Introduction to Rust programming language

Rafał Draws

Foreword

Agenda

- A brief history of Rust
- Typesystem
- Memory Safety Core - Ownership, Borrowing, Lifetimes
- Heap and Stack
- Appliences
- How to start

Who am I

Rafał Draws, MSc

- Software Engineer at Nordea Bank
- Rust Poland founder
- Music nerd
- Automotive nerd

how can you tell that someone drives a Lexus?

they'll tell you

yes its '99, and has 300 thousand miles

What is Rust?

"Rust is a general-purpose programming language. It is noted for its emphasis on performance, type safety, concurrency, and memory safety.

Rust supports multiple programming paradigms. It was influenced by ideas from functional programming, including immutability, higher-order functions, algebraic data types, and pattern matching.

It also supports object-oriented programming via structs, enums, traits, and methods.

Rust is noted for enforcing memory safety (meaning that all references point to valid memory) without a conventional garbage collector;

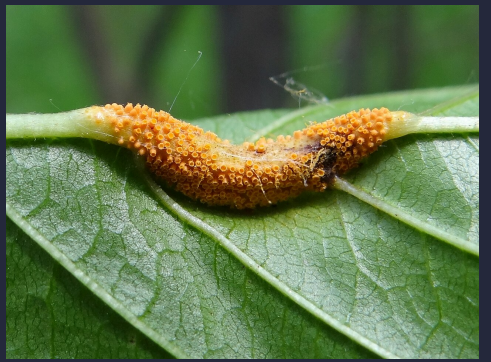
instead, memory safety errors and data races are prevented by the "borrow checker", which tracks the object lifetime of references at compile time."

~ Wikipedia

But why Rust?

Canadian software developer Graydon Hoare, created Rust in 2006 while working at Mozilla as a side project.

He named the language after a specific type of fungi that is "over-engineered for survival".



A brief history of Rust

Rust interpreter was first written in OCaml, and the language was inspired by programming languages from 1970-1990s, such as: CLU, BETA, Mesa, etc.

Hoare described Rust as 'technology from the past come to save the future from itself'

"Our target audience is "frustrated C++ developers". I mean, that's us.

So if you are in a similar situation we're in, repeatedly finding yourself forced to pick C++ for systems-level work due to its performance and deployment characteristics, but would like something safer and less painful,

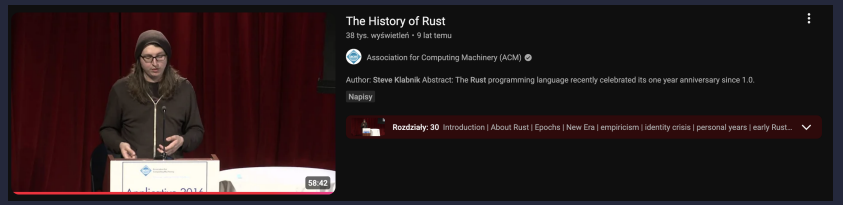
we hope we can provide that."

And they did.

Rust history over the years

- 2006, Rust is created in Mozilla Labs
- 2012, Rust compiler is rewritten in Rust
- 2014, The "Great Simplification", introducing Ownership/Borrowing instead of Garbage Collection
- 2015, Rust 1.0 is released
- 2020, Discord rewrote Read states from Go to Rust to eliminate Garbage Collection spikes
- 2022, Rust merged into Linux Kernel && Cloudflare replaced Nginx with Pingora, proxy written in Rust
- 2024, White House endorses Rust explicitly urging the industry to adopt memory safe languages to secure national infrastructure
- 2025, Rust no longer experimental in Linux Kernel

The History of Rust, by Steve Klabnik




```
1 echo https://www.youtube.com/watch?v=79PSagCD_AY | qrencode -t utf8i
```

snippet +exec is disabled, run with -x to enable

What's next?

- A brief history of Rust ✅
- Typesystem
- Memory Safety Core - Ownership, Borrowing, Lifetimes
- Heap and Stack
- Appliences
- How to start

Primitive types - integer

Size	Signed Type	Unsigned Type	Description
8-bit	i8	u8	Tiny integers (0 to 255 for u8), (-127 to 127 for i8)
16-bit	i16	u16	Small integers. (0 - 2^16 for u16)
32-bit	i32	u32	i32 is the standard default integer in Rust
64-bit	i64	u64	Large integers (64-bit precision).
128-bit	i128	u128	Massive integers for specialized math.
Arch-dependent	isize	usize	Matches your CPU pointer size (32 or 64-bit).

```
1    let ptr_size = std::mem::size_of::<usize>();
2
3    println!("My architecture is {}-bit!", ptr_size * 8);
```

snippet +exec is disabled, run with -x to enable

Primitive types - floats, bools, char

Floating-Points Types

Size	Type	Description
32-bit	f32	32 bit floating point
64-bit	f64	64 bit floating point, default if not specified otherwise

bool type

Size	Variant	Description
1 byte	true	It's true!
1 byte	false	It's false!

