Rust - Practice Sheet

10 conceptual Q&A Programming Languages

Q1. What are the primary design goals of Rust?

A: Rust was designed with specific priorities around performance, safety, or developer productivity depending on its domain. These goals shape its syntax, type system, and standard library choices.

Q2. How does Rust handle type checking: static or dynamic?

A: Rust uses its type system to catch errors at compile time (static) or at runtime (dynamic). This affects refactoring safety, tooling support, and the kinds of bugs that reach production.

Q3. How does Rust manage memory?

A: Rust uses one of three approaches: manual allocation/deallocation, a garbage collector that periodically reclaims unreachable objects, or automatic reference counting. Each has different latency and throughput tradeoffs.

Q4. What is Rust's concurrency model?

A: Rust supports concurrency through threads, coroutines, async/await, or an actor model. The choice determines how I/O-bound and CPU-bound workloads are scaled and how shared state is safely accessed.

Q5. How are packages and dependencies managed in Rust?

A: Rust uses a package manager and a manifest file to declare dependencies and version constraints. A lock file pins exact versions to ensure reproducible builds across environments.

Q6. How does Rust implement object-oriented or functional programming?

A: Rust supports classes and inheritance for OOP, or first-class functions and immutability for FP, or both. Many modern Rust programs blend both paradigms depending on the problem.

Q7. How does Rust handle errors?

A: Rust surfaces errors via exceptions, Result/Either types, or error return values. The choice impacts whether errors are checked at compile time and how calling code is structured.

Q8. What testing tools are commonly used in Rust?

A: Rust has a standard or widely adopted test runner and assertion library. Tests are typically organized into unit, integration, and end-to-end layers with coverage reporting built in or via plugins.

Q9. What makes Rust well-suited for its target domain?

A: Rust excels in its domain due to a combination of performance characteristics, ecosystem maturity, language ergonomics, and community support that makes common tasks simple.

Q10. What are common performance pitfalls in Rust?

A: Typical performance issues in Rust include excessive allocations, blocking the main thread or event loop, O(n^2) data structures, and missing caching. Profiling and benchmarking tools are available to diagnose them.