of adopting a particular language.	H_5
RI	Does real-world relevance affect PLP?	Whether students perceive a programming language as valuable in professional settings.	H_6
LFU	Will future intent affect PLP?	Students' intent to continue using a programming language beyond their introductory coursework.	H_7

Note. Constructs were adapted to fit the context of programming language selection.

Employing PLS-SEM, this study analyzed the direct and indirect effects of these constructs on programming language preference (PLP). The results provide insights into how these factors collectively shape students' programming language choices, offering valuable implications for curriculum design and educational policy (see Figure. 2 below for the proposed structural model).

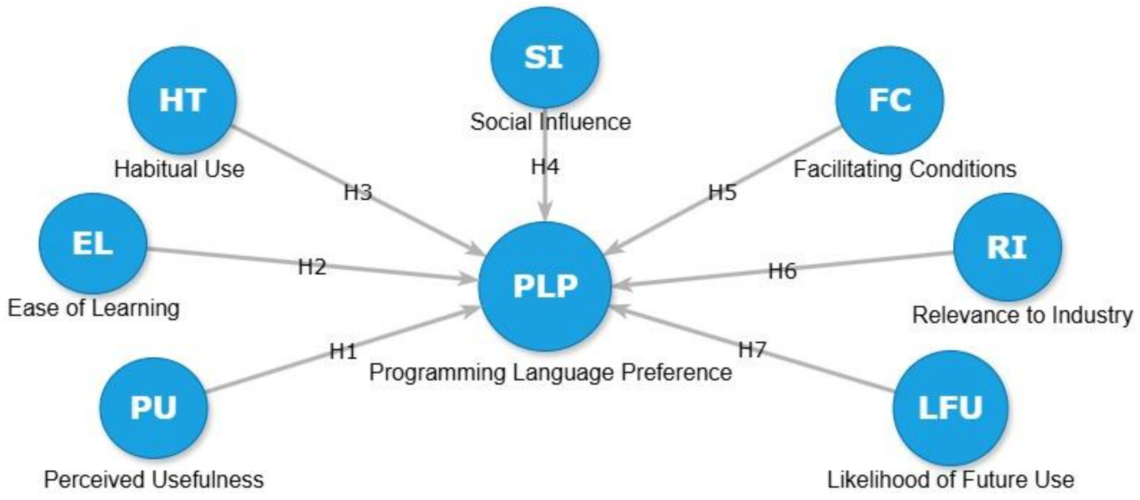

Figure 2. Proposed Structural Model

Data Preparation and Screening

To screen the dataset for analysis, the participants were assessed based on their preference between Python and Java, we calculated difference scores for each construct by subtracting the Python-related item scores from the corresponding Java-related item scores (e.g., PU_Diff = PU_Java - PU_Python). A score of zero indicated neutrality, while positive or negative scores reflected a preference for Java or Python in the dataset, respectively. Of the 308 participants, 221 showed a clear preference, with 167 (75.6%) favoring Python and 46 (20.8%) preferring Java. A small subset of 8 participants (3.6%) displayed neu tral scores, and were excluded from the analysis leaving a final sample of 213 participants: 167 who preferred Python and 46 who preferred Java.

For the multi-group analysis (MGA) in SmartPLS, the dataset of 213 participants was initially organized in a wide format, with separate columns for each construct's scores for Python and Java (e.g., PU_Python, PU_Java). However, SmartPLS requires a "grouped" structure where each observation corresponds to a single case within a group. To meet this requirement, the dataset was reshaped into a long format, where each participant's responses were recorded in two rows: one for Python and one for Java. A new Gro up variable was created to indicate the programming language context. The constructs were then unified under common column names (e.g., PU, EL, HT, etc.). This restructuring resulted in 426 rows (two for each of the 213 participants based on their preferences for java and python) and enabled the assessment of structural relationships and measurement invariance across the Python and Java groups using PLS-SEM.

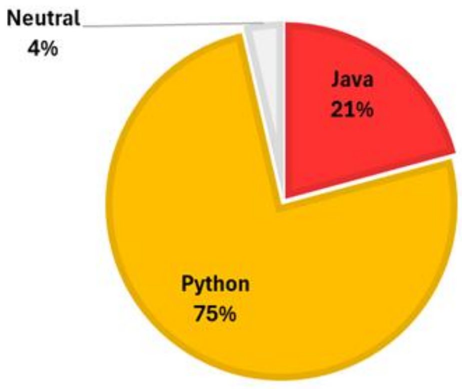

Figure 3. Programming Language Preference Distribution

Results

In this study, we present the results of our analysis, beginning with examining the measurement model evaluation, which confirms the reliability and validity of the constructs used in the study. The reliability and convergent validity of the measurement model was assessed by examining the outer loadings and Composite Reliability (CR) for each single-item construct across both Python and Java groups. Since each construct was represented by a single reflective indicator, Cronbach's Alpha and Average Variance Extracted (AVE) were not separately reported, as CR and AVE are mathematically equivalent in this context. Outer loadings exceeded the recommended threshold of 0.60, and CR values were satisfactory, supporting the adequacy of the measurement model. Table 3 below presents the detailed results.

Table 3. Convergent Validity and Reliability of Single-Item Constructs

Measurement	Outer Loadings	Mean (SD)	Composite Reliability (CR)
Perceived Usefulness (PU)			
pu_python	0.710	4.41 (0.85)	0.50
pu_java	0.777	3.74 (1.08)	0.60
Ease of Learning (EL)			
el_python	0.785	4.09 (0.99)	0.62
el_java	0.729	3.31 (1.04)	0.53
Habitual Use (HT)			
ht_python	0.753	4.20 (0.91)	0.57
ht_java	0.802	3.51 (1.02)	0.64
Social Influence (SI)			
si_python	0.682	4.12 (0.93)	0.46
si_java	0.782	3.45 (1.00)	0.61
Facilitating Conditions (FC)			
fc_python	0.640	3.54 (1.18)	0.41
fc_java	0.614	3.04 (1.20)	0.38
Relevance to Industry (RI)			
ri_python	0.761	4.34 (0.87)	0.58
ri_java	0.682	3.78 (1.00)	0.46
Likelihood of Future Use (LFU)			
lfu_python	0.757	4.24 (0.91)	0.57
lfu_java	0.773	3.55 (1.07)	0.60

Measurement Model Evaluation

To ensure the validity and reliability of the measurement model, outer loadings, internal consistency reliability, and convergent validity were assessed. All constructs demonstrated strong outer loadings above 0.70, indicating that each item reliably represented its respective latent variable. Internal consistency reliability was confirmed, with Composite Reliability (CR) values exceeding the 0.70 threshold across all constructs. Convergent validity was also established, as each construct's Average Variance Extracted (AVE) surpassed the 0.50 benchmark, demonstrating that a significant proportion of the variance was captured by the indicators relative to the measurement error.

