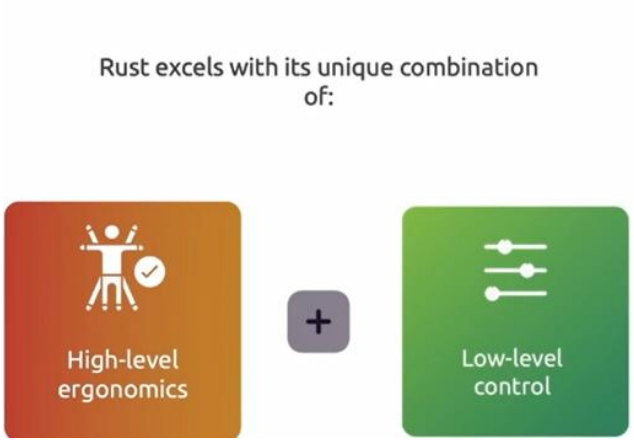

Systems programming language

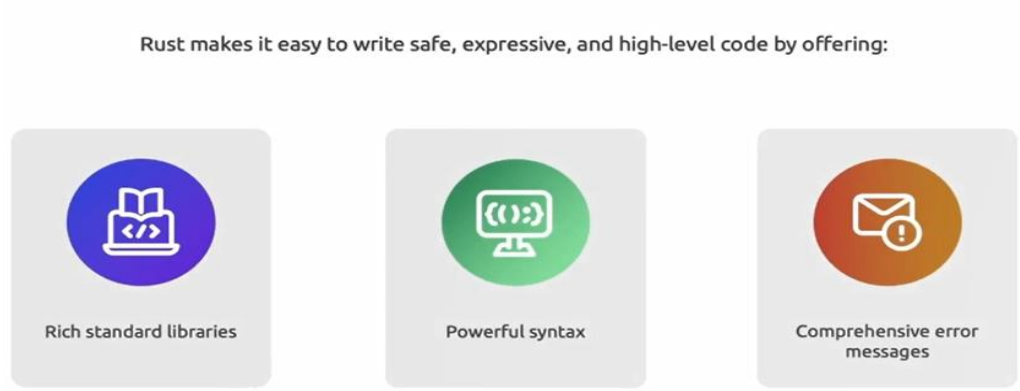
****


Helps in writing fast and reliable software

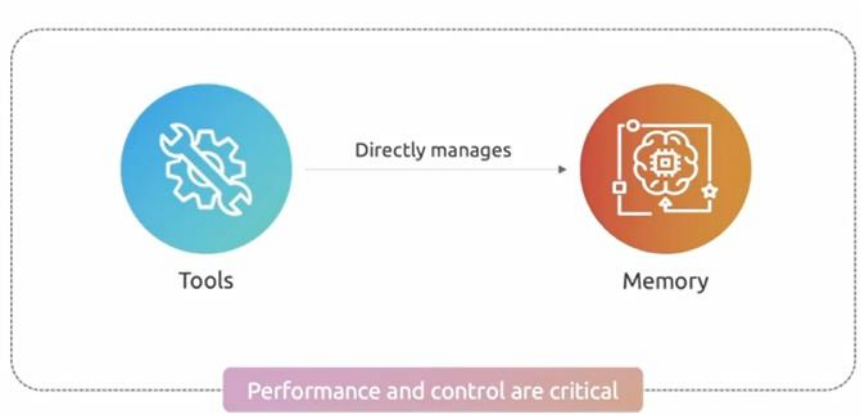
****


Rust excels with its unique combination of:
High-level ergonomics
+
Low-level control

****


Rust makes it easy to write safe, expressive, and high-level code by offering:
Rich standard libraries
Powerful syntax
Comprehensive error messages

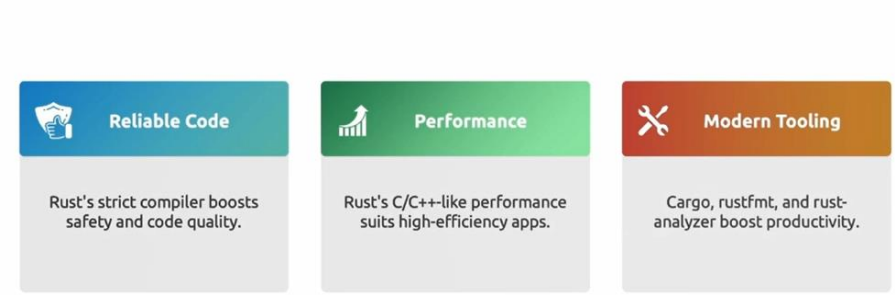
****



****

****

For Professionals



****

Rust – Trusted by Industry Leaders for Performance

Mozilla

Rust was built by Mozilla for Firefox's performance-critical parts.

