RUST

FAST, SAFE, AND CONCURRENT

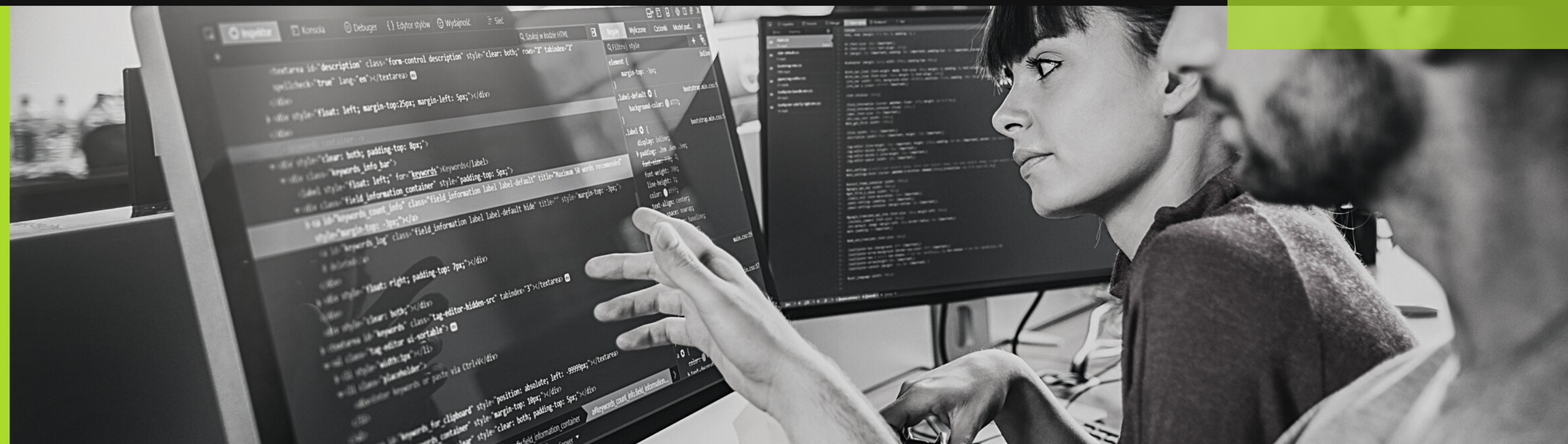
START SLIDE

WHY RUST

- Memory Safety: Guarantees safety without a garbage collector, eliminating common bugs found in other languages.
- High Performance: Speed comparable to C and C++, ideal for system-level tasks.
- Modern Tooling: Cargo for package management, a rich standard library, and a syntax that embraces modern programming paradigms.
- Fearless Concurrency: Write concurrent code with confidence, thanks to Rust's ownership and borrowing system.
- Growing Ecosystem: An expanding community, extensive libraries, and widespread industry adoption.

OVERVIEW

In the landscape of programming languages, few combine the best of both worlds: memory safety and high performance. Meet Rust, a modern systems programming language that does precisely that.



2015

Release Date

ADVANTAGES OF RUST OVER OTHER LANGUAGES

- Memory Safety: Achieves it without a garbage collector, offering predictable performance.
- Concurrency: Guarantees thread safety, eliminating common concurrency bugs.
- Avoids Null Errors: Rust’s type system minimizes null pointer issues.
- C/C++ Interoperability: Seamlessly integrates with existing C and C++ codebases.
- Modern Tooling: Offers fresh syntax and integrated tools like the Cargo package manager.
- Performance & Security: Matches C/C++ speed with fewer vulnerabilities.

In essence, Rust blends high performance, safety, and modern development practices, standing out in the landscape of programming languages.

APPLICATION

- Browser Components
- Cross-platform Development
- Systems Programming
- Databases
- Game Development

- Utilities and Tooling
- Networking
- Web Development
- Cryptography
- Simulation and Scientific Computing

The adoption of Rust is growing, and as the ecosystem matures, its applications are bound to expand even further. The main draw is the promise of developing highly performant software without sacrificing safety or increasing the potential for runtime errors.

6000+

Crates Available

ADOPTION

Recognition: Voted the "most loved language" on Stack Overflow multiple years in a row.

Big Tech:

- Mozilla: Incorporated Rust into Firefox.
- Microsoft: Eyeing Rust for safer system programming.
- Google: Funded Rust-based projects like Fuchsia.
- Facebook: Used Rust in projects like the Diem blockchain.

Open Source: Growing interest in integrating Rust into the Linux kernel and projects like Tor.

Web Development: Rust's WebAssembly compilation broadens its appeal for web apps.

Startups & Companies: Many adopt Rust for its safety and performance advantages.

Community Growth: A rising number of Rust libraries, conferences, and global events.

Safety-Critical Domains: Aerospace and automotive sectors are exploring Rust for its robustness.

