

# Concurrent programming in Rust

A Journey into Fearless Concurrency

Juan Camilo Garcia Martinez  
Daniel Alejandro Melo Nuvan  
Elían Gonzalez Ordóñez  
Camilo Arturo Echeverry Ayala





# What's on this journey?

## 01 Rust: not just a Metal Oxide

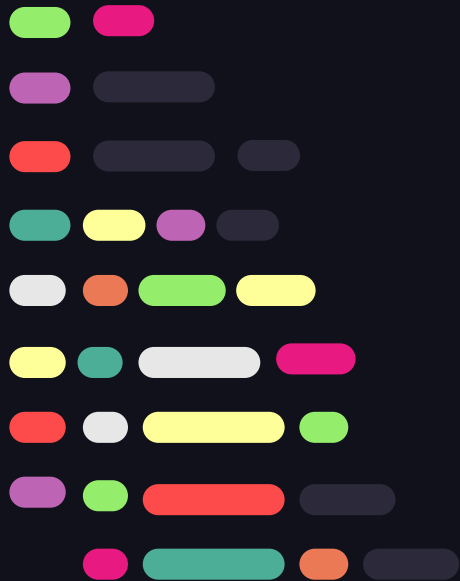
Overview of the Rust programming language!

## 02 Getting Rusty

How to setup your coding playground with Rust.

## 03 Memory Management in Rust

A different way of thinking about memory.





01 { ..

Rust: not just a  
metal oxide

Overview of the Rust programming language!





# Overview of rust



## Performant

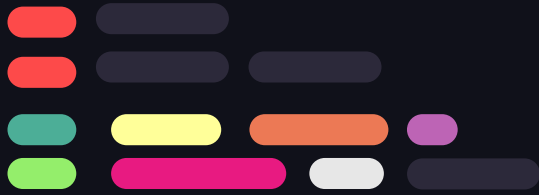
No extra operations as it's memory safe through ownership, borrowing and lifetimes,

## Type safe

Static and strong typing, designed to catch type errors on compile time.

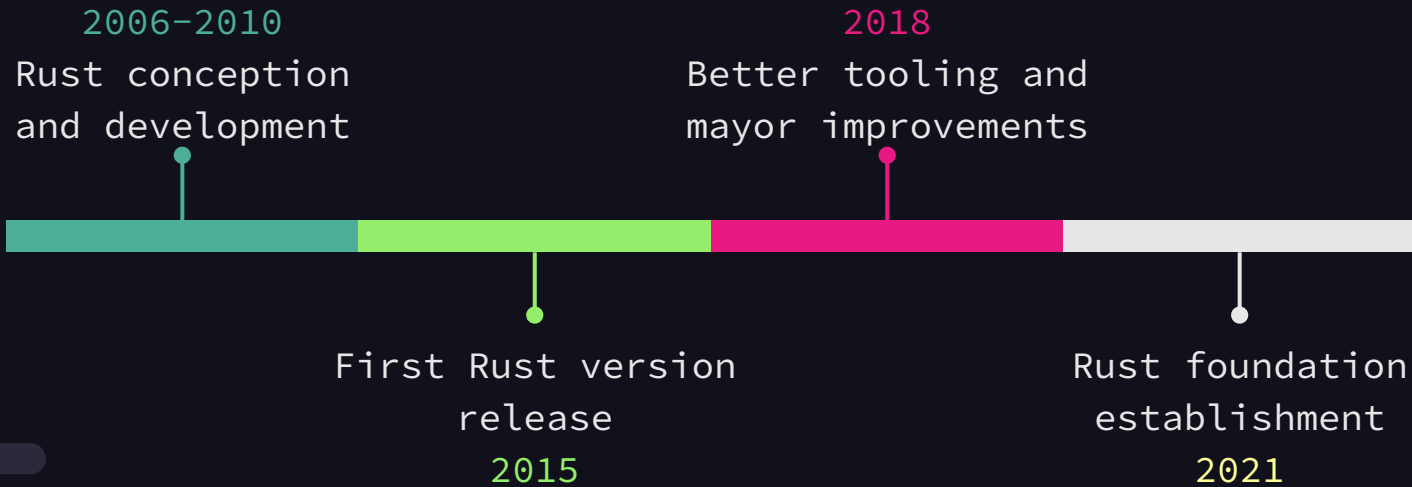
## Concurrent

Designed for a fluent experience in threads management and communication.