The reliability and validity of the measurement model were further evaluated using Outer Loadings and Composite Reliability (CR) (see Table 3). The results demonstrate strong internal consistency and convergent validity across all constructs. Path coefficients for the relationships between the constructs and PLP are displayed in Figure 4. Perceived Usefulness (PU) (H_1): The Composite Reliability (CR) for Perceived Usefulness was 0.50. Students rated Python (M = 4.41, SD = 0.85) as more useful than Java (M = 3.74, SD = 1.08), supporting the hypothesis that perceived usefulness influences programming language preference. The Outer Loading for PU was 0.710 for Python and 0.777 for Java. Ease of Learning (EL) (H_2): The CR for Ease of Learning was 0.62. Python (M = 4.09, SD = 0.99) was perceived as easier to learn than Java (M = 3.31, SD = 1.04), suggesting that ease of learning influences language choice. The Outer Loading for EL was 0.785 for Python and 0.729 for Java. Habitual Use (HT) (H_3): The CR for Habitual Use was 0.57. Python (M = 4.20, SD = 0.91) was used more habitually than Java (M = 3.51, SD = 1.02), indicating that prior experience shapes habitual use of the language. The Outer Loading for HT was 0.753 for Python and 0.802 for Java. Social Influence (SI) (H_4): The CR for Social Influence was 0.46. Python (M = 4.12, SD = 0.93) was more influenced by social factors than Java (M = 3.45, SD = 1.00), highlighting the role of social perceptions in language selection. The Outer Loading for SI was 0.682 for Python and 0.782 for Java. Facilitating Conditions (FC) (H_5): The CR for Facilitating Conditions was 0.41. Both Python (M = 3.54, SD = 1.18) and Java (M = 3.04, SD = 1.20) had relatively low ratings for institutional support, indicating the need for better resources for both languages. The Outer Loading for FC was 0.640 for Python and 0.614 for Java. Relevance to Industry (RI) (H_6): The CR for Relevance to Industry was 0.58. Python (M = 4.34, SD = 0.87) was considered more relevant to industry than Java (M = 3.78, SD = 1.00), suggesting that industry demand influences language preference. The Outer Loading for RI was 0.761 for Python and 0.682 for Java. Likelihood of Future Use (LFU) (H_7): The CR for Likelihood of Future Use was 0.57. Python (M = 4.24, SD = 0.91) was rated as more likely to be used in the future than Java (M = 3.55, SD = 1.07), reinforcing the idea that perceived usefulness and ease of learning affect continued use. The Outer Loading for LFU was 0.757 for Python and 0.773 for Java.

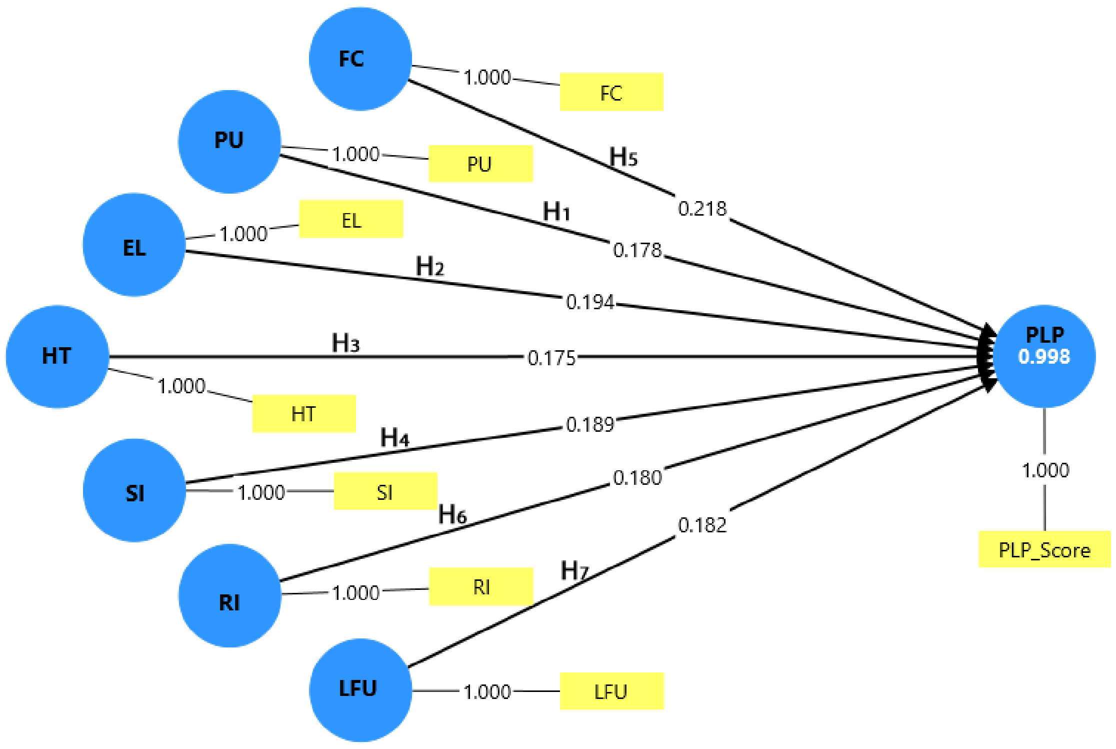

Figure 4. Tested Structural Model with Hypothesized Paths (H1–H7)

Figure 4 illustrates the hypothesized relationships between the constructs and Programming Language Preference (PLP). Labels (H1–H7) correspond to the hypotheses presented in Table 2. All constructs are modeled as direct contributors to PLP, a composite variable (PLP_Score) calculated as the mean of all predictor constructs. The path coefficients and R^2 values are derived from PLS-SEM analysis in SmartPLS 4, with significance levels indicated for each relationship.

Structural Model Evaluation

Although the initial structural model was conceptualized with Programming Language Preference (PLP) as a latent endogenous variable, in practice, PLP was computed as a composite score, the arithmetic mean of its seven formative constructs: Perceived Usefulness (PU), Ease of Learning (EL), Habitual Use (HT), Social Influence (SI), Facilitating Conditions (FC), Relevance to Industry (RI), and Likelihood of Future Use (LFU). Therefore, the model represents a formative structure in which each construct contributes directly to the computed PLP score. This does not alter the theoretical paths proposed but instead reframes the model as an evaluation of the relative influence of each construct on programming language preference (PLP) rather than a causal test of a latent outcome.

The structural model was evaluated to test the hypothesized relationships (H1 to H7) between constructs and Programming Language Preference (PLP). Path coefficients were examined for significance and directionality. Results indicated that all constructs had positive effects on PLP, with path coefficients ranging between 0.17 and 0.24. Specifically, Relevance to Industry (RI), Perceived Usefulness (PU), and Ease of Learning (EL) exhibited comparatively stronger effects. The model's explanatory power was substantial, with an R-squared (r^2) value of 0.997 for both Python and Java groups, indicating that the independent variables explained nearly all the variance in PLP.