Embedded Systems: Rust is challenging the C/C++ dominance in this space.

POTENTIAL DRAWBACK TO ADOPTION

Learning Curve: Rust's unique concepts like ownership and lifetimes can be challenging, especially for those new to the language.

Verbose Syntax: This can slow down development, particularly for those unfamiliar with Rust.

Ecosystem Size: Rust's library and framework ecosystem is not as mature as some older languages.

Compilation Time: The emphasis on safety and thorough checks can lead to longer compile times.

Corporate Adoption: Rust hasn't been as widely adopted in enterprise environments compared to languages like Java or C++.

Tooling: While growing, some aspects of Rust tooling, such as IDE support or certain profiling tools, have room for improvement.

RUST 101

To get started with Rust, you primarily need to install rustup, which includes both the Rust compiler (rustc) and its package manager (cargo). Next, you will install an editor of your choice

RUST: DATA TYPES, FUNCTIONS & CONTROL FLOW

01.

Data Types

- **Primitive:**
  - Integers: i32, u32, i64, u64, etc.
  - Floating-point: f32, f64.
  - Boolean: bool.
  - Char: char.
- **Compound**:
  - Tuples: (i32, f64, u8).
  - Arrays: [i32; 5].
  - Slice: &[i32].

02.

Function

- Declaration: fn
- fn_name(args) -> return_type {}.
- Parameters & Return Types: fn add(a: i32, b: i32) -> i32.
- Closures: |x| x + 1.

03.

Control Flow

- Conditionals:
  - if, else if, else.
  - Match: Rust's version of a switch-case.
- Loops:
  - loop, while, for.
- Pattern matching with match.

RUST OWNERSHIP & MEMORY MANAGEMENT

Understanding Safe & Efficient Code

Ownership in Rust:

- Memory safety without a garbage collector.
- Concurrency without data races.
- Clear, explicit code contracts.

Rules of Ownership:

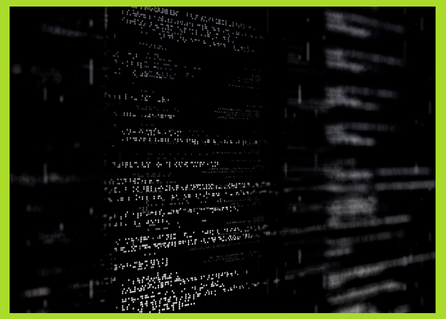
- Each value in Rust has a single owner.
- Value out of scope? Memory gets freed.
- Transferring ownership = "Moving" in Rust.

Memory in Computers

- Two main parts:
  - Stack: Fast, LIFO, Fixed-size.
  - Heap: Dynamic, Slower.

STACK

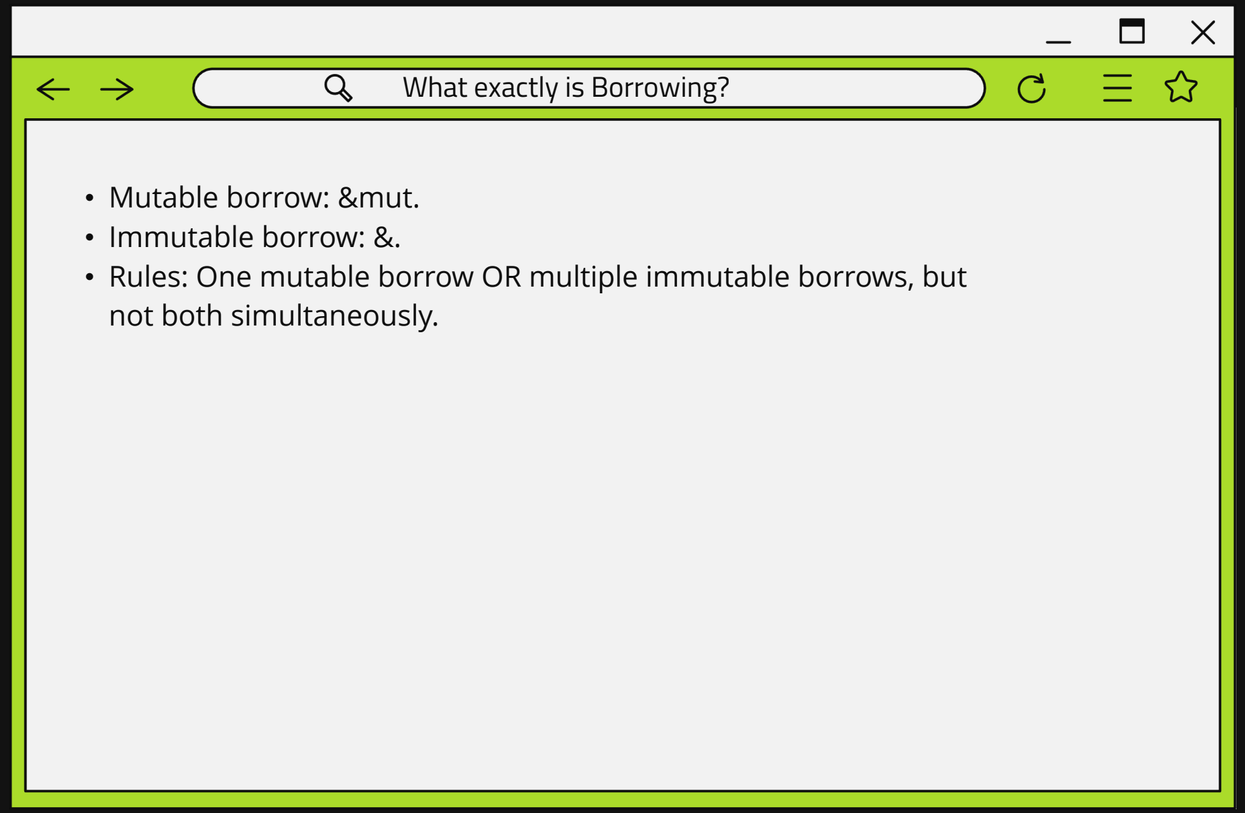
- Data known at compile time.
- Automatic cleanup.
- Examples: fixed-size arrays, integers.