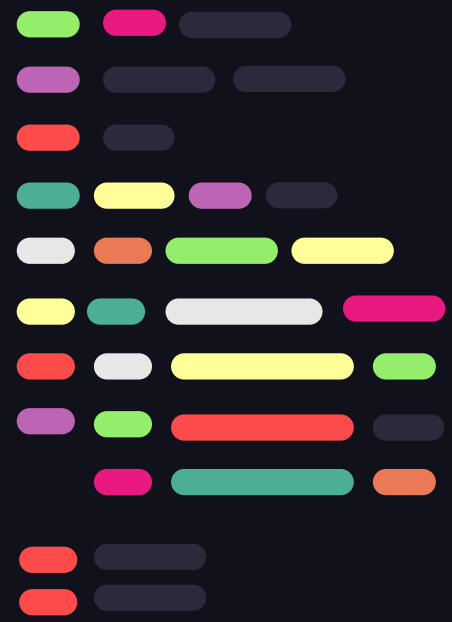


# On the time machine..





# What it's used for?



## CLI tools

Useful program with a CLI interface.

## Web development

Building and maintenance of websites.

## Systems

### programming

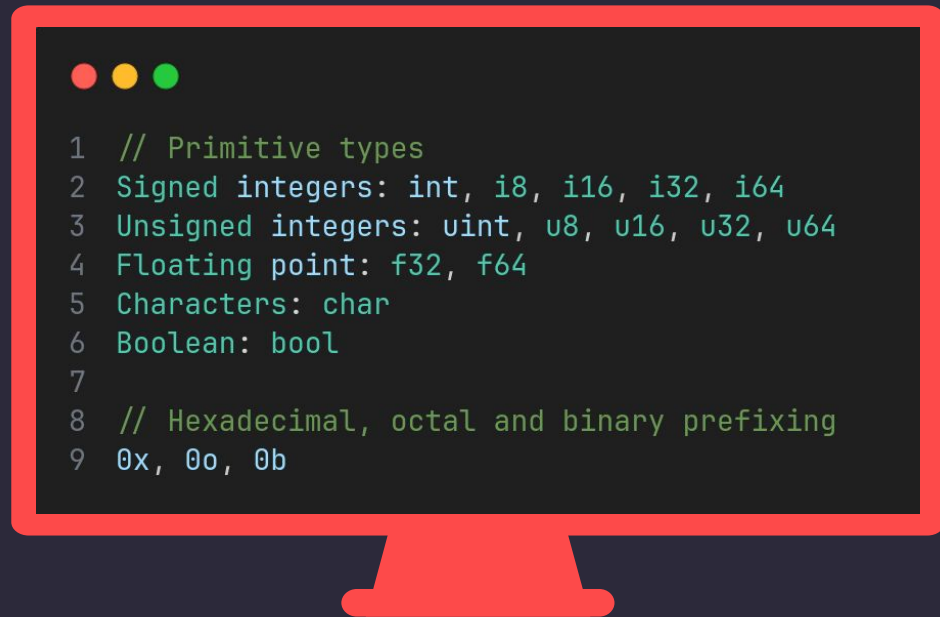
Aims to produce software for the hardware.

## Network

### programming

Programs that communicate across network.

# { Rust primitives





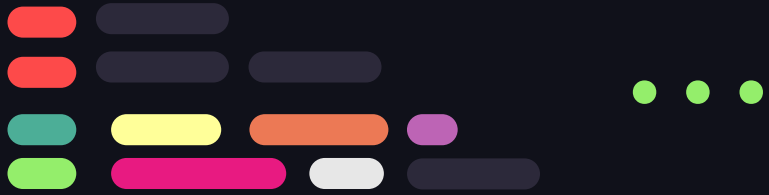
```
1 //functions
2 fn <name>(<args>) → <return type> {
3     <statements>
4 }
```

{ Rust  
functions

}



# { Rust conditionals

A stylized computer monitor with a red frame and a red base. The screen displays code. In the top left corner of the screen, there are three small colored dots (red, yellow, green).

```
1 // Rust conditional statements
2 if <condition> { <statements> }
3 else if <condition> { <statements> }
4 else { <statements> }
```





```
1 // Rust loops
2 loop {
3     // Loop until break
4     <statements>
5 }
6
7 // Rust for loop
8 for <var> in <iterable> {
9     <statements>
10 }
11
12 // Rust while loop
13 while <condition> {
14     // Loop until condition is false
15     <statements>
16 }
```



# Rust loops



# { Rust matches





```
1 // Rust structs
2 struct <name> {
3     <field1>: <type1>,
4     <field2>: <type2>,
5     ...
6 }
```

Rust  
{ structures

}



02 { ..