Model fit was evaluated using the Standardized Root Mean Square Residual (SRMR), a commonly used measure in Partial Least Squares Structural Equation Modeling (PLS-SEM). The SRMR values were 0.053 for Python and 0.053 for Java, both well below the recommended threshold of 0.08. This confirms that the model fits the data adequately and supports the validity of the hypothesized structure. However, SmartPLS 4 also reported an SRMR of 0.000 due to the deterministic construction of the PLP variable as a direct average of the predictors, leaving no residual variance. To obtain a more meaningful assessment of model fit, CB-SEM-style fit indices were approximated using covariance-based matrix calculations in Python. These approximations yielded an SRMR of 0.049, alongside excellent fit indices including CFI = 1.000, TLI = 1.000, and RMSEA = 0.000, further confirming the model's structural validity.

To further support the structural model, covariance-based SEM-style fit indices were estimated, including Chi-Square, RMSEA, CFI, and TLI. These values were derived from an approximation of the full covariance matrix and indicated an excellent model fit: χ^2 (28) = 0.615, RMSEA = 0.000, CFI = 1.000, and TLI = 1.000. These results confirm that the proposed structural model not only achieves strong predictive power (as evidenced by R^2) but also adheres to established structural fit standards.

Table 5. Covariance-Based Model Fit Indices

Fit Index	Value	Threshold / Interpretation
Chi-Square (χ^2)	0.615	Near-zero = excellent fit
Degrees of Freedom	28	–
RMSEA	0.000	≤ 0.05 = excellent
CFI	1.000	≥ 0.95 = excellent
TLI	1.000	≥ 0.95 = excellent
SRMR	0.049	≤ 0.08 = excellent fit

Multi-Group Analysis (MGA)

To explore differences in construct effects between Python and Java groups, Multi-Group Analysis (MGA) was performed using permutation testing. The analysis compared the path coefficients of the two groups to determine whether the effects of the constructs on PLP significantly differed.The MGA results revealed no statistically significant differences between the Python and Java groups for any of the constructs (p-values > 0.05 across all comparisons). This indicates that, although there were minor variations in the direction and magnitude of the effects, none of these differences were statistically meaningful (see table 6). For instance, Perceived Usefulness (PU) and Ease of Learning (EL) had slightly higher effects among Python users, while Habitual Use (HT) and Social Influence (SI) showed marginally stronger effects for Java users. However, these differences did not reach statistical significance (p-values > 0.05). These differences, although not statistically significant, suggest that students' preferences are shaped by similar factors, with subtle variations attributable to language-specific experiences or contextual influences.

Table 6. Path Coefficients, t-Values, and p-Values for Multi-Group Analysis (MGA)

Path	Python Group Coefficient (β)	Java Group Coefficient (β)	Difference (β)	t-value (Python)	t-value (Java)	p-value
PU → PLP	0.191997	0.171997	-0.000011	-0.023588	-0.023588	0.106
EL → PLP	-0.013943	0.205633	0.000052	-0.028457	-0.039517	0.329
HT → PLP	-0.018581	0.188338	-0.00016	-0.025063	-0.025063	0.144
SI → PLP	0.194167	0.200314	-0.030427	-0.030427	-0.030427	0.69
FC → PLP	0.244256	-0.019417	0.000302	-0.023331	-0.023331	0.683
RI → PLP	0.171997	0.005121	-0.023588	-0.023588	-0.023588	0.106
LFU → PLP	0.185643	0.190551	0.000138	-0.023331	-0.023331	0.106

Since none of the path coefficients had statistically significant p-values (i.e., all p-values > 0.05), we can conclude that there were no significant differences between the Python and Java groups in how the constructs influenced Programming Language Preference (PLP). The variations observed in the direction and magnitude of the effects were minor and not statistically significant. This suggests that both Python and Java users are influenced by a common set of factors, with only slight variations likely due to language-specific experiences or contextual factors.

The MGA results support the robustness of the model and reinforce the theoretical framework that combines UTAUT2 and TAM2 to explain programming language preference. While there are minor variations between Python and Java users in terms of the influence of the constructs, these differences were not statistically significant. Nevertheless, the findings offer valuable insights for educators and curriculum designers, highlighting that both Python and Java are similarly influenced by factors such as perceived usefulness, ease of learning, and industry relevance, which can guide programming instruction aligned with learner expectations and industry demands.

Discussion and Implications

Interpretation of Key Findings

Based on the integrated UTAUT2 and TAM constructs in this study, we have gotten empirical insights into students' preferences for a beginner-level programming language. It has been revealed that there is a consistent preference towards Python over Java. The results show that Perceived Usefulness (PU), Ease of Learning (EL), and Relevance to Industry (RI) were the most influential factors shaping programming language preference, which corroborate prior research that positions Python as a beginner-friendly language with strong professional applicability (Jain et al., 2024; Ling et al., 2021). Python's simplified syntax and readability appear to lower the cognitive load for novices while boosting their motivation, contributing to a more positive view of its learning effectiveness and future utilization. Despite Java's established role in computer science education, its comparatively lower scores in ease of learning (EL) and perceived usefulness (PU) suggests a growing mismatch between traditional curriculum design and student expectations. Notably, the Likelihood of Future Use (LFU) and Social Influence (SI) scores for Python also surpassed those for Java, indicating that peer perceptions and career-oriented motivations are increasingly influential in early programming education.

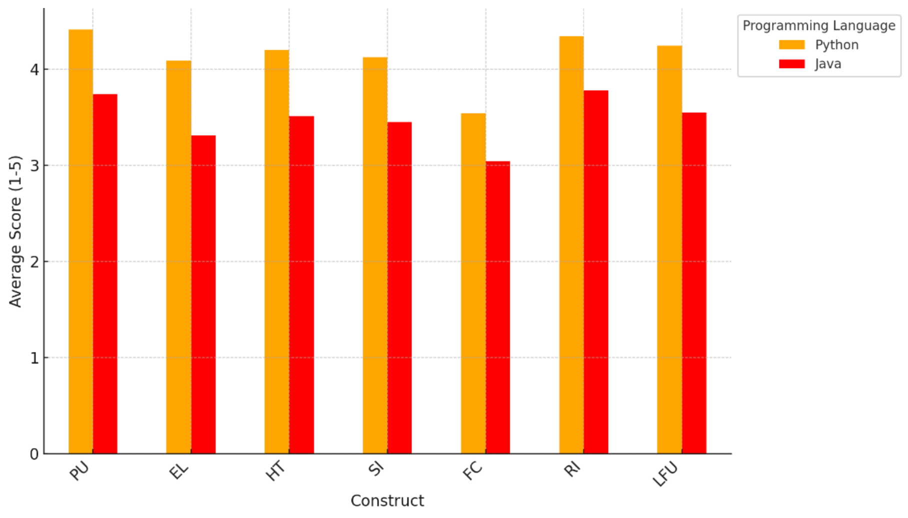

Figure 5. Comparison of Python vs Java on Key Constructs

Nevertheless, the Multi-Group Analysis (MGA) results from the study revealed no statistically significant differences in the effect of the key UTAUT2 and TAM2 constructs across Python and Java users, suggesting that the same set of psychological and contextual factors drives programming language preference regardless of the programming language. This supports the robustness and generalizability of the theoretical model applied.