char type

```
let charek: char = 'c'; // single ticks!
```

Compound types - tuple

tuple

```
1     let ip_address: (u8, u8, u8, u8) = (127, 0, 0, 1);
2     let (a, b, c, d) = ip_address;
3
4     println!("Ain't no place like {}.{}.{}.{}", a, b, c, d);
5
6     println!("The first octet of an address - {}", ip_address.0);
```

snippet +exec is disabled, run with -x to enable

Compound types - array

- Array type

```
1     let brtdz = [2,1,1,5];
2     let zero_1d_tensor: [f64; 100] = [0.0; 100];
3
4     println!("{:?}", brtdz);
5     println!("{:?}", zero_1d_tensor);
```

snippet +exec is disabled, run with -x to enable

Compound types - Vec

- Vector

```

let first_valid_vec_of_i32: Vec<i32> = vec![-123, i32::MAX, 123, i32::MIN];
let second_valid_vec: Vec<i32> = Vec::new();

let mut stack: Vec<i32> = vec![0, 1, 3];
let popped_from_stack = stack.pop();

println!("Seems like Vec is just like array in python! {} == {}",
          3, popped_from_stack.unwrap());

```

snippet +exec is disabled, run with -x to enable

char, String, and string slice types

Type	Size	Description
String	24 bytes	growable, UTF-8 encoded text
&str	16 bytes	a "slice" or view into the text
char	4 bytes	a UTF-8 encoded scalar (f.e. U+1F680)
[u8; n]	n bytes	fixed size array of bytes
Vec<u8>	24 bytes	growable array of raw bytes, often used for non-textual data or manual encoding

```
1  let s = String::from("Hi🚀");
2  let view: &str = &s;
3  let char: char = s.chars().nth(2).unwrap();
4  let emoji: &[u8] = &s.as_bytes()[2..(view.len())];
5  let bytes_vec: Vec<u8> = s.as_bytes().to_vec();
6
7  println!("s: {:?}\nview: {:?}\nchar: {:?}\nemoji: {:?}\nbytes_vec: {:?}\n", s, view, char, emoji,
  bytes_vec);
8
```

snippet +exec is disabled, run with -x to enable

Functions must take in typed parameters

This won't compile:

```
fn eat_pizza(pizza) {
  if pizza.source == "piratto" {
    return "mmm yummy"
  } else {
    return "nice"
  }
}
```

snippet +exec is disabled, run with -x to enable

Functions must take in typed parameters

This will compile:

```
fn eat_pizza(pizza: Pizza) -> &'static str {
  if pizza.source == "piratto" {
    return "mmm yummy"
  } else {
    return "nice"
  }
}

  let questionmark_pizza: Pizza = Pizza { source: "piratto" };
  let result = eat_pizza(questionmark_pizza);
  println!("{}", result);
```

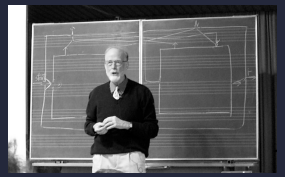
snippet +exec is disabled, run with -x to enable

There are no nulls in Rust

except for std::ptr::null() but we like to pretend that there aren't :)

There are no nulls in Rust

▒▒▒ Sir Charles Antony Richard Hoare (what a coincidence!), a british computer scientist, inventor of Quicksort algorithm and grandfather of "design by contract" famously said:



"I call it my billion-dollar mistake. It was the invention of the null reference in 1965. [...] This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years."

So how do we omit nulls?

```
Option<T>
or
Result<T, Err>
```

Option of type <T> has two wariants, either Some(T) or None.

Structs and Option<T> intro

```
struct Pizza {
  source: Pizzeria,
  cheese: String,
  meat: Option<Vec<MeatIngridient>>,
  sauce: Option<Sauce>,
  veggies: Option<Veggie>
}
```

Let's make a pizza - throw some meat

```
struct Pizza {
  source: Pizzeria,
  cheese: String,
  meat: Option<Vec<MeatIngridient>>,
  sauce: Option<Sauce>,
  veggies: Option<Veggie>
}

struct MeatIngridient(String)
```

Let's make a pizza - source please?

```
struct Pizza {
  source: Pizzeria,
  cheese: String,
  meat: Option<Vec<MeatIngridient>>,
  sauce: Option<Sauce>,
  veggies: Option<Veggie>
}

struct MeatIngridient(String)

enum Pizzeria {
  PIRATTO,
  SOPRANO,
  DOLCEVITA,
  PARMA
}
```