# Getting Rusty

How to setup your coding playground with Rust.





# But, how it's installed?



## Rustup

Rust installer and  
version management tool

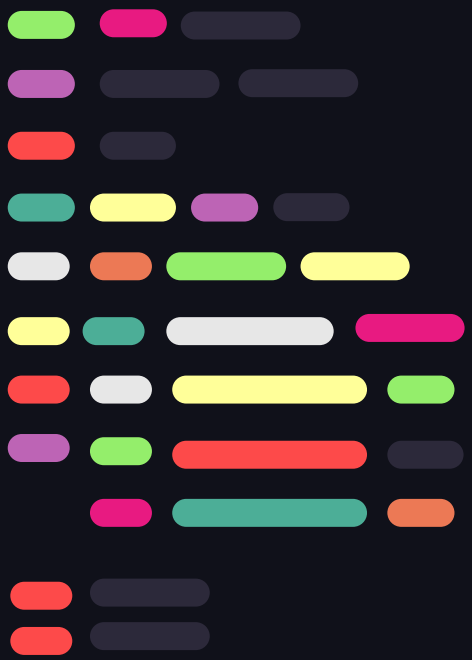
## Cargo

Rust build and package  
manager, automatically  
installed with **Rustup**.





# Start a new project



Start new project with `cargo new`  
`<project-name>`:

Generates:

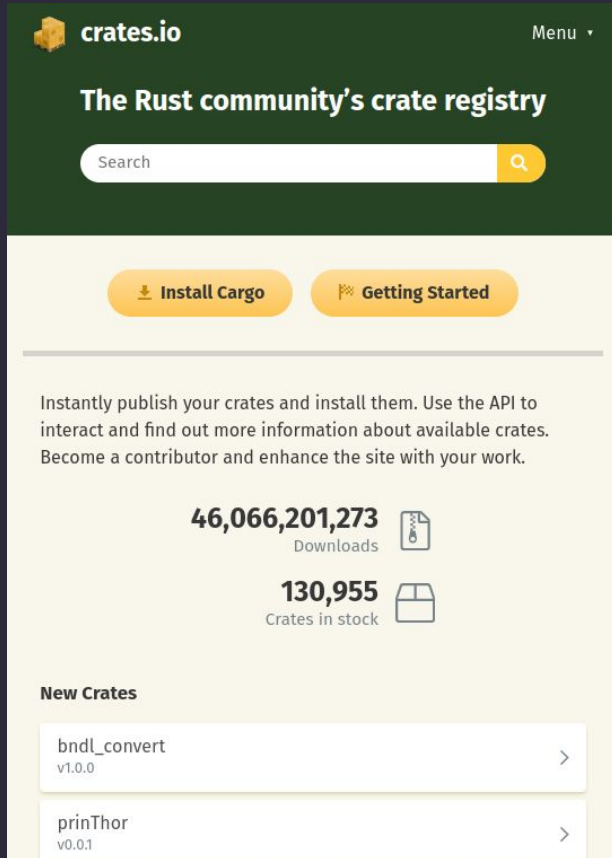
- Cargo.toml: metadata of the project
- src/main.rs: application code

Hello world project with `cargo new`



# And the libraries?, installing **crates**

Install with `cargo add <package-name>` from [creates.io](https://crates.io), the version is saved in `Cargo.toml` file



The screenshot shows the crates.io website interface. At the top, there is a dark green header with the crates.io logo and a "Menu" dropdown. Below the header is a white search bar with a magnifying glass icon. Two yellow buttons, "Install Cargo" and "Getting Started", are positioned below the search bar. The main content area has a light yellow background and contains the following text: "Instantly publish your crates and install them. Use the API to interact and find out more information about available crates. Become a contributor and enhance the site with your work." Below this text are two statistics: "46,066,201,273 Downloads" with a document icon, and "130,955 Crates in stock" with a crate icon. A section titled "New Crates" lists two items: "bndl\_convert v1.0.0" and "prinThor v0.0.1", each with a right-pointing arrow.

crates.io Menu ▾

The Rust community's crate registry

Search

Install Cargo Getting Started

Instantly publish your crates and install them. Use the API to interact and find out more information about available crates. Become a contributor and enhance the site with your work.