Dropbox

It uses Rust in storage for performance and reliability.

Coursera

It uses Rust in data pipelines for efficiency and safety.

****

Rust – Trusted by Industry Leaders for Performance

Figma

It switched from TypeScript to Rust for better multiplayer performance.

Microsoft

It integrates Rust for safer, faster systems programming.

****

Rust – Key Features

Rust is designed with a focus on:

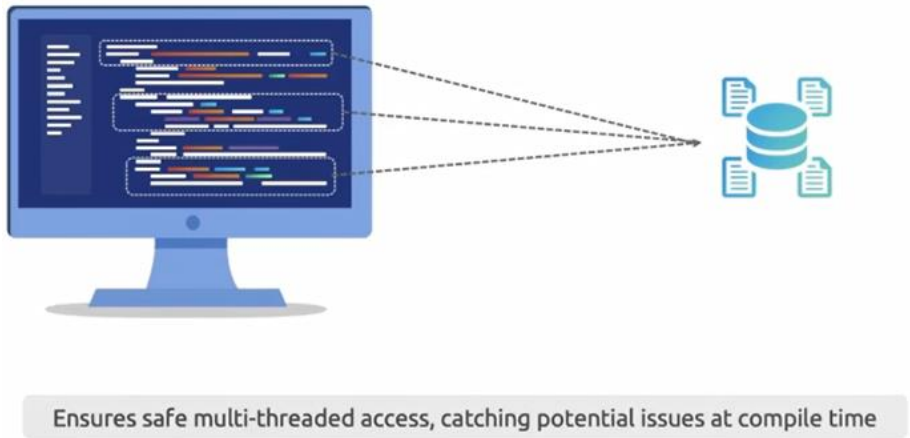
****


1
Memory Safety
Safely manages memory without a garbage collector

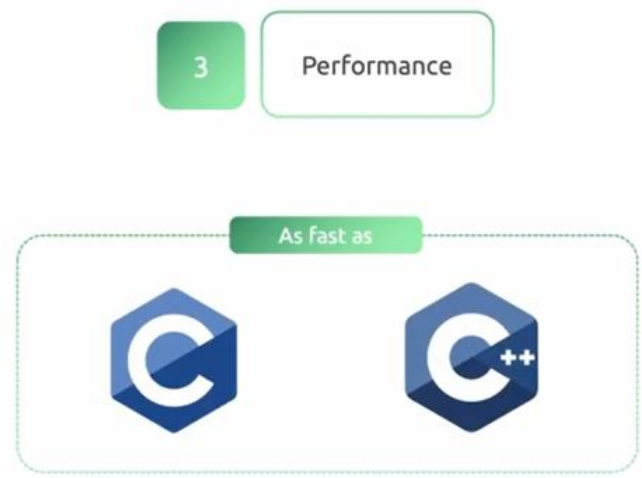
****


Rust enforces ownership rules at compile time.
01
Ownership
Each piece of data in Rust has a single owner.
02
Borrowing
References can be created without owning the data.
03
LIFETIME
Lifetimes
These ensure that references are always valid.

