Ekene Onyekachi

Organization of Programming

Xin Wang

Date Pending

Section 1: Introduction

The languages selected for this comparison are Python for its simplicity (especially for new/aspiring programmers), Java for its best used object orientation, and C for its low-level control.

Each language has a specific insight that sets them apart from one another with Python being good at prioritizing productivity, readability, and ease of use, Java offering portability, standard-based interoperability, enterprise robustness, and C delivering raw performance and efficiency on computationally intensive tasks.

For comparison, I will be using Performance vs. Productivity and Simplicity vs. Power for both the Java and Python programming languages as I am more familiar with these two than C. Comparing the two languages out of the three would be valuable as it can be used to teach new programmers how a certain language will operate.

Section 2: Language Profiles

Python, which was created in 1991 by Guido van Rossum, was designed to make programming more readable, simple and accessible than C or Java. Its main goal is to become a general-purpose scripting language that emphasizes ease of use while still being powerful enough for large-scale software development. Its core design principles are readability, explicitly, simplicity, “one obvious way”, and practicality.

Python comes with a dynamic, strong type interface, with optional type hints. It also includes automatic memory management with garbage collection (refence counting and cyclic garbage collector), is a multi-paradigm, which is imperative, object-oriented, and functional all at once, as the primary paradigm, and is an interpreted execution model, running on the CPython interpreter.

There are areas where Python can excel such as rapid development, readability and maintainability, cross-domain utility, and housing a huge ecosystem of libraries and frameworks which is best suited for web development, data science & AI/ML, scripting, automation, prototyping, and education due to its simplicity. Its easy to learn, readable and expressive syntax along with the vibrant community, vast library ecosystem, and cross-platform portability make it popular among developers.

While it has its strengths, Python has its fair share of weaknesses such as slower performance than C++ and Java due to interpretation, a Global Interpreter Lock (GIL) limiting multi-threaded performances in CPU-bound tasks and has higher memory usage than others which makes it not suitable for performance-critical applications and mobile development. It has also been criticized for being too slow for some high-performance needs as well.

Java, which was created in 1995 by James Gosling, was designed to allow programs to run on multiple platforms without modifications. Its main goal is to be more secure and simpler than C++ while maintaining object-oriented capabilities for large-scale systems. Its core design principles are readability, simplicity, platform independence, robustness, and object-oriented design.

Java comes with a static, strong but limited type interface. It also includes automatic memory management with Java Virtual Machine (JVM) garbage collection, has an object-oriented paradigm as the primary paradigm, and is also a complied execution model.

There are areas where Java can excel such as cross-platform development, large-scale enterprise applications, a strong ecosystem of libraries and frameworks, and reliable, robust and secure for production systems which are best suited for enterprise web apps and backend systems, Android mobile apps, and high-reliability sever-side applications. Its platform independence, strong typing, which can reduce runtime errors, along with a rich ecosystem, strong tooling, and mature community makes it just as attractive to developers as Python.

However, like Python, Java also has its fair share of weaknesses such as its verbose syntax and slow startup time because of JVM which makes it not suitable for quick prototyping and memory or CPU-critical tasks. It has also been criticized for its verbosity, making code longer and sometimes harder to read and deployment complexity increasing due to JVM dependency.

C, which was created in 1972 by Dennis Ritchie, thus making it the oldest among the 3 chosen languages, is seen as the foundation of many modern languages. It was designed to write system software efficient and portable across hardware platforms providing low-level memory access while being higher level than assembly. Its core design principles are efficiency, portability, and simplicity.

C comes with a static typing, strong but low-level with no type interface. It also includes a manual memory management system, is a procedural/imperative paradigm as the primary paradigm and is a complied execution model directly to machine code.

Thera are areas where C can excel such as high-performance applications and low-level system programming which is best suited for operating systems and kernels, embedded systems and microcontrollers and high-performance computing tasks. Its absolute control among memory and hardware, extremely efficient execution, and minimal runtime make it attractive among developers.

Like the previous two languages, C has its fair share of weaknesses such as having no built-in garbage collection, manual memory management being prone to error, and lack of modern features makes it not suitable for rapid development and developer productivity. It has also been criticized for its steep learning curve, memory bugs and low-level syntax.

Section 3: Comparative Analysis

Performance vs. Productivity

The comparison between performance and productivity is critical because software projects must often strike a balance between how fast the program runs and how quickly developers can build and maintain it. In domains such as financial systems, high-performance backends, or real-time applications, runtime efficiency is essential. On the other hand, in research, data science, or startups, the ability to iterate quickly and produce working software matters more than raw execution speed.

Java approaches this balance by emphasizing runtime performance through its compiled JVM bytecode, Just-In-Time (JIT) optimization, and modern garbage collectors such as G1 and ZGC. These features allow Java applications to run with predictable latency and high throughput, making them ideal for enterprise-scale systems. Additionally, the introduction of virtual threads in Java 21 enables massive concurrency without complicating the programming model. Python, in contrast, focuses on developer productivity. Its interpreted and dynamically typed nature, along with a concise and readable syntax, allows developers to write and test code rapidly. While execution is slower compared to Java, Python offsets this limitation by relying on optimized native libraries such as NumPy or by using alternative runtimes like PyPy to improve speed.

Evidence supports these differences. Benchmarks such as the TechEmpower Round 23 (2025) consistently place Java frameworks among the leaders in web throughput, while Python frameworks typically perform lower. Java’s JIT compiler and low-pause garbage collectors further reinforce its reputation for runtime efficiency. On the other hand, Python demonstrates its strength in developer productivity. Studies of GitHub repositories and industry surveys, including those by Stack Overflow and JetBrains, show Python’s popularity in areas like machine learning and rapid prototyping, where developer time is more valuable than runtime performance.

