ONE PAGE SUMMARY

IT & Software › Programming Foundations › Rust

Introduction to Rust

Unlock the power of safe and efficient programming with Rust.

DEFINITION

> Rust is a systems programming language focused on safety and performance. It enables developers to write fast and reliable software without common bugs.

KEY CONCEPTS

Ownership

Rust uses an ownership model to manage memory safely without a garbage collector.

Borrowing

Borrowing allows functions to access data without taking ownership, ensuring safe data access.

Concurrency

Rust's design makes it easier to write concurrent programs that are free from data races.

Pattern Matching

Pattern matching in Rust allows for powerful control flow and data handling with enums and match statements.

Cargo

Cargo is Rust's package manager and build system, simplifying project management and dependencies.

EXAMPLES

→ let x = 5; // Immutable variable

→ let mut y = 10; // Mutable variable

→ fn add(a: i32, b: i32) -> i32 { a + b }

MEMORY TIPS

★ Remember 'ownership' as 'who owns the data?' to grasp memory management.

★ Think of 'borrowing' like lending a book; you can read it but not keep it.

★ Use 'Cargo' as a reminder of 'cargo ship' to manage your project's dependencies.

COMMON MISTAKES

✗ Forgetting to specify mutability when needing to change a variable.

✗ Confusing ownership transfer with borrowing, leading to compile errors.

✗ Neglecting to handle errors properly, which can cause panics.

QUICK RECAP

Rust emphasizes safety and performance through its unique ownership model. Understanding key concepts like borrowing and concurrency is crucial for effective programming in Rust.