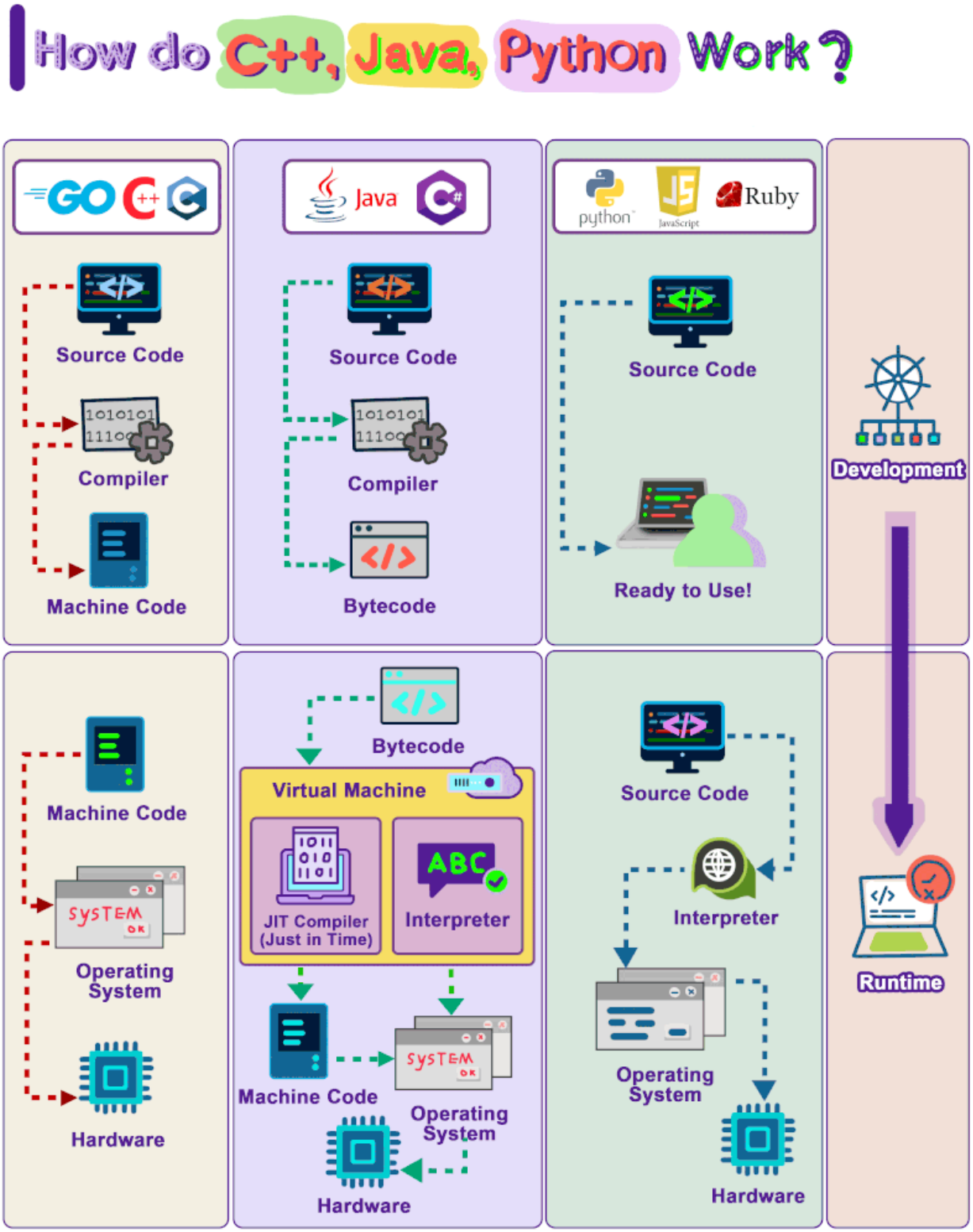
How C++, Java, Python works



Created by Hassan RJ

How C++, Java, Python works

1. C++ (Aur Saath mein C, Go waghera)

1. **Source Code (Likha hua code)**
   Aap sab se pehle C++ mein jo code likhte ho (jaise main.cpp file), woh aapka **source code** kehlata hai. Yeh human-readable hota hai, matlab log asaani se parh sakte hain.
2. **Compiler (Jaise g++ ya clang++)**
   Jab aap apna source code compile karte ho, ek **compiler** use hota hai. Compiler ka kaam yeh hai ke aapke high-level code ko **direct machine code** mein convert kare (ya kabhi pehle "object files" bana deta hai, phir link karke final machine code).
   - **High-level code**: Human-friendly code.
   - **Machine code**: 1s aur 0s wali language jo computer hardware samajhta hai.
3. **Machine Code**
   Jab compiler ka process khatam hota hai, to aapko ek **executable file** milti hai (Windows pe .exe, Linux/Mac pe bina extension ya .out). Yeh file **directly hardware** pe run ho sakti hai. Ismein koi bytecode layer nahin hoti, kyunke C++ native compilation karta hai.
4. **Run-time (Operating System + Hardware)**
   Jab aap yeh executable run karte ho, to aapka OS (Windows, Linux, Mac) usko hardware pe execute karwata hai.
   - **OS** instructions leta hai aur hardware ko kehta hai ke is program ke liye memory allocate karo, CPU cycles do, waghera.
   - **Hardware** (CPU, RAM) yeh sab instructions follow karta hai.

Ek Jumla Mein: C++ code pehle compile hota hai, phir direct machine code ban jata hai, phir aapka program hardware pe OS ke through run hota hai. Yeh compiled language hai, is liye bohat fast hoti hai. ⚡

---

2. Java

1. **Source Code (Likha hua code: .java files)**
   Aap Java mein jo code likhte ho, woh sab se pehle ek text-based source code hota hai (jaise HelloWorld.java).
2. **Compiler (Java Compiler: javac)**