Preferred Alternative Programming Languages

Programming education often begins with high-level languages like Python and Java due to their simplicity, readability, and industry relevance (Birillo et al., 2024). However, recognizing that students have different learning styles and needs, this study allowed participants to provide their preferred alternative programming languages to Python and Java. This approach offered students the opportunity to explore languages that might better suit their individual preferences and learning experiences. By incorporating a broader range of programming languages, educators can create a more inclusive and adaptable curriculum, enhancing student engagement and potentially improving their confidence and competence in the subject.

To further investigate this, the analysis included an open-ended option for respondents to mention any other languages they preferred. The resulting list of alternative languages, analyzed by frequency of mention, provided deeper insights into student preferences. The survey results revealed that C++ was the preferred alternative, with a significant number of students (over 70) selecting it. JavaScript was the second most popular language, with just over 30 students choosing it. C Programming Language followed with between 10 and 15 students selecting it, while PHP garnered exactly 10 mentions. Other languages such as COBOL, Go, Ruby, Swift, and Visual Basic were mentioned less frequently, with most languages receiving fewer than 10 mentions (see Figure 6).

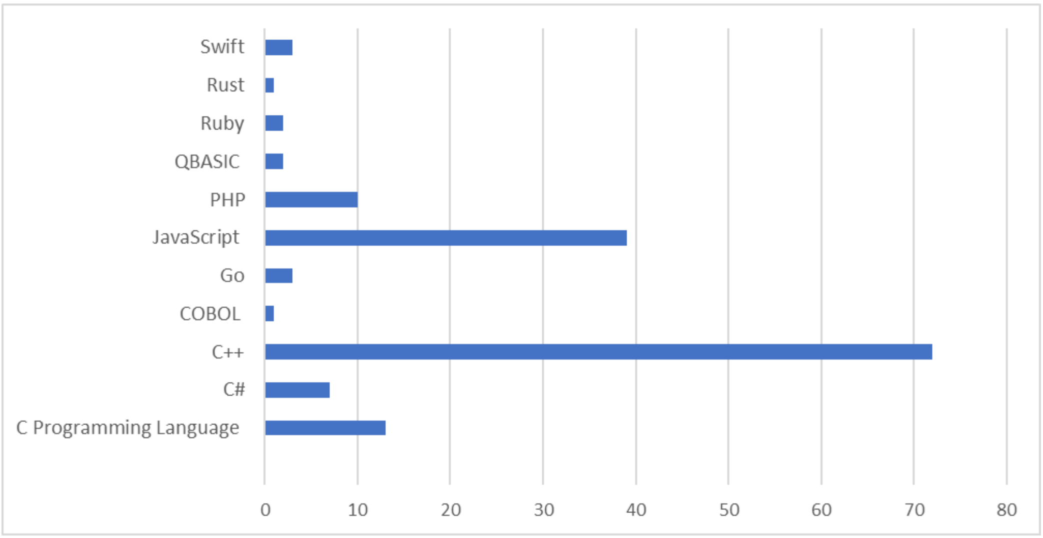

Figure 6. Frequency distribution of student’s alternative preferred programming language.

These preferences directly relate to the core construct of Programming Language Preference (PLP) in the proposed structural model. The data supports the idea that PLP is influenced by a combination of perceived usefulness, ease of use, and contextual factors, consistent with the integrated TAM2 and UTAUT2 framework discussed earlier. The prominence of languages like C++ and JavaScript suggests that students value factors beyond curriculum defaults, such as perceived industry relevance, familiarity, or alignment with personal learning goals.

Theoretical Contributions

This research contributes to the educational technology theory by developing and validating an integrated framework that merges UTAUT2 and TAM2, enhanced with domain-specific factors: Relevance to Industry (RI) and Likelihood of Future Use (LFU). While UTAUT2 addresses contextual and social influences and TAM2 focuses on cognitive assessments. This incorporation of RI and LFU strengthens the model's applicability for curriculum design in fast-changing fields like computing. The model also aligns with the Expectancy-Value Theory (EVT) (see figure 7), which reinforces the notion that students' motivation to adopt a programming language is influenced not only by ease of use and perceived usefulness but also by their belief in the language's future value and utility. Based on this revelation, the study bridges a gap in technology acceptance research by applying motivational and future oriented constructs in the context of programming language instruction which has been an area traditionally dominated by tool based and pedagogy focused models.

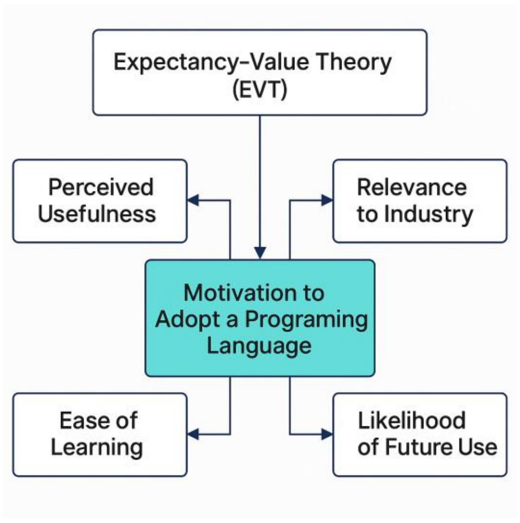

Figure 7. Motivational Factors in Programming Language Adoption (Based on EVT).

By demonstrating that both extrinsic factors, such as industry relevance and peer influence, and intrinsic elements, like ease of learning and perceived usefulness, coming together to shape students' programming choices, this study expands the scope of acceptance theories in educational contexts and establishes a foundation for further research on learner-driven curriculum design.

Practical and Pedagogical Implications

The current state of programming education continues to evolve under the pressure of rapid technological change, shifting industry demands, and student expectations. Yet, the findings of this study reveal a persistent inertia in curriculum structures, particularly in the selection of programming languages and the responsiveness of instructional design. This raises a central question: how can curriculum designers ensure programming education remains relevant, inclusive, and future-ready?

One of the most persistent challenges stems from the use of traditional programming languages, especially Java, in introductory computer science courses (Mason et al., 2024). Despite its longstanding presence in academia, Java appears increasingly misaligned with industry expectations, particularly in domains such as web development, agile environments, data analysis and machine learning, as well as with the learning needs of novice programmers (Puri, 2024). Empirical insights from this study reveal a clear preference among students for Python which is rated highly for its simplicity, readability, and industry applicability (Islam et al., 2024). This divergence invites a reconsideration of legacy language preferences in curriculum design. Prioritizing Python in introductory programming courses, can help educators create inclusive learning environments, reduce common points of frustration for beginners, and better prepare students for fast-evolving careers that demand adaptable thinking and advanced problem-solving skills (Y. Lin & Fang, 2023).

