Research Article

Empirical Comparison of C++ And Python For Teaching Introductory Programming Students and Their Impact On Learning Outcomes

Junaid Asghar [1], Shaista Habib[2], Ali Hussain[3]*, Muhammad Majid Hussain[4]

[1, 3, 4] Department of Computer Science and IT, The University of Lahore, Lahore Pakistan

[2] Department of Artificial Intelligence, University of Management and Technology, Lahore Pakistan

*Corresponding Author: Junaid Asghar Email: mjasghar52@gmail.com

Received: January 18, 2025 **Accepted:** April 10, 2025 **Published:** April 14, 2025

Abstract: Programming is at the bedrock of the computing discipline and the decision between languages of choice to be studied as first programming language lays the critical aspect of the learning of the fundamental programming concepts, coding standards, and problem-solving methods. In the context of a rapidly developing technology, it is necessary to reconcile pedagogical efficiency in academic training as well as with the trends in the industry. C++ has enjoyed a long history as a dominant programming language in academia and industry but doubts are being raised that it is not the most effective language to be learnt as a beginner due to its perhaps more complex code than other languages such as Python. This paper is comparative research on C++ and Python as the introductory languages in undergraduate studies. An evaluation of the literature and student performance and engagement through survey allows us to conclude that Python, with its simplicity, readability, syntactic regularity, orthogonality, and greater degree of abstraction, is likely to produce higher learning outcomes and engagement even in novice programmers than C++. The paper concludes that Python (as a programming language) is potentially more effective in teaching elementary programming, but that also it is necessary to consider the goals of a curriculum and industry practices.

Keywords: C++ and Python, Introductory Programming Language, Learner's Perspective

1. Introduction

The appropriate programming language to be used in the beginning of the programming courses has been the most common or disputed question around the globe. Python has gained popularity as a student introductory programming language in many international institutions e.g. the University of Hertfordshire UK, the Royal Melbourne Institute of Technology (RMIT) Australia and Massachusetts Institute of Technology (MIT). It is also imperative that the teacher teaching a particular language is someone who has left a good impression among the learners that emboldens them to keep learning, and to pursue the programming as a possible career option [1].

The principal objective is to introduce students to the basics of effective problem-solving to deliver solutions in clear and concise manner [4][5]. The introductory programming language to learners should have a high flexibility in solving daily life challenges that would enable the learner to adopt the projects and industry needs easily [6]. The language that will probably be used should be capable of exhibiting good work in relation to the simplicity of the programming language, orthogonal, semantic, regular, turnaround time, syntactic and debugging.

The teaching of programming in Pakistan started at the school level and is available at all levels of the doctorate in education. Although a few students are already familiar with programming languages while enrolled in computing-related degrees which has a positive impact on their learning of coding and language techniques, most of the students are not familiar with programming languages [4]. So, for non-familiar students, there is a complete procedure, and process to learn the programming language from scratch. This stage required the debate of choosing the right introductory language so that he/she may perform well in problem-solving, their courses, and specifically in the industry for reliability in the computing market [7].

When it comes to choosing a programming language to teach introductory computer science courses, both C++ and Python are good options. However, the choice depends on the course's learning objectives and the student's background. Python is an excellent language to learn to start introductory courses on computer science due to its straightforwardness, legibility, and usability. The syntax used in Python is simple and this simplicity inspired beginners to learn and comprehend different programming concepts. Python also has a great collection of modules that make it simplistic to perform tasks such as data analysis, web scraping, and machine learning. C++ is the powerful programmable language that is widely applied in system programming, games and other programs which require high performance. It is more complex than Python and can take longer to learn, but it provides a better understanding of computer architecture, memory management, and other low-level concepts. So, if the goal is to teach basic programming concepts, Python is a better choice. However, if the course focuses on system programming or performance optimization, then C++ might be more appropriate. The aim of this paper is to evaluate the success of using Python and C++ to introduce basic concepts (Conditional Structures, Loop Structures, Debugging, and use of libraries, etc). Such evaluation is based on the analysis of student assessments [7].