HEAP

- Dynamically sized data.
- Requires manual management in many languages.
- Examples: Vectors, Strings.

BORROWING



Mutable vs. Immutable Borrow

Mutable Borrow:

- Allows data modification.
- Exclusive access.

Immutable Borrow:

- Read-only access.
- Can have multiple readers.

02. Lifetimes

- Ensures references are valid.
- Annotated with 'a, 'b, etc.
- Ensures no dangling references.

01. Pattern Matching

- Allows you to destructure complex types.
- Especially powerful with match.
- Can move ownership conditionally.

04. Generics & Traits

- Writing generic functions, structs, and methods.
- Defining shared behavior with Traits.

03. Closures & Iterators

- Basics of closures: |x| x + 1.
- Iterators: methods like map(), filter(), and collect().

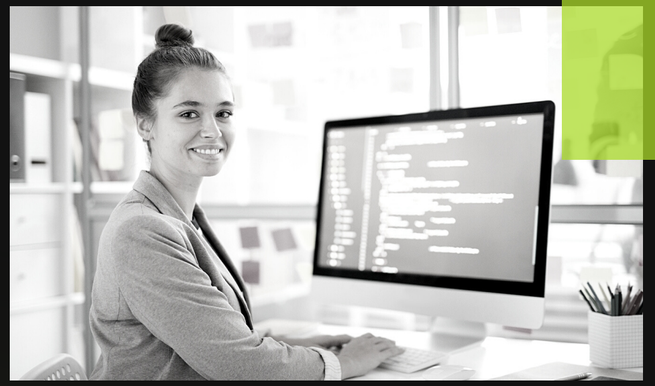
06. Data Structures: Enum

- Define a type by enumerating possible values.
- Usage: Ideal for values with several distinct possibilities.

05. Data Structures: Struct

- Custom data types combining multiple values.
- Types: Classic (named fields), Tuple (unnamed fields), and Unit-like (no fields).

ADVANCED DATA STRUCTURES

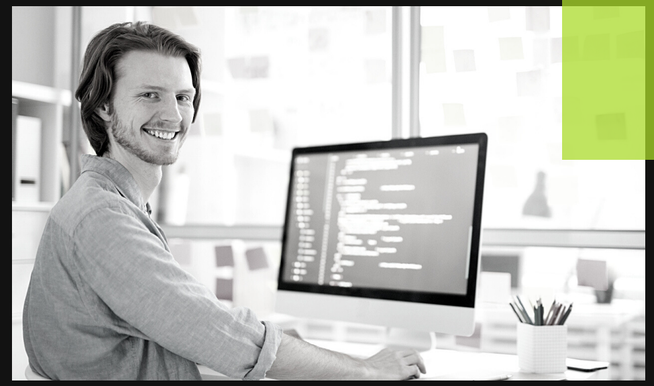


Vec<T>

- Stored on the heap.
- Grows and shrinks dynamically.
- Ownership challenges: resizing, dropping.

String

- UTF-8 encoded text.
- Can grow and mutate.
- Indexing is non-trivial due to multi-byte characters.



HashMap<K, V>

- Key-value storage.
- Hashing ensures O(1) complexity in optimal cases.
- Ownership considerations when inserting and accessing.

CARGO

Introduction to Cargo

- Rust's build tool and package manager.
- Basic commands: cargo build, cargo run.

Growing with Rust: Packages, Crates, Modules

- Structuring larger projects.

Managing Dependencies & Builds with Cargo

- Adding dependencies.
- Compiling projects and managing builds.

ANY QUESTIONS?


THANK YOU