However, introducing a more accessible language alone is not sufficient without the supporting infrastructure to ensure its effective delivery. The study's findings on Facilitating Conditions (a construct from the UTAUT2 model, that measures the degree to which a student believes that sufficient organizational and technical infrastructure exists to support the adoption and effective use of a programming language) revealed a lack of institutional readiness in this regard. In many educational settings, particularly those constrained by limited resources, the availability of modern learning platforms, updated software environments, and adequately trained instructors remains a major barrier to programming language adoption (Ansari et al., 2024). This underscores the paradox that while curriculum reform may be conceptually sound, its implementation weakens in the absence of systemic support. As such, investments in digital infrastructure and professional development must be seen not as an auxiliary aspect but as a foundational and meaningful curriculum transformation.

Another under-explored but powerful implication to emerge from the findings is the importance of students' voice or perspective in shaping curriculum decisions. Often sidelined in favor of academic or administrative priorities, student perspectives offer valuable insight into lived educational experiences and career aspirations. While students may not possess expertise in curriculum design, their feedback reflects how learning translates into confidence, competence, and career readiness (Zhu et al., 2021). Integrating student input alongside academic research and industry forecasts ensures that curriculum development is grounded not only in theory and labor market analytics but also in the realities of classroom engagement.

This dialogic approach becomes even more crucial when considering how programming skills connect with future employment. The findings suggest that making current links between course content and career relevance significantly enhances student motivation and persistence. When educators articulate how fundamental concepts such as loops, functions, or data structures underpin real-world applications, such as automation, app development, or AI, the programming education classroom transforms from an abstract training ground into a launchpad for professional identity formation (Ho, 2024). Framing programming education through the lens of relevance in industry is, therefore, not merely motivational but also strategic.

Perhaps the most forward-looking implication from this study concerns the role of Generative Artificial Intelligence (GenAI) in reshaping programming pedagogy. As GenAI tools become increasingly sophisticated and accessible, their potential to scaffold learning, reduce cognitive load, and offer personalized guidance presents exciting opportunities for curriculum innovation (Prather et al., 2024). Python's compatibility with GenAI ecosystems, such as large language models, code generators, and tutoring agents, further strengthens its suitability as the gateway programming language for novice learners (Phung et al., 2023; Prather et al., 2025). The integration of GenAI, if done ethically and pedagogically, can amplify both engagement and equity in programming education.

Considering these findings, curriculum designers in programming education should take targeted actions to ensure relevance and effectiveness. This includes: adopting Python as the default introductory programming language to enhance accessibility and alignment with industry needs; investing in the institutional infrastructure and instructor training necessary to support such transitions; integrating student feedback into curriculum development to reflect real learning experiences; linking programming content to real-world applications and career pathways; and strategically integrating GenAI tools into teaching practice to personalize and scaffold learning. By moving from a static, one-size-fits-all model to a responsive, evidence-informed approach, programming education can become more inclusive, motivating, and aligned with the demands of the modern workforce.

Limitations and Future Research Directions

While this study provides valuable insights, several limitations should be acknowledged. The sample was drawn from a single institution, Modibbo Adama University, which may limit the generalizability of the findings to other universities and educational systems. Additionally, the participant pool was predominantly male (92%) and from Computer Science backgrounds (88%), potentially introducing selection bias and reducing demographic representativeness. The cross-sectional nature of the study captures student perceptions at a single point in time, without accounting for how these views evolve with experience. Future research should adopt a multi-institutional approach, implement rigorous sampling strategies, and include longitudinal studies to track programming language preferences across multiple semesters. Targeted outreach to underrepresented groups, such as non-computing students and female participants, would enhance inclusivity. Furthermore, as student perceptions may not always correlate with actual programming proficiency, future studies should integrate performance-based metrics alongside self-reported preferences to gain a more comprehensive understanding of programming language effectiveness in education.

Beyond these limitations, several promising avenues for future research exist. Investigating the role of GenAI tools in reducing learning barriers is an emerging area of interest. Structured integration of Python, Java, and C++ within programming curricula could also optimize skill development, ensuring students receive balanced exposure to different paradigms. Comparative studies across academic disciplines, age groups, and cultural contexts may provide further insights into variations in programming language adoption and learning outcomes. Additionally, pedagogical innovations such as project-based learning, peer collaboration and immersive learning environments should be explored for their potential to enhance programming instruction. Addressing these areas will contribute to refining programming language selection, improving teaching methodologies, and ultimately strengthening programming education in HEIs.

Strategic Curriculum Design for Programming Education in Developing Countries

While programming education continues to evolve globally, higher education institutions (HEIs) in developing countries face distinct challenges in aligning their curricula with both cognitive development principles and rapidly shifting technological demands. This study sheds light on several instructional gaps and pedagogical inefficiencies that limit the efficacy of programming instruction in these contexts. A careful analysis of learner preferences, language complexity, and emerging technologies underscores the need for a structured reimagining of programming curricula.

A key observation from this study is the underutilization of beginner-friendly programming languages during the foundational stages of instruction. Despite Python's global reputation for simplicity, readability, and industry relevance, it is often not prioritized in introductory courses across many HEIs (Balreira et al., 2023). This discrepancy raises concerns about the cognitive burden placed on novice programmers, many of whom struggle with syntactically dense languages. By introducing Python at the outset, institutions can significantly lower entry barriers, reduce student apprehension, and promote early engagement with core programming constructs (Jiang et al., 2024; Murugesh et al., 2024). Its increasing application across data science, AI, and web development further strengthens the argument for its early curricular inclusion (Dhandayuthapani, 2024; Mehare et al., 2023).

In contrast, languages such as Java, though powerful and widely adopted, present a steeper learning curve due to their verbosity and complexity. Yet, completely excluding Java from curricula would be pedagogically shortsighted. Instead, this study advocates for a deliberate repositioning of Java to later stages of instruction, such as in advance stages of object-oriented programming or software engineering modules. This allows students to encounter Java at a point when their computational thinking has matured, increasing their ability to grasp abstraction, inheritance, and modular design patterns without being overwhelmed. Pedagogical sequencing, therefore, becomes a critical design principle in curriculum restructuring.

Moreover, the strong preference expressed for C++ among learners in this study cannot be overlooked. As a middle-level language, C++ provides unique affordances for developing deeper conceptual understanding by bridging high-level programming with low-level memory and performance management tasks (Heller et al., 2016). Integrating such languages into the curriculum can sharpen students' problem-solving abilities and better prepare them for specialized fields like systems programming and embedded software development (Zhao, 2021).

The advent of GenAI presents both a challenge and an opportunity for programming pedagogy. Despite its growing use in code generation and as an instructional feedback tool, the educational sector in many developing countries has yet to explore its full potential. GenAI tools offer promising avenues for adaptive, personalized, and interactive learning environments, features that can significantly enhance student engagement and mastery when appropriately integrated (Zastudil et al., 2023). However, without structured research into its pedagogical implications, integration may remain sporadic without any strategy. Future inquiry must focus on frameworks for responsible GenAI adoption, balancing innovation with instructional integrity.