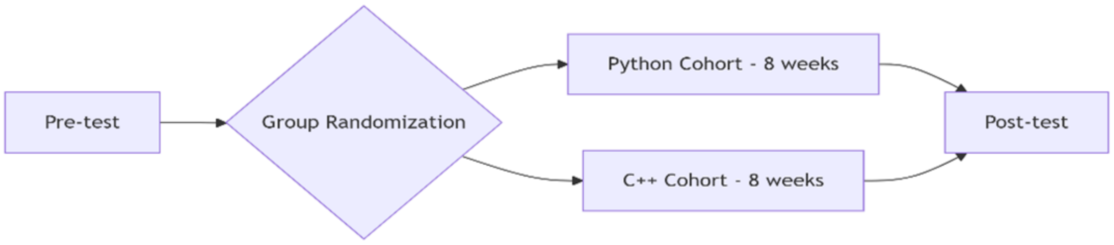


Figure 1 Working flow of assessment taken

To measure the success of the proposed method, we present it to highlight the importance to develop an experimental setting for a fair comparison and the importance to control the flow of execution of programming language carefully. The experimental setting used in the survey-based paper is the survey-based analysis of a session of BSCS students in the same session but different sections that are taught by the same teacher with different programming languages. There are two sections for first-semester students, one is taught in Python and the other one is taught in C++.

2. Proposed Methodology

In contemporary computer science education, selecting the most effective programming language for beginners is a foundational decision that impacts student engagement, comprehension, and long-term success. Within this paper, Python assessment submissions are compared against C++ submissions to evaluate differences in student performance, cognitive load, and programming proficiency development. Since the structure and learning outcomes of the Python and C++ assessments were nearly identical, any observed performance differences could primarily be attributed to the programming languages themselves rather than task variation. This methodological consistency enables a reliable comparison of the cognitive and practical challenges that each language poses for learners.

Both sets of assessments required students to engage with a variety of core programming tasks: implementing basic logic (e.g., arithmetic operations), predicting output for given code segments, and identifying and correcting syntax and logical errors in faulty programs. Variables, conditional statements, loops, functions, and fundamental data structures are just a few of the important concepts covered in the exercises for beginners in programming. Importantly, the format and difficulty level of the questions were held constant across both languages [7]. The survey collected data on the following factors [5].

- Ease of learning: How easy is it for beginners to learn the language?
- Technical Features: Does the language have necessary technical features for modern programming?
- AI analysis accuracy: How accurate is the language in implementing AI algorithms?

The following comparisons have been carried out. The grades of the submitted program are a measure of the success of the student's ability to implement programs. Bugs are interpreted as a measure of their overall understanding of programming. The combined results of performance assessments and student feedback suggest that Python offers significant advantages as a first programming language. Its concise syntax, readability, and forgiving execution environment reduce the barrier to entry and create a positive first experience for students. Importantly, this initial success correlates with improved engagement, confidence, and retention in computing disciplines. Educators are therefore encouraged to introduce Python in foundational courses to help students build core programming logic without being overwhelmed by language idiosyncrasies.

On the other hand, whereas C++ can be useful in learning low-level computing concepts and system- level programming, it is perhaps overkill to use in initial or basic-level courses where students are still gaining initial skills. The performance of the students can be enhanced and the initial frustration discouraged by postponing their introduction to C++ until they have mastered the skills of debugging and thus abstract thinking. This comparative analysis of the research will have implications on the bigger academic planning and alignment with the industry. Programming fluency, in turn, is becoming a graduate competency as AI, data science and software engineering remain advanced and progressive disciplines. The prevalence of Python in the world of data-driven and artificial intelligence makes it a valuable language in terms of both higher education and employment. Yet, on such areas as embedded systems, games, and performance-sensitive applications, C++ will not die away any time soon.

Thus, academic programs might adopt a tiered approach, beginning with Python to instill algorithmic thinking and transitioning to C++ to strengthen computational rigor and low-level understanding. This ensures that graduates are well-rounded, able to adapt across various software development paradigms and meet the diverse demands of the technology industry. They are indicators of students' use of conditions, loops, sub-programming, and use of header files (the library files). Two different academic sections of the same session have been used to collect data. During the Fall-2024 academic session, the approach to emphasize the principles of programming and design using C++ from the very beginning is followed and the approach to first teach the basic programming concepts (loops, sub-programming, and use of libraries) using Python and then move on to using C++ is followed. Using a mixed-method approach, this study compares and assesses the accessibility and learning effectiveness of C++ and Python among beginning programmers by combining quantitative and qualitative data. The participants were split evenly between the Python and C++ learning groups.

