Programming Languages: Levels

1. **What is a low-level language? Can you name one/some?**
    - A low-level language is a programming language that is closer to machine code and provides minimal abstraction from a computer's architecture. Examples include Assembly Language and Machine Code.
2. **What features do low-level languages have?**
    - Low-level languages provide direct hardware manipulation, require detailed memory management, and use minimal abstraction. They often result in highly efficient and optimised code but are more complex to write and maintain.
3. **What kind of feature does a higher-level language have?**
    - Higher-level languages provide abstraction from the hardware, making programming easier and more readable. They typically include automatic memory management, built-in functions, error handling, and a more human-readable syntax. Examples include Python, Java, and C#.
4. **What programming languages can we think of, and can we classify them into 'high-level' and 'low-level'?**
    - **High-Level:** Python, Java, C#, JavaScript
    - **Low-Level:** Assembly, Machine Code
    - **Intermediate (Closer to Low-Level):** C, C++

Programming Languages: Levels & Safety

5. **Are higher-level programming languages safer? What do we mean by safer?**
    - Higher-level programming languages can be considered safer because they offer built-in memory management, automatic error handling, and security features that help prevent vulnerabilities such as buffer overflows, null pointer dereferences, and memory leaks.
    - Safety in this context refers to reducing the risk of bugs, security flaws, and unintended behaviour in a program.

High-Level, Low-Level, or Both?

6. **Can a programming language be only high-level or low-level, and not both?**
    - Not necessarily. Some languages, like C, can be considered both high-level and low-level depending on the context. While C is higher-level than Assembly, it is lower-level compared to Python or Java. Some languages, like Rust, aim to provide low-level control while incorporating higher-level safety features.

Programming Paradigms

7. **Can we program in any paradigm in any language?**
    - While many languages support multiple paradigms, not all languages are flexible enough to accommodate every paradigm. For example, C is primarily procedural and does not have built-in support for object-oriented programming (OO), whereas Python and Java support procedural, object-oriented, and some functional programming features.
    - Object-oriented programming in C is possible using structs and function pointers, but it lacks formal support like classes and inheritance.

Functional Programming

8. **What does functional programming offer that conventional structured imperative programming does not?**
    - Functional programming provides immutability, first-class functions, and the avoidance of side effects, leading to more predictable and testable code. It enables easier debugging and parallel execution due to the absence of shared mutable state.
9. **Can we program in this paradigm in any language?**
    - Many modern languages incorporate functional programming features. C++ (especially C++20) has added support for functional-style programming. Python, JavaScript, and Rust also allow functional programming techniques, even though they are not purely functional languages.

System-Level Programming

10. **What are the problems with C / C++ that make it ‘buggy’?**

- Unset or dangling pointers
- Null pointer dereferences
- Out-of-bounds errors
- Integer overflow errors
- Undefined behaviour that is not caught at compile time
- Lack of built-in memory safety features
- Manual memory management leading to leaks and vulnerabilities

---

Combination of Languages

11. **How about using more than one language for a project – for safety?**

- This is a common practice in software development. A combination of high-level and low-level languages can provide safety and efficiency:
    - **High-level languages (e.g., Python, Java):** Easier development, better memory management, and increased safety.
    - **Low-level languages (e.g., C, Rust):** Better performance, hardware-level access, and fine-grained control.
- Many systems use a combination, such as writing performance-critical components in Rust or C while using Python or Java for higher-level business logic.