   Jab aap javac HelloWorld.java chalate ho, yeh compiler aapke Java source code ko **Bytecode** mein convert karta hai.

- Yeh bytecode .class files ki shakal mein store hota hai.

3. **Bytecode**
   Java ka bytecode **platform-independent** hota hai. Matlab, yeh Windows, Linux, Mac, sab pe run ho sakta hai, bas aapko correct version ka Java Virtual Machine (JVM) chahiye.

4. **Java Virtual Machine (JVM)**
   Java code ko run karne ke liye aap java HelloWorld chalate ho, to JVM yeh bytecode load karta hai aur **Just-In-Time (JIT) compilation** waghera use karke aapke code ko actual machine instructions mein convert karta rehta hai run-time pe.
   - **Virtual Machine** ek beech ka layer hai jo bytecode ko samajh kar hardware instructions chalati hai.
   - **JIT Compiler** performance ko improve karta hai by converting frequently used bytecode parts into native machine code.

5. **Run-time (OS + Hardware)**
   JVM khud OS ke upar run ho rahi hoti hai, aur OS hardware resources manage karta hai.
   - Is tarah Java “Write Once, Run Anywhere” ka concept implement karti hai.

Ek Jumla Mein: Java mein code pehle Java compiler se **bytecode** banta hai, phir yeh bytecode **JVM** ke through run hota hai, jo usko run-time pe machine code mein translate karti rehti hai.
☕

---

3. Python

1. **Source Code (Likha hua code: .py files)**
   Aap Python mein jo code likhte ho, uski extension .py hoti hai (jaise app.py).

2. **Interpreter**
   Python mein ek compiler + interpreter mix tarah ka system hota hai, lekin user ke liye zyada tar yeh **interpreter-based** hi lagta hai. Jab aap python app.py chalate ho, Python internally code ko pehle **bytecode** mein convert kar sakti hai (.pyc files), phir us bytecode ko Python Virtual Machine (PVM) run karta hai.
   - Lekin yeh process zyada tar background mein hota hai. Aap directly .pyc file run nahin karte, balkay python interpreter run karta hai.