Table 1 Description of variable and their types

Variable	Type	Description
Programming Language	Independent	C++ or Python
Learning Curve	Dependent	Measured by progress rate and comprehension speed
Syntax Understanding	Dependent	Survey and test-based
Debugging Efficiency	Dependent	Errors per hour, success rate of fixes
Final Project Time	Dependent	Measured in hours
Perceived Difficulty	Dependent	Rated on 5-point Likert scale

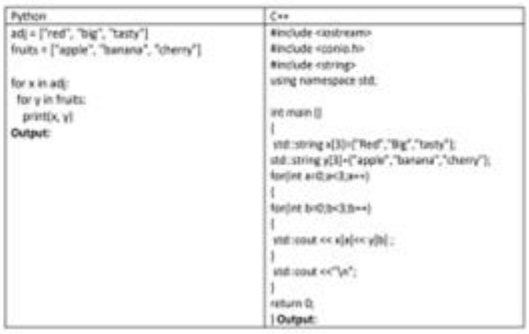
Using a mixed-method approach, this study compares and assesses the accessibility and learning effectiveness of C++ and Python among beginning programmers by combining quantitative and qualitative data. The participants were split evenly between the Python and C++ learning groups. SPSS for statistical testing and Python (Pandas, Seaborne) were used for quantitative data analysis. Excel and Matplotlib were among the visualization tools used to produce comparative graphs. The Python group used Google Colab for code execution and logging, whereas the C++ group used Code: Blocks. In order to collect learning behavior analytics, including the number of attempts and syntax errors, debug logs and compile/run statistics were tracked. For the first score analysis, descriptive statistics (mean, median, and standard deviation) were employed. Independent sample t-tests were performed for every task category in order to ascertain statistical significance in performance differences. 95% confidence intervals were upheld. To evaluate the size of the observed differences, effect sizes (Cohen's d) were computed. Furthermore, data variability across performance parameters was depicted using box plots and heat maps. The quasi-experimental design restricts full control over confounding factors like individual learning styles, instructor nuances, and tutoring. The findings may not apply to intermediate or advanced programming levels because the research is also limited to introductory programming. Another limitation includes the use of different IDEs, which, while minor, might influence debugging experiences. To ensure construct validity, tasks and assessment tools were aligned with standard programming outcomes. A panel of programming educators reviewed the evaluation questions to confirm their relevance and fairness. Reliability was maintained through standardized rubrics and cross-evaluation by multiple graders for final project assessments. Additionally, triangulation of data sources (scores, feedback, observation) enhanced the study's credibility and internal consistency.

3. Results and Discussion

In both cases, students were assessed after approximately ten weeks of teaching. C++ assessments are from section A of Fall-2024, whereas Python assessments are from section B of Fall-2024. The research is based on the 200+ students from both sections and an assumption that students are related to similar academic levels. This assumption is supported by the fact that the admission criteria for both sections were the same. We obtain a score of 60 to perform the analysis on their scores in different modules of programming languages like condition structures, loop structures, and list/array structures. The following graph shows the performance of the students of Section A (C++) and Section B (Python).Some EDA and Data Visualizations (Comparisons of results obtain) are discussed below:

The mean of the values of Section-A C++ is 4.58 for data of 60 samples while Section-B Python has 6.76 (as in the attached file of Google Colab and sample questions for evaluation are as above).

The other comparison is obtaining scores of debugging (error detection), conditional structure (if, if else statements), loop structure (for loop (addition, multiplication, etc.), and List/Array structure (contiguous memory locations to store data) programs. The mean of the values of Section-A C++ is 5.01 for data of 60 samples while Section-B Python has 6.76. The mean of values of Section-A C++ is 4.300 for data of 60 samples while Section-B Python has 6.683. The mean of the values of Section-A C++ is 5.68 for data of 60 samples while Section-B Python has 6.90 (as in the attached file of Google Colab) also for nested loop structure the mean value for C++ is 4.33 and Python is 6.95.