46,066,201,273 Downloads

130,955 Crates in stock

New Crates

- bndl\_convert v1.0.0
- prinThor v0.0.1







# Integrations





03 { ..

# Memory Management in Rust

A different way of thinking about memory.

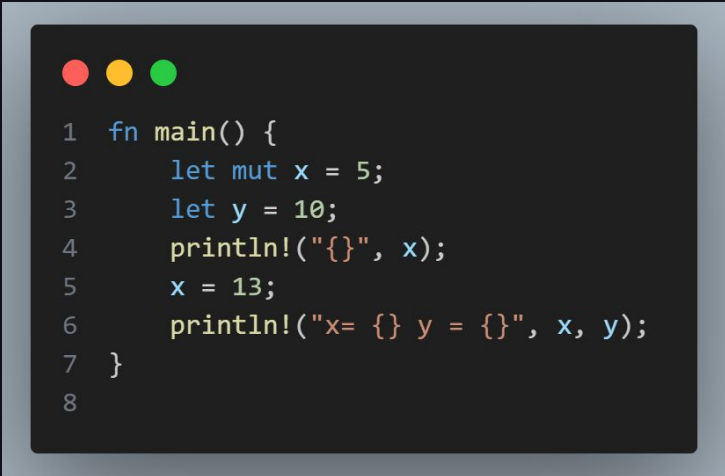


# Mutability



## Values fall into two types:

- **Mutable**: The compiler allows the variable to be written to and read from.
- **Immutable**: The compiler only allows the variable to be read from.



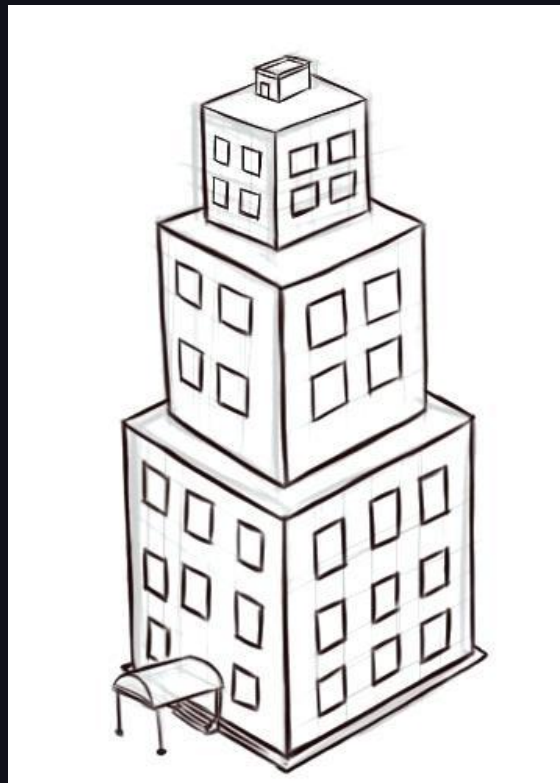
```
1 fn main() {  
2     let mut x = 5;  
3     let y = 10;  
4     println!("{}", x);  
5     x = 13;  
6     println!("x= {} y = {}", x, y);  
7 }  
8
```



# Memory Management: Ownership

## Three ways:

- Explicit allocation and releasing of memory.
- Garbage collection
- Rust uses a third approach: memory is managed through a system of ownership





# Rules of Ownership

1. A variable called its "owner".
2. Only one owner at a time.
3. When the owner goes out of scope, the value will be dropped.

```
1 fn main() {  
2     // s is not valid here  
3     let s = "hello"; //valid from here  
4     // do stuff with s  
5 } //this scope is now over, and s is no longer valid  
6  
7
```

When `s` comes into scope, it is valid. it remains valid until it goes out of scope.



# Scope-Based Resource Management

Rust uses the end of scope as the place to deconstruct and deallocate a resource.

This is called a `drop`.

```
1 struct Bar {
2     x: i32,
3 }
4
5 struct Foo {
6     bar: Bar
7 }
8
9 fn main(){
10     let foo = Foo { bar: Bar{ x: 42 } };
11     println!("{}", foo.bar.x);
12     // foo is dropped first
13     // Then foo.bar is dropped
14 }
```





# Move semantics and Ownership

When doing assignments or passing function arguments **by value (not reference)**, the ownership of the resources is transferred.