3. **Run-time**
   Python interpreter aapke OS ke upar run hota hai, aur OS hardware ko instructions deta

hai. Is tarah code run hota rehta hai line-by-line (ya chunk-by-chunk, depending on Python's internal mechanism).

4. **Dynamic & Interpreted**
    - Python **dynamically typed** hai, to aap variables ke types run-time pe assign kar sakte ho.
    - Yeh approach development ko asaan banati hai, lekin performance C++ jitni fast nahin hoti, kyunke har cheez run-time pe check ho rahi hoti hai.

Ek Jumla Mein: Python code interpreter chalata hai, jo pehle code ko bytecode mein convert karke, phir line-by-line instructions ko execute karta hai. Aap directly .py file run karte ho, aur interpreter OS ke zariye hardware instructions chalata hai. 🐍

---

4. Comparative Summary

Language	Compilation/Interpretation	Runtime	Performance	Portability
C++	Direct Machine Code (Compiled)	OS → Hardware	Very Fast (Native)	Must Recompile for each OS
Java	Bytecode + JVM (Hybrid)	Bytecode → JVM → OS → Hardware	Fast (JIT Optimization)	"Write Once, Run Anywhere"
Python	Interpreter (Bytecode + PVM)	Interpreter → OS → Hardware	Slower (Dynamic)	Portable if Python installed

- **C++**: Sab se tez, kyunke direct machine code hai, lekin har platform ke liye alag compile karna padta hai.
- **Java**: Bytecode banati hai, phir JVM run karwata hai, is liye bohat portable hai, performance decent hoti hai (JIT compiler ki wajah se).
- **Python**: Interpreter-based, likhna bohat asaan hai, bohat flexible hai, lekin performance compiled languages se kam ho sakti hai. Phir bhi, AI, data science, web dev mein bohat popular hai.

---

5. Development vs. Runtime

1. **Development Phase**

- Aap code likhte ho (source code).
- Java/C++ mein compile karte ho, Python mein directly run kar sakte ho.

2. **Runtime Phase**
    - C++: Executable file direct hardware pe chal rahi hoti hai.
    - Java: Bytecode JVM pe chal rahi hoti hai.
    - Python: Interpreter real-time code ko read karke execute kar raha hota hai.

---

6. Key Takeaways

1. **Compilation vs. Interpretation**
    - C++ jaisi compiled languages pehle se machine code bana deti hain.
    - Python jaisi interpreted languages run-time pe code parse karte hain.
    - Java ne beech ka rasta liya hai (bytecode + JVM).
2. **Performance vs. Ease**
    - Generally, compiled languages (C/C++) zyada fast.
    - Interpreted languages (Python) zyada developer-friendly, lekin performance mein thori compromise.
    - Java beech mein hai, performance theek hai aur portability achhi hai.
3. **Portability**
    - C++ code ko har OS ke liye separately compile karna parta hai.
    - Java code ko JVM sab OS pe run kar sakti hai, bas correct version of JVM chahiye.
    - Python code ko Python interpreter sab jagah run kar sakta hai, jab tak environment set ho.

---

7. Conclusion

Yeh infographic basically yeh samjha rahi hai ke **C++**, **Java**, aur **Python** apne code ko kis tarah se machine tak pohanchate hain. C++ **purely compiled** approach use karta hai, Java **bytecode +**

JVM approach, aur Python **interpretation** approach (with an internal compile-to-bytecode step, lekin mostly hidden from the user).

In Short:

- **C++:** Speed ke liye best, magar har OS ke liye compile karna parta hai.
- **Java:** Ek dafa code likho, har jaga run karo (JVM ki waja se).
- **Python:** Asaan, fast development, bohat libraries, lekin performance compiled languages jaisi nahin (phir bhi bohat use hoti hai, especially AI, data science, scripting waghera mein).

I hope yeh detailed Roman Urdu explanation aapki madad karegi! Agar aur sawal hain, to zaroor poochiye. Happy learning! 🚀😊