Python

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
Output:
```

C++

```
#include <iostream>
#include <conio.h>
#include <string>
using namespace std;

int main ()
{
 std::string x[3]={"Red","Big","tasty"};
std::string y[3]={"apple","banana","cherry"};
for(int a=0;a<3;a++)
{
for(int b=0;b<3;b++)
{
 std::cout << x[a]<< y[b] ;
}
 std::cout <<"\n";
}
return 0;
} Output:
```

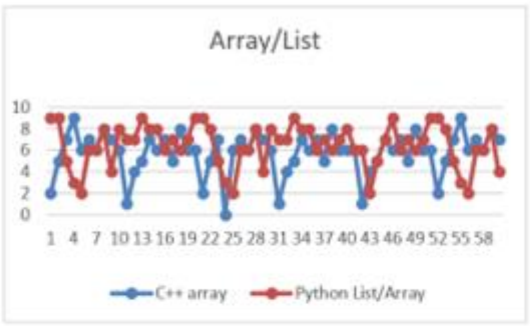
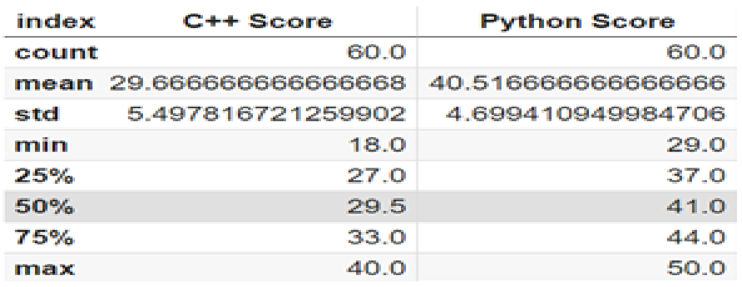


Figure 2 Sample array question and the students' performance chart

The mean of the values of Section-A C++ is 5.75 for data of 60 samples while Section-B Python has 6.45

The basic statistics on the obtained scores from the student evaluation are as follows:

We discuss the results from our survey regarding the choice of the first programming language. [1] It must be considered that students studied Python and studying C++ with no assumed background in programming. So after 10 weeks/1 of the semester, students perform well in python as compared to C++ in terms of the questionnaire and interviews based on concepts of the simple calculating programs.



index	C++ Score	Python Score
count	60.0	60.0
mean	29.666666666666668	40.516666666666666
std	5.497816721259902	4.699410949984706
min	18.0	29.0
25%	27.0	37.0
50%	29.5	41.0
75%	33.0	44.0
max	40.0	50.0

Figure 3 Score of student regarding both programming languages C++ and Python

The above results clearly show that the course that is to teach students the fundamentals of computer science and programming, Python may be a better choice because the average/mean of C++ students is 29.66 and Python has a mean value of 40.51 due to its ease of use and versatility.

The student (novice) performs well in Python because Python has a large and active community with many libraries and tools available for various applications, making it a good choice for students interested in these fields because Python's syntax is simple and concise, and it is often used in data analysis, web scraping, machine learning, and web development. The survey results indicate that Python is the most suitable programming language as a first programming language for beginners at the bachelor's level.

3.1 **Research factors of comparison C++ and Python programming language**

Here are the results for each factor:

1.Ease of learning: 84% of the respondents found Python easier to learn than C++.

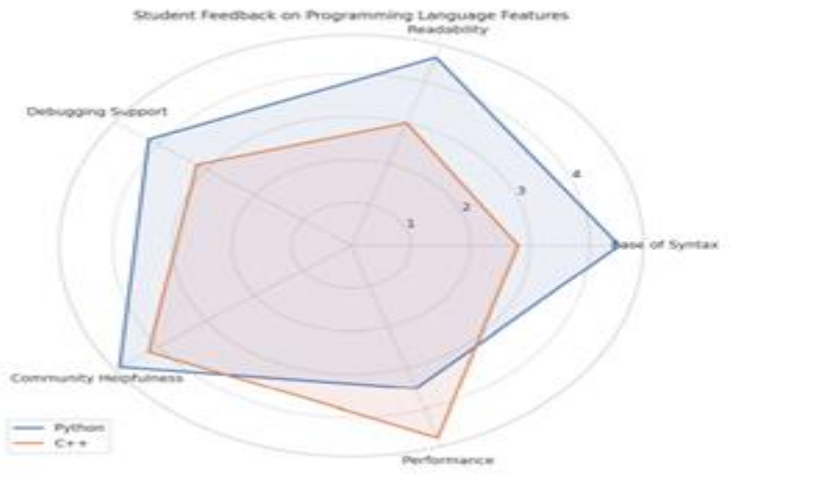


Figure 4 Radar Chart : Student Feedback on Programming Language Features

2. Technical features: 72% of the respondents found both languages equally capable, while 28% thought Python was better suited for modern programming.

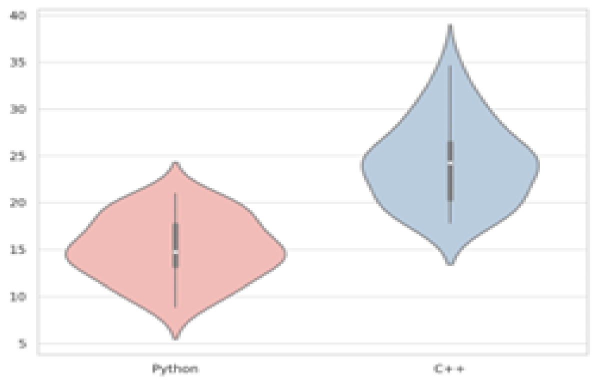


Figure 5 Violin plot shows Python students typically understand technical features in 13-17 hours, while C++ students take much time to understand

AI analysis accuracy: 92% of the respondents thought Python was better suited for AI analysis accuracy than C++. Among the most interesting results of student survey was the comparison of Python and C++ in terms of accuracy of AI analysis. One outstanding result was that a stunning 92 percent of respondents showed the opinion that Python is more applicable to processes that are concerned with AI especially when it comes to accuracy in implementation, model building, and library resources. Modern Python libraries like TensorFlow, PyTorch, Keras, and Scikit-learn can be listed only among the factors contributing to such a contrasting level of popularity. Such tools are used in academia as well as in industry, and allow students to learn to use real-world data and models without having to deal with low-level issues of memory allocation, pointer arithmetic, etc.