All together, these insights offer a roadmap for transforming programming education in HEIs within developing countries. A curriculum that is cognitively aligned, technologically relevant, and progressively scaffolded will not only improve student outcomes but also prepare graduates to thrive in dynamic digital economies. Moving forward, collaboration between curriculum designers, industry experts, and educational technologists will be essential in bringing these recommendations to life.

Ethical Considerations

This study adhered to strict ethical guidelines and complied with the Nigeria Data Protection Act (NDPA) (Federal Government of Nigeria, 2023), to ensure participant privacy and data security. Participation was entirely voluntary, with informed consent obtained from all respondents before data collection. In accordance with lawful processing principles, no personal identifiable information was collected, and all responses were fully anonymized to protect participant identities. To maintain data security, responses were collected using Google Forms, ensuring no email addresses were recorded. Data access was restricted to authorized researchers, and stored responses were protected within Google's encrypted cloud infrastructure. However, in line with storage limitation requirements, the data will be retained only for the duration necessary for research purposes and permanently deleted thereafter. To further enhance security, exported anonymized responses are stored in a password-protected local storage before final deletion from Google's cloud.

Special care was taken to ensure that demographic data did not compromise participant anonymity. Additionally, all academic performance-related responses were analyzed in aggregate to prevent individual identification. These measures align with international ethical standards in educational research, ensuring transparency, confidentiality, and participant trust while adhering to Nigeria's data protection regulations.

Conclusion

This study highlights the growing capacity, availability and influence of Python as the preferred language for beginner-level programming education in Nigerian higher education institutions. Its perceived ease of learning, usefulness, and relevance to industry make it an ideal choice for introductory programming courses, while Java remains valuable as a strategically integrated programming language in curricula for later or advance stages of programming education. This will provide students with a well-rounded programming foundation. Furthermore, as programming education evolves, institutions should consider adopting hybrid approaches that balance pedagogical effectiveness with industry demands, strategically incorporating both Python and Java.

Additionally, exploring the role of GenAI in programming instruction presents an opportunity to revolutionize how students can learn and be taught how to code. GenAI can improve programming education by making it more personalized, accessible, and engaging while also developing student's higher-order thinking skills (Becker et al., 2023). GenAI serves as a cognitive scaffold, providing real-time feedback, examples, and explanations. By adapting to each student's learning pace and needs, it tailors their educational experience, ensuring that students can progress at their own speed while receiving the support they need to master complex programming concepts. This personalized approach not only aids in immediate learning but also encourages critical thinking and problem-solving, essential components of higher-order thinking.

By aligning curricula with both student preferences and technological trends, educational institutions can create future-ready programming education that not only meets industry standards but also builds an engaging and effective learning experience for learners. Future research could further investigate the role of emerging technologies like GenAI in shaping programming pedagogy and student engagement, ensuring that programming instruction remains adaptive to the evolving needs of both learners and the industry.

Statements and Declarations

Acknowledgements: The authors would like to thank the students who participated in the survey, and the computer science departments at Modibbo Adama University for their approval and support to conduct the data collection.

Availability of Data and Materials: The datasets collected and analyzed during the current study are not publicly available due to privacy and ethical considerations but are available from the corresponding author on reasonable request.

Funding: This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

References

Abbas, S., Latif, S., & Khoso, F. J. (2023). Relationship between Students' Cognitive Abilities and Cumulative Grade Point Average at University Level. *Global Educational Studies Review*, *VIII*(I), 438–444. https://doi.org/10.31703/gesr.2023(VIII-I).38

Ansari, M., Waris, S., & Zara, C. (2024). Barriers to Educational Technology Adoption: Navigating Challenges in Integration. *Qlantic Journal of Social Sciences*, *5*(3), 240–247. https://doi.org/10.55737/qjss.123732538

Asgari, M., Tsai, F.-C., Mannila, L., Strömbäck, F., & Sadique, K. M. (2024). Students' perspectives on using digital tools in programming courses: A cross country case study between Sweden and Taiwan. *Discover Education*, *3*(1), 57. https://doi.org/10.1007/s44217-024-00144-4

Balreira, D. G., Silveira, T. L. T. D., & Wickboldt, J. A. (2023). Investigating the impact of adopting Python and C languages for introductory engineering programming courses. *Computer Applications in Engineering Education*, *31*(1), 47–62. https://doi.org/10.1002/cae.22570

Becker, B. A., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Prather, J., & Santos, E. A. (2023). Programming Is Hard - Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation. *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, 500–506. https://doi.org/10.1145/3545945.3569759

Birillo, A., Tigina, M., Kurbatova, Z., Potriasaeva, A., Vlasov, I., Ovchinnikov, V., & Gerasimov, I. (2024). Bridging Education and Development: IDEs as Interactive Learning Platforms. *Proceedings of the 1st ACM/IEEE Workshop on Integrated Development Environments*, 53–58. https://doi.org/10.1145/3643796.3648454

BusinessDay. (2024). *Only 22% of STEM graduates are females in nigeria – FITC*. https://businessday.ng/news/article/only-22-of-stem-graduates-are-females-in-nigeria-fitc/

Chakraborty, P., Shahriyar, R., Iqbal, A., & Uddin, G. (2021). How do developers discuss and support new programming languages in technical Q&A site? An empirical study of Go, Swift, and Rust in Stack Overflow. *Information and Software Technology*, *137*, 106603. https://doi.org/10.1016/j.infsof.2021.106603

Dela Rosa, A. P. (2023). Effectiveness of an Online Course in Programming in a State University in the Philippines. *International Journal of Computing Sciences Research*, *7*, 1685–1698. https://doi.org/10.25147/ijcsr.2017.001.1.127

Demir, F. (2022). The effect of different usage of the educational programming language in programming education on the programming anxiety and achievement. *Education and Information Technologies*, *27*(3), 4171–4194. https://doi.org/10.1007/s10639-021-10750-6

Dhandayuthapani, B. V. (2024). Enhancing Jakarta Faces Web App with AI Data-Driven Python Data Analysis and Visualization. *International Journal of Information Technology and Computer Science*, *16*(5), 36–51. https://doi.org/10.5815/ijitcs.2024.05.03

Dobslaw, F., Angelin, K., Öberg, L.-M., & Ahmad, A. (2023). *The Gap between Higher Education and the Software Industry—A Case Study on Technology Differences* (Version 1). arXiv. https://doi.org/10.48550/ARXIV.2303.15597

Eccles, J. S., & Wigfield, A. (2023). Expectancy-value theory to situated expectancy-value theory: Reflections on the legacy of 40+ years of working together. *Motivation Science*, *9*(1), 1–12. https://doi.org/10.1037/mot0000275

Edeh, M., Ugboaja, S., Ugwuja, N., Igwe, J., Daniel, I., & Richard-Nnabu, N. (2022). Smartphone usage among computer science students in higher education during covid-19 lockdown. *Journal of Computer Science and Its Application*, 20–26.