The trade-off is straightforward: Java delivers better runtime performance at the cost of verbosity and longer development cycles, while Python enables rapid prototyping and experimentation at the cost of execution speed. For performance-critical systems, Java is the natural choice, but for tasks where iteration speed and flexibility dominate, Python excels.

Safety vs. Control

The balance between simplicity and power is equally important because programming languages must cater both to beginners seeking easy adoption and to advanced teams building highly complex systems. Simplicity reduces the learning curve and accelerates development, whereas power provides the expressive tools and scalability needed for long-term, large-scale applications.

Python embodies simplicity. Its design philosophy, captured in the Zen of Python, emphasizes readability and having “one obvious way” to accomplish a task. Features such as optional typing, standardized through PEP 484, allow teams to gradually adopt type safety as projects grow. Its syntax is concise—printing “Hello World” takes one line compared to Java’s multiple lines of boilerplate—and its asynchronous programming model with asyncio offers simple yet effective tools for handling concurrent I/O. Java, in contrast, prioritizes power. It provides a strong static type system, robust concurrency libraries, and expressive features like the Streams API. Although traditionally verbose, Java has been evolving records reduce boilerplate in data-centric classes, while virtual threads in modern releases make concurrency scalable without complex frameworks.

Examples highlight this contrast clearly. Python’s ease of use has made it dominant in education, scripting, and data science, while Java’s concurrency libraries and virtual threads equip it for building high-performance backend services. Python’s simplicity, however, comes at the cost of limited parallel CPU performance because of the Global Interpreter Lock (GIL), which prevents true multi-threaded execution of Python bytecode. Developers must work around this with multiprocessing, C extensions, or asynchronous I/O. Java’s power, while offering scalability and control, requires developers to manage additional complexity, such as understanding garbage collector tuning, type constraints, and concurrency primitives.

The trade-off lies in choosing between Python’s simplicity, which accelerates learning and development but limits raw parallelism, and Java’s power, which provides enterprise-grade tools at the expense of greater complexity. For small teams, rapid prototyping, or research, Python’s simplicity is a strong advantage. For large, mission-critical systems where robustness and scalability are paramount, Java’s power makes it the stronger candidate.

Section 4: Practical Decision-Making

I would use the Python coding language to make a simple dice rolling game. I have selected Python for its simplicity and my prior knowledge of the language. The factors that contribute to my decision are its simple power and slow productivity. The trade-off I will be accepting is its quick iteration, learning, and research.

References

* <https://medium.com/%40techathoncert/python-is-a-high-level-interpreted-programming-language-known-for-its-simplicity-readability-e06b903b1caf>
* <https://docs.python.org/3.11//faq/general.html?utm_source=chatgpt.com>
* <https://dl.acm.org/doi/fullHtml/10.1145/2716560>
* <https://medium.com/%40cedric.nanni/enterprise-application-development-java-standards-are-still-relevant-631802584252>
* <https://arxiv.org/abs/1409.0252>
* [https://peps.python.org/pep-0020/](https://peps.python.org/pep-0020/?utm_source=chatgpt.com)
* [https://docs.python.org/3/faq/general.html](https://docs.python.org/3/faq/general.html?utm_source=chatgpt.com)
* <https://pythoninstitute.org/python-history>
* [https://docs.python.org/3/c-api/memory.html](https://docs.python.org/3/c-api/memory.html?utm_source=chatgpt.com)
* [https://docs.python.org/3/library/gc.html](https://docs.python.org/3/library/gc.html?utm_source=chatgpt.com)
* [https://peps.python.org/pep-0703/](https://peps.python.org/pep-0703/?utm_source=chatgpt.com)
* [https://www.pypy.org/](https://www.pypy.org/?utm_source=chatgpt.com)
* [https://numpy.org/](https://numpy.org/?utm_source=chatgpt.com)
* [https://www.djangoproject.com/](https://www.djangoproject.com/?utm_source=chatgpt.com)
* [https://www.tensorflow.org/](https://www.tensorflow.org/?utm_source=chatgpt.com)
* [https://docs.oracle.com/javase/specs/](https://docs.oracle.com/javase/specs/?utm_source=chatgpt.com)
* [https://www.oracle.com/java/technologies/history-jsp.html](https://www.oracle.com/java/technologies/history-jsp.html?utm_source=chatgpt.com)
* <https://openjdk.org/groups/hotspot/docs/>
* <https://openjdk.org/jeps/310>
* <https://openjdk.org/jeps/295>
* [https://spring.io/projects/spring-framework](https://spring.io/projects/spring-framework?utm_source=chatgpt.com)
* <https://developer.android.com/guide/platform>
* <https://en.cppreference.com/w/c/history>
* <https://www.iso.org/standard/74528.html>
* [https://en.cppreference.com/w/c/memory/malloc](https://en.cppreference.com/w/c/memory/malloc?utm_source=chatgpt.com)
* [https://man7.org/linux/man-pages/man3/malloc.3.html](https://man7.org/linux/man-pages/man3/malloc.3.html?utm_source=chatgpt.com)
* <https://en.cppreference.com/w/c/language>
* <https://docs.oracle.com/en/java/javase/17/gctuning/garbage-first-g1-garbage-collector1.html>
* <https://docs.oracle.com/en/java/javase/21/gctuning/z-garbage-collector.html?utm_source=chatgpt.com>
* <https://www.techempower.com/blog/2025/03/17/framework-benchmarks-round-23/?utm_source=chatgpt.com>
* <https://docs.python.org/3/library/asyncio.html?utm_source=chatgpt.com>
* <https://openjdk.org/jeps/444>