This is called a **move**.

```
1 fn foo(s: String) {
2     // s is moved into foo, and s is no longer valid
3     println!("{}", s);
4 }
5
6 fn main() {
7     let s = String::from("Hello, world!");
8     foo(s);
9     //println!("{}", s); will not compile
10
11 }
```



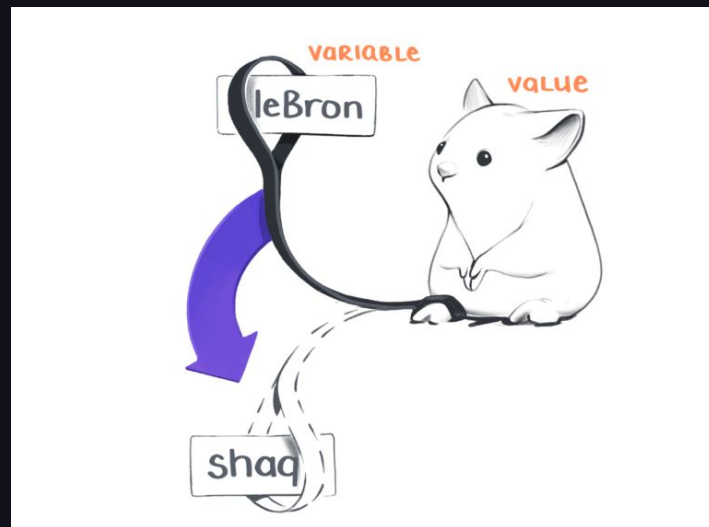
# Borrowing

You can **borrow ownership**.

Instead of passing objects by value,  
we can pass these objects by  
reference.

The **borrow checker** guarantees that  
references always point to valid  
objects.

All of this ensures that programs are  
**memory safe**.

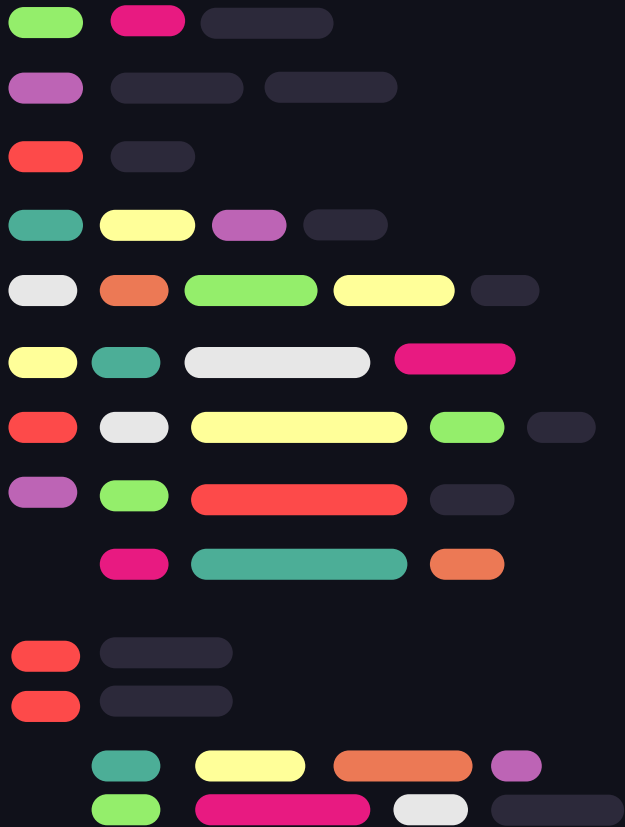






# Jupyter Notebook!

Access live Rust code with the shown  
concepts here : [Notebook](#)





# Resources

- Drawing Skill. Building Image. Retrieved from <https://www.drawingskill.com/art/39893>
- JetBrains. Rust in the Dev Ecosystem. Retrieved from <https://www.jetbrains.com/lp/devecosystem-2021/rust/>
- Brson. A Guide to Rust Syntax. Retrieved from <https://gist.github.com/brson/9dec4195a88066fa42e6>
- OpenAI. Chat GPT as a powered Search Engine. Retrieved from <https://chat.openai.com/>

