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, speed, and concurrency. It enables developers to write reliable and efficient code without sacrificing performance.

KEY CONCEPTS

Ownership

Rust uses ownership to manage memory, ensuring that each value has a single owner.

Borrowing

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

Lifetimes

Lifetimes track how long references are valid, preventing dangling references and ensuring memory safety.

Pattern Matching

Pattern matching provides a powerful way to handle different data types and structures in Rust.

Concurrency

Rust's concurrency model allows safe parallel execution of code, minimizing data races.

EXAMPLES

→ let x = 5; let y = &x;

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

→ let result = add(2, 3);

MEMORY TIPS

★ Remember 'ownership' as 'one owner, one value' for clarity.

★ Use the phrase 'borrow but don't take' to recall borrowing rules.

★ Think of lifetimes as 'time stamps' for references to avoid confusion.

COMMON MISTAKES

✗ Forgetting to specify lifetimes in function signatures.

✗ Attempting to use a value after it has been moved.

✗ Confusing mutable and immutable references.

QUICK RECAP

Rust emphasizes safety and performance through ownership and borrowing. Understanding key concepts like lifetimes and pattern matching is crucial for effective programming in Rust.