Learners also emphasized that the ease of pythons syntax and high level abstractions enabled them to have more attention on algorithmic logic and data processing than debugging technicalities. This was of particular importance when it came to activities focused on neural networks, processing of data, and construction of AI pipelines. C++ on the other hand is known to be of high performance and usually deployed in production resource where AI systems need maximized speed; in terms of learning and experiment, C++ was considered cumbersome because of its verbose syntax and a relatively steep learning curve. Most of the advanced students would prefer C++ to program AI, mostly those who had already experienced performance engineering or contest programming.

The findings of the paper indicate that Python makes AI engineering accessible more than ever before but the principles also indicate more innovativeness and experimentation amongst undergraduate learners. This is important in fields such as Artificial Intelligence and Data Science, where quick iteration is a key concept, namely rapid prototyping. The high level of Python usability in the AI setting is another point in support of its adoption as the language of choice to be used during the introductory programming classes of an AI nature. Future directions of the research will examine how exposure to Python at an early age achieves its effect on the proficiency in AI related course work and project deliveries in the long term.

FUTURE WORK:

This research seeks to examine the learning effectiveness of C++ and Python for first-year undergraduate students. However, it is critical to understand that the study serves only as a baseline assessment snapshot of the students' programming skills. The teaching of programming encompasses much more than mere rote recitation of keywords and algorithms; mastery is achieved over time, revealing multiple layers. To arrive at definitive insights, this study must be conducted over a longer time span—tracking participants throughout their four-year degree program. Such an analysis would enable educators and researchers to determine the impact of early language exposure on students' programming skills and performance in advanced courses conducted later in their studies. In the context of professional practice, the ultimate measure of effectiveness in programming education is readiness of the learner for industry and job performance. This therefore justifies examining how graduates from Python- or C++-focused pathways perform in actual positions in software engineering, data science, embedded systems, and artificial intelligence. By integrating educational assessment with actual demand in the market for web development, machine learning, systems programming, and cloud computing, future studies may establish whether foundational skills provided by certain programming languages.

Curriculum integration

As students' progress beyond introductory courses, their engagement with core subjects like Data Structures, Analysis of Algorithms, Object-Oriented Programming, Databases, and Operating Systems becomes crucial. These courses often require a solid grasp of abstraction, modularity, performance optimization, and algorithmic thinking. A comparative study of how students from C++ and Python foundations perform in these core courses could reveal which language better scaffolds learning in theoretical and applied computing domains.

Addition in core computing courses