Enudi, O. L., & Umoeshiet E. Akpan. (2023). *Availability And Utilization Of Instructional Resources On Academic Performance Of Business Education Students In Tertiary Institutions In Delta State*. https://doi.org/10.5281/ZENODO.10208554

Eteng, I., Akpotuzor, S., Akinola, S. O., & Agbonlahor, I. (2022). A review on effective approach to teaching computer programming to undergraduates in developing countries. *Scientific African*, *16*, e01240. https://doi.org/10.1016/j.sciaf.2022.e01240

Federal Government of Nigeria. (2023). *Nigeria data protection act (NDPA), 2023*. https://cert.gov.ng/ngcert/resources/Nigeria_Data_Protection_Act_2023.pdf

Fulton, K. R., Chan, A., Votipka, D., Hicks, M., & Mazurek, M. L. (2021). Benefits and drawbacks of adopting a secure programming language: Rust as a case study. *Seventeenth Symposium on Usable Privacy and Security (SOUPS 2021)*, 597–616. https://www.usenix.org/conference/soups2021/presentation/fulton

Hair, J., & Alamer, A. (2022). Partial Least Squares Structural Equation Modeling (PLS-SEM) in second language and education research: Guidelines using an applied example. *Research Methods in Applied Linguistics*, *1*(3), 100027. https://doi.org/10.1016/j.rmal.2022.100027

Haroud, S., & Saqri, N. (2025). Generative AI in Higher Education: Teachers' and Students' Perspectives on Support, Replacement, and Digital Literacy. *Education Sciences*, *15*(4), 396. https://doi.org/10.3390/educsci15040396

Heller, T., Kaiser, H., Diehl, P., Fey, D., & Schweitzer, M. A. (2016). Closing the Performance Gap with Modern C++. In M. Taufer, B. Mohr, & J. M. Kunkel (Eds.), *High Performance Computing* (Vol. 9945, pp. 18–31). Springer International Publishing. https://doi.org/10.1007/978-3-319-46079-6_2

Ho, A. (2024). *Linking Scholarly Learning Activities and Professional Identity in OTD Students* [University of Nevada, Las Vegas]. https://doi.org/10.34917/37650832

Iftikhar, S., Guerrero-Roldán, A.-E., & Mor, E. (2022). Practice Promotes Learning: Analyzing Students' Acceptance of a Learning-by-Doing Online Programming Learning Tool. *Applied Sciences*, *12*(24), 12613. https://doi.org/10.3390/app122412613

Ishaq, K., & Alvi, A. (2023). *Personalization, Cognition, and Gamification-based Programming Language Learning: A State-of-the-Art Systematic Literature Review* (Version 1). arXiv. https://doi.org/10.48550/ARXIV.2309.12362

Islam, M. T., Islam, M. R., Jhilik, R. A., Islam, M. A., Raihan, P. M. S., Faruque, M. S., & Shahjahan, A. M. (2024). A Comparative Analysis of Programming Language Preferences Among Computer Science and Non-Computer Science Students. *European Journal of Theoretical and Applied Sciences*, *2*(3), 900–912. https://doi.org/10.59324/ejtas.2024.2(3).70

Jain, M. P., Soni, H., Singh, A. P., Mehta, P., & Babani, N. (2024). *Comparative review of python and C: Choosing the right language for beginners*.

Jiang, Y., Wu, H., Yu, X., & Ji, T. (2024). A study of factors influencing programming anxiety among non-computer students. *Proceedings of the 2024 9th International Conference on Information and Education Innovations*, 63–69. https://doi.org/10.1145/3664934.3664956

Joshi, A., & Tiwari, H. (2023). An Overview of Python Libraries for Data Science. *Journal of Engineering Technology and Applied Physics*, *5*(2), 85–90. https://doi.org/10.33093/jetap.2023.5.2.10

Keuning, H., Alpizar-Chacon, I., Lykourentzou, I., Beehler, L., Köppe, C., De Jong, I., & Sosnovsky, S. (2024). Students' Perceptions and Use of Generative AI Tools for Programming Across Different Computing Courses. *Proceedings of the 24th Koli Calling International Conference on Computing Education Research*, 1–12. https://doi.org/10.1145/3699538.3699546

Kruglyk, V., Dovhopiatyi, A., Tesliuk, N., & Strykhar, A. (2012). Selecting programming language for teaching students in technical universities. In A. Petrenko, H. C. Mayr, & D. A. Pospelov (Eds.), *Proceedings of the 8th international conference on ICT in education, research and industrial applications (ICTERI 2012)* (Vol. 848, pp. 188–198). CEUR-WS.org. http://ceur-ws.org/Vol-848/ICTERI-2012-CEUR-WS-paper-37-p-188-198.pdf

Lin, J. (2022). The effects of gamification instruction on the roles of perceived ease of learning, enjoyment, and useful knowledge toward learning attitude. *Turkish Online Journal of Educational Technology-TOJET*, *21*(2), 81–91.

Lin, Y., & Fang, L. (2023). Exploring Innovations in Teaching Reform for Python Programming Under Engineering Education Accreditation: *Proceedings of the 2nd International Seminar on Artificial Intelligence, Networking and Information Technology*, 355–361. https://doi.org/10.5220/0012283900003807

Ling, H.-C., Hsiao, K.-L., & Hsu, W.-C. (2021). Can Students' Computer Programming Learning Motivation and Effectiveness Be Enhanced by Learning Python Language? A Multi-Group Analysis. *Frontiers in Psychology*, *11*, 600814. https://doi.org/10.3389/fpsyg.2020.600814

Mahalaxmi, G., Donald, A. D., & Srinivas, T. A. S. (2023). A Short Review of Python Libraries and Data Science Tools. *South Asian Research Journal of Engineering and Technology*, *5*(1), 1–5. https://doi.org/10.36346/sarjet.2023.v05i01.001

Mason, R., Simon, Becker, B. A., Crick, T., & Davenport, J. H. (2024). A Global Survey of Introductory Programming Courses. *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, 799–805. https://doi.org/10.1145/3626252.3630761

Mehare, H. B., Anilkumar, J. P., & Usmani, N. A. (2023). The Python Programming Language. In M. 'Sufian' Badar (Ed.), *A Guide to Applied Machine Learning for Biologists* (pp. 27–60). Springer International Publishing. https://doi.org/10.1007/978-3-031-22206-1_2

Murugesh, T. S., Vasudevan, S. K., & Pulari, S. R. (2024). *Python: A Practical Learning Approach* (1st ed.). CRC Press. https://doi.org/10.1201/9781032712673

Nguyen, A., Hong, Y., Dang, B., & Huang, X. (2024). Human-AI collaboration patterns in AI-assisted academic writing. *Studies in Higher Education*, *49*(5), 847–864. https://doi.org/10.1080/03075079.2024.2323593

Panchbudhe, S., Shaikh, S., Swami, H., Kadam, C. Y., Padalkar, R., Shivkar, R. R., Gulavani, G., Gulajkar, S., Gawade, S., & Mujawar, F. (2024). Efficacy of Google Form–based MCQ tests for formative assessment in medical biochemistry education. *Journal of Education and Health Promotion*, *13*(1).