****

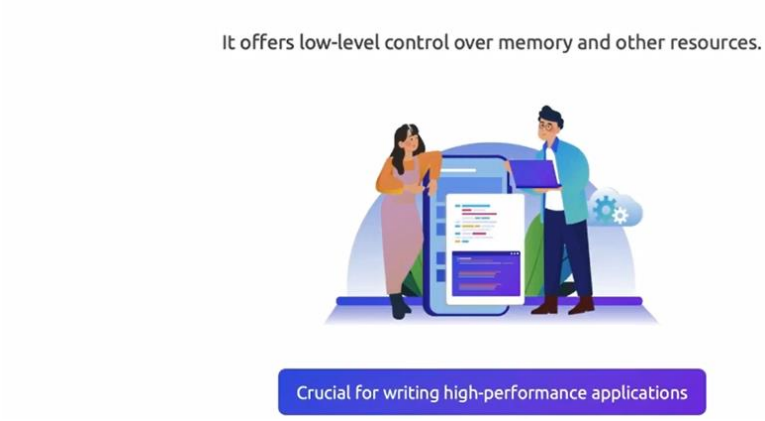

Ensures safe multi-threaded access, catching potential issues at compile time

****



****

Rust – Key Features



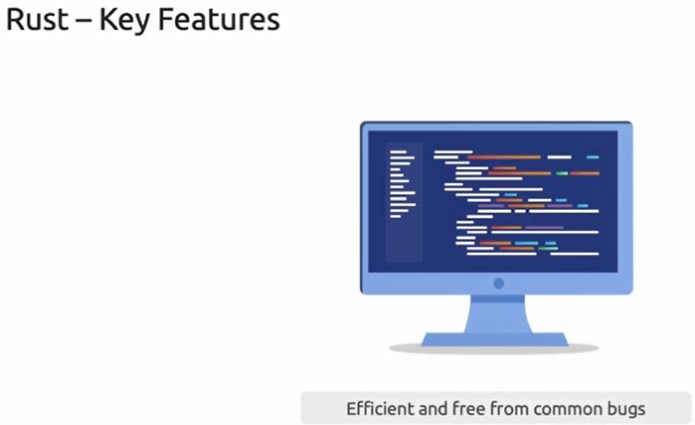
****

Rust – Key Features

****

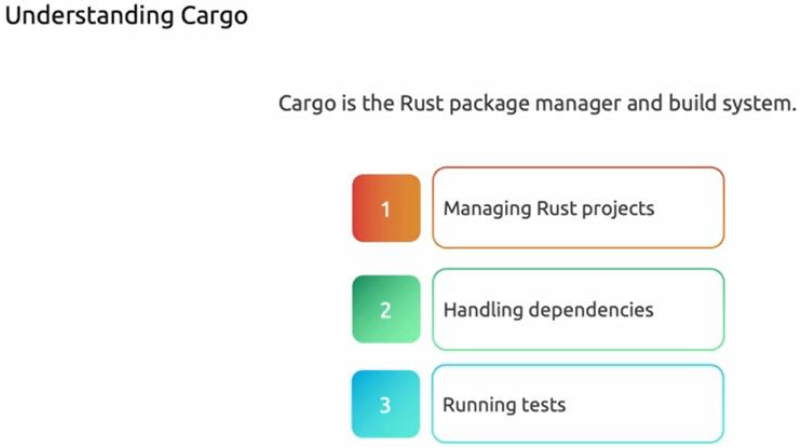
****

Rust – Key Features



****

Understanding Cargo



****

****

To Create Simple Rust Project Using Cargo:

- ***Run Command***: *cargo new project_name (ex: hello_rust)*

```
C:\Tests\Rust\From-Youtube-01>cargo new hello_rust
    Creating binary (application) `hello_rust` package
note: see more `Cargo.toml` keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html
```

****

****

Cargo – Summary

Cargo is a powerful tool that simplifies Rust development by:

****

Macros In Rust Are Powerful Way To Generate Codes In Compile Time, It Ends With: **!**

****

To build the project using cargo, we run command: *cargo build*

To run the project using cargo, we run command: *cargo run*

****

To keep your code consistent and well-formatted, you can use the rustfmt tool, which is included with the standard Rust distribution. Just run rustfmt on your Rust files to automatically format your code.

****

Single-Line Comments

Starts with two forward slashes **//** and is ignored by the compiler

main.rs

```
// This is a single-line comment
fn main() {
    // This line prints "Hello, world!"
    println!("Hello, world!");
}
```

****

Multi-Line Comments

Useful to write longer explanations or temporarily disable blocks of code. **Start** with **/*** and **end** with ***/**

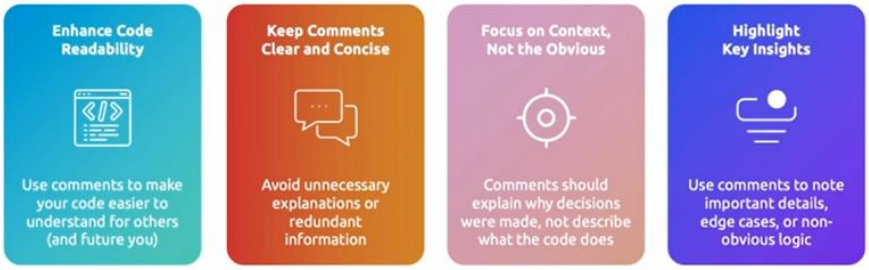
main.rs

```
/*
 * This is a multi-line comment.
 * It spans multiple lines and is useful for
   providing more detailed explanations.
 */

fn main() {
        println!("Hello, world!");
}
```

****

Comments – Best Practices



****

Immutability by Default

main.rs

```
fn main() {
    let y = 10;
    println!("The value of y is: {}", y);
    y = 20; // This line will cause a compilation error
    println!("The value of y is: {}", y);
```

****

Making Variables Mutable

main.rs

```rust
fn main() {
    let mut y = 10;
    println!("The value of y is: {}", y);
    y = 20;
    println!("The value of y is: {}", y);
}
```

****