In addition, incorporating evaluations into project-based capstone courses and interdisciplinary applications (such as robotics, IoT, and NLP) would give a complete picture of how adaptable and effective programming is. Given the rising specialization within computing disciplines, such as BS in Computer Science, BS in Artificial Intelligence, and BS in Data Science, it is pertinent to conduct language-specific batch-wise analyses across these programs. Each degree program brings out slightly varied flavors of computational paradigm, with Python its ecosystem and libraries taken to advantage in each case, whereas, systems-oriented CS tracks may fare better with C++ when it comes to low-level control and efficiency. Future research could assess student progression and project output during the degree as well as internships or employment provided at a later date to reveal not only which language best helps learn immediate understanding but also which is best translated to independent competency and problem-solving skills in the field of computing in general. As students advance past intro courses it is essential that they become acquainted with more substantive coursework in fields such as Data Structures, Analysis of Algorithms, Object-Oriented Programming, Databases, and Operating Systems. Such courses can demand a firm understanding of abstraction, modularity, performance optimization and algorithmic thinking. A comparative analysis study of performance of C++ foundation v/s Python foundation on these core courses may tell us which language can best scaffold learning the theoretical/applied computing domain .

References

[1] M. Ateeq, H. Habib, A. Umer, and M. Ul Rehman, "C++ or Python? Which one to begin with: A learner's perspective," in *2014 International Conference on Learning and Teaching in Computing and Engineering (LaTiCE)*, 2014, doi: 10.1109/LaTiCE.2014.20.

[2] Python Software Foundation, "Comparing Python to Other Languages," *Python.org*, 2017. [Online]. Available: https://www.python.org/doc/essays/comparisons/

[3] J. Wainer and E. C. Xavier, "A controlled experiment on Python vs. C for an introductory programming course: Students' outcomes," *ACM Transactions on Computing Education*, vol. 18, no. 3, 2018, doi: 10.1145/3152894.

[4] C. Martínez, M. J. Gómez, and L. Benotti, "A comparison of preschool and elementary school children learning computer science concepts through a multilanguage robot programming platform," in *Proc. ITiCSE*, 2015, doi: 10.1145/2729094.2742599.

[5] R. Turner, M. Falcone, B. Sharif, and A. Lazar, "An eye-tracking study assessing the comprehension of C++ and Python source code," in *Proc. ACM Symposium on Eye Tracking Research & Applications*, 2014, doi: 10.1145/2578153.2578218.

[6] N. Alzahrani, F. Vahid, A. Edgcomb, K. Nguyen, and R. Lysecky, "Python versus C++: An analysis of student struggle on small coding exercises in introductory programming courses," in *Proc. 49th ACM Technical Symposium on Computer Science Education (SIGCSE)*, 2018, pp. 246–251, doi: 10.1145/3159450.3160586.

[7] R. J. Enbody, W. F. Punch, and M. McCullen, "Python CS1 as preparation for C++ CS2," *ACM SIGCSE Bulletin*, vol. 41, no. 1, pp. 116–120, 2009, doi: 10.1145/1539024.1508907.

[8] D. Ginat, "On novice loop boundaries and range conceptions," *Computer Science Education*, vol. 24, no. 3–4, pp. 232–255, 2014, doi: 10.1080/08993408.2014.961507.

[9] L. Porter, C. Bailey Lee, B. Simon, and D. Zingaro, "Peer instruction: Do students really learn from peer discussion in computing?" in *Proc. 7th International Computing Education Research Workshop (ICER)*, 2011, pp. 45–52, doi: 10.1145/2016911.2016923.

[10] A. Robins, J. Rountree, and N. Rountree, "Learning and teaching programming: A review and discussion," *Computer Science Education*, vol. 13, no. 2, pp. 137–172, 2003, doi: 10.1076/csed.13.2.137.14200.