https://doi.org/10.4103/jehp.jehp_981_23

Parveen, K., Phuc, T. Q. B., Alghamdi, A. A., Hajjej, F., Obidallah, W. J., Alduraywish, Y. A., & Shafiq, M. (2024). Unraveling the dynamics of ChatGPT adoption and utilization through Structural Equation Modeling. *Scientific Reports*, *14*(1), 23469. https://doi.org/10.1038/s41598-024-74406-4

Penney, E. K., Agyei, J., Boadi, E. K., Abrokwah, E., & Ofori-Boafo, R. (2021). Understanding Factors That Influence Consumer Intention to Use Mobile Money Services: An Application of UTAUT2 With Perceived Risk and Trust. *Sage Open*, *11*(3), 21582440211023188. https://doi.org/10.1177/21582440211023188

Perera, P., Tennakoon, G., Ahangama, S., Panditharathna, R., & Chathuranga, B. (2021). A Systematic Mapping of Introductory Programming Languages for Novice Learners. *IEEE Access*, *9*, 88121–88136. https://doi.org/10.1109/ACCESS.2021.3089560

Phung, T., Pădurean, V.-A., Cambronero, J., Gulwani, S., Kohn, T., Majumdar, R., Singla, A., & Soares, G. (2023). Generative AI for Programming Education: Benchmarking ChatGPT, GPT-4, and Human Tutors. *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 2*, 41–42. https://doi.org/10.1145/3568812.3603476

Prather, J., Leinonen, J., Kiesler, N., Gorson Benario, J., Lau, S., MacNeil, S., Norouzi, N., Opel, S., Pettit, V., Porter, L., Reeves, B. N., Savelka, J., Smith, D. H., Strickroth, S., & Zingaro, D. (2025). Beyond the Hype: A Comprehensive Review of Current Trends in Generative AI Research, Teaching Practices, and Tools. *2024 Working Group Reports on Innovation and Technology in Computer Science Education*, 300–338. https://doi.org/10.1145/3689187.3709614

Prather, J., Reeves, B. N., Leinonen, J., MacNeil, S., Randrianasolo, A. S., Becker, B. A., Kimmel, B., Wright, J., & Briggs, B. (2024). The Widening Gap: The Benefits and Harms of Generative AI for Novice Programmers. *Proceedings of the 2024 ACM Conference on International Computing Education Research - Volume 1*, 469–486. https://doi.org/10.1145/3632620.3671116

Puri, A. (2024). Skills Mismatch in Entry-level Programmer Positions: Employer Expectations vs. Observations in Lalitpur, Nepal. *Journal of Education and Research*, *14*(1), 1–135. https://doi.org/10.51474/jer.v14i1.733

Quintero-Manes, R., & Vieira, C. (2024). Differentiated measurement of cognitive loads in computer programming. *Journal of Computing in Higher Education*. https://doi.org/10.1007/s12528-024-09411-7

Ringle, C. M., Wende, S., & Becker, J.-M. (2024). *SmartPLS 4* [Manual]. SmartPLS GmbH.

Romao, L., Kalinowski, M., Barbosa, C., Araújo, A. A., Barbosa, S. D. J., & Lopes, H. (2024). *Agile Minds, Innovative Solutions, and Industry-Academia Collaboration: Lean R&amp;D Meets Problem-Based Learning in Software Engineering Education* (Version 1). arXiv. https://doi.org/10.48550/ARXIV.2407.15982

Rudhumbu, N. (2022). Applying the UTAUT2 to predict the acceptance of blended learning by university students. *Asian Association of Open Universities Journal*, *17*(1), 15–36. https://doi.org/10.1108/AAOUJ-08-2021-0084

Sandoval-Medina, C., Arevalo-Mercado, C., Munoz-Andrade, E., & Munoz-Arteaga, J. (2024). Self-Explanation Effect of Cognitive Load Theory in Teaching Basic Programming. *Journal of Information Systems Education*, *35*(3), 303–312. https://doi.org/10.62273/GMIV1698

Schoeffel, P., Ramos, V. F. C., Cechinel, C., & Wazlawick, R. S. (2021). The Expectancy-Value-Cost Light Scale to Measure Motivation of Students in Computing Courses. *Informatics in Education*. https://doi.org/10.15388/infedu.2022.04

Shahzad, M. F., Xu, S., Khan, K. I., & Hasnain, M. F. (2023). Effect of social influence, environmental awareness, and safety affordance on actual use of 5G technologies among Chinese students. *Scientific Reports*, *13*(1), 22442. https://doi.org/10.1038/s41598-023-50078-4

Siegfried, R. M., Herbert-Berger, K. G., Leune, K., & Siegfried, J. P. (2021). Trends Of Commonly Used Programming Languages in CS1 And CS2 Learning. *2021 16th International Conference on Computer Science & Education (ICCSE)*, 407–412. https://doi.org/10.1109/ICCSE51940.2021.9569444

Singh, D., & Rajendran, R. (2024). Cognitive engagement as a predictor of learning gain in Python programming. *Smart Learning Environments*, *11*(1), 58. https://doi.org/10.1186/s40561-024-00330-9

Sobral, S. R. (2021). The Old Question: Which Programming Language Should We Choose to Teach to Program? In T. Antipova (Ed.), *Advances in Digital Science* (Vol. 1352, pp. 351–364). Springer International Publishing. https://doi.org/10.1007/978-3-030-71782-7_31

Sosale, S., Harrison, G. M., Tognatta, N., & Nakata, S. (2023). *Engendering access to STEM education and careers in south asia*. World Bank Publications.

Tey, T. C. Y., & Moses, P. (2018). UTAUT: Integrating Achievement Goals and Learning Styles for Undergraduates' Behavioural Intention to Use Technology. *EAI Endorsed Transactions on E-Learning*, *5*(17), 155573. https://doi.org/10.4108/eai.25-9-2018.155573

Venkatesh, Thong, & Xu. (2012). Consumer Acceptance and Use of Information Technology: Extending the Unified Theory of Acceptance and Use of Technology. *MIS Quarterly*, *36*(1), 157. https://doi.org/10.2307/41410412

White, A. (2015). Students' Familiarity With Google Applications for Education at a Thai University. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.3311329

Zastudil, C., Rogalska, M., Kapp, C., Vaughn, J., & MacNeil, S. (2023). Generative AI in Computing Education: Perspectives of Students and Instructors. *2023 IEEE Frontiers in Education Conference (FIE)*, 1–9. https://doi.org/10.1109/FIE58773.2023.10343467

Zhao, Y. (2021). Research on Application of Computer Recognition Technology in C Language Programming Modeling System. *Journal of Physics: Conference Series*, *2083*(4), 042024. https://doi.org/10.1088/1742-6596/2083/4/042024

Zhu, G., Raman, P., Xing, W., & Slotta, J. (2021). Curriculum design for social, cognitive and emotional engagement in Knowledge Building. *International Journal of Educational Technology in Higher Education*, *18*(1), 37. https://doi.org/10.1186/s41239-021-00276-9