Let's make a pizza - other ingridients

```
struct Pizza {
  source: Pizzeria,
  cheese: String,
  meat: Option<Vec<MeatIngridient>>,
  sauce: Option<Sauce>,
  veggies: Option<Veggie>
}

struct MeatIngridient(String)

enum Pizzeria {
  PIRATTO,
  SOPRANO,
  DOLCEVITA,
  PARMA
}

enum Sauce {
  BIANCO,
  ROSSO,
  WLOCLAWEK,
}

struct Veggie(String)
```

Let's make a pizza - specyfing values

```
let source: Pizzeria = Pizzeria::SOPRANO;

let cheese: String = "Mozzarella".to_string();

let meat: Vec<MeatIngridient> = vec![MeatIngridient("ham".to_string())];

let sauce: Option<Sauce> = Some(Sauce::ROSSO);
let no_sauce: Option<Sauce> = None;

let veggies: Option<Veggie> = Some(Veggie("rocket".to_string()));
```

Let's order a pizza

```
let source: Pizzeria = Pizzeria::SOPRANO;
let cheese: String = "Mozzarella".to_string();
let meat: Option<Vec<MeatIngridient>> = Some(vec![MeatIngridient("ham".to_string())]);
let sauce = Some(Sauce::ROSSO);
let veggies = Some(Veggie("rocket".to_string()));

let my_wonderful_pizza: Pizza = Pizza {
            source,
            cheese,
            meat,
            sauce,
            veggies
};

println!("{:?}", my_wonderful_pizza);

```

snippet +exec is disabled, run with -x to enable

Result<T, E>

(valid python code)

```python
try:
  this_is_going_to_work(data)
except Exception as e:
  print(f"Oh no, the function crashed :( bohoo good luck now {e}")
finally:
  whatever_dude_it_works_either_way(data)
```

Result<T, E> - NO NULL SURPRISES

```rust
enum ConfigError {
    FileNotFound,
    EmptyFile,
}
                            // Either Ok(String) or Err(ConfigError)
fn read_token(filename: &str) -> Result<String, ConfigError> {
    println!("reading {}", filename);
    let contents = ""; // let's pretend the file is empty
    if contents.is_empty() {
        return Err(ConfigError::EmptyFile); // explicit return
    }
    Ok("secret_token_42".to_string()) // We don't need to put return here
}

fn main() {
    let filename = "config.txt";
    match read_token(filename) {
        Ok(token) => println!("Logged in with: {}", token),
        Err(e) => match e {
            ConfigError::FileNotFound => println!("Error: Please create config.txt"),
            ConfigError::EmptyFile => println!("Error: The config file is blank"),
        },
    }
}
```

snippet +exec is disabled, run with -x to enable

Elephant in the room

infamous `.unwrap()`

YouTube · The PrimeTime
Wyświetlenia: ponad 424,2 tys. · 2 miesiące temu

The line of code that took down the Internet

They should rewrite it in **Rust** (again). 10:08 · Go to channel The PrimeTime · How React took down **Cloudflare**. The PrimeTime•356K views.

Hackaday
https://hackaday.com › 2025/11/20 · Tłumaczenie strony

How One Uncaught Rust Exception Took Out Cloudflare

20 lis 2025 — **Blaming Rust the language as a whole for the Cloudflare outage** is like blaming Michelin when you crash your car for driving too fast in the ...

Hacker News
https://news.ycombinator.com › i... · Tłumaczenie strony

Cloudflare outage on November 18, 2025 post mortem

18 lis 2025 — This is a failure caused by lazy **Rust** programming and not relying on the language's design features. It's a shame this code can even be written.

unwrap() on None type

```
let may_or_may_not_be: Option<u8> = None;

println!("That doesn't break :D also its side effect for logging! {}", may_or_may_not_be.unwrap());
```

snippet +exec is disabled, run with -x to enable

unwrap() on Err type

```
let it_cannot_break_right: Result<u8, ImportantError> = Err(ImportantError::ParsingError);

println!("That doesn't break :D also its side effect for logging! {}", it_cannot_break_right.unwrap());
```

snippet +exec is disabled, run with -x to enable

Rule of Thumb

```
.unwrap() <- NEVER in production, ok if you're learning
.expect("should work because xyz") <- for debugging, but NEVER in production
```

```
let x: Option<u8> = 42;

if let Some(my_value) = some_option_type {
  perform operations
} else {
  handle grafecully
}

match my_value {
  Some(n) => {
    perform operations
  },
  None => {
    handle gracefully
  }
}
```

Exhaustive pattern matching

```
enum WatchBrand {
  OMEGA,
  GRANDSEIKO,
  TAGHEUER,
  ROLLEYJEDEN,
  ROLLEYDRHUGI
}

let watch: WatchBrand = WatchBrand::GRANDSEIKO;

let answer = match watch {
  WatchBrand::OMEGA => "It's not Omega",
  WatchBrand::GRANDSEIKO => "ok ok",
  WatchBrand::TAGHEUER => "nice",
};

println!("{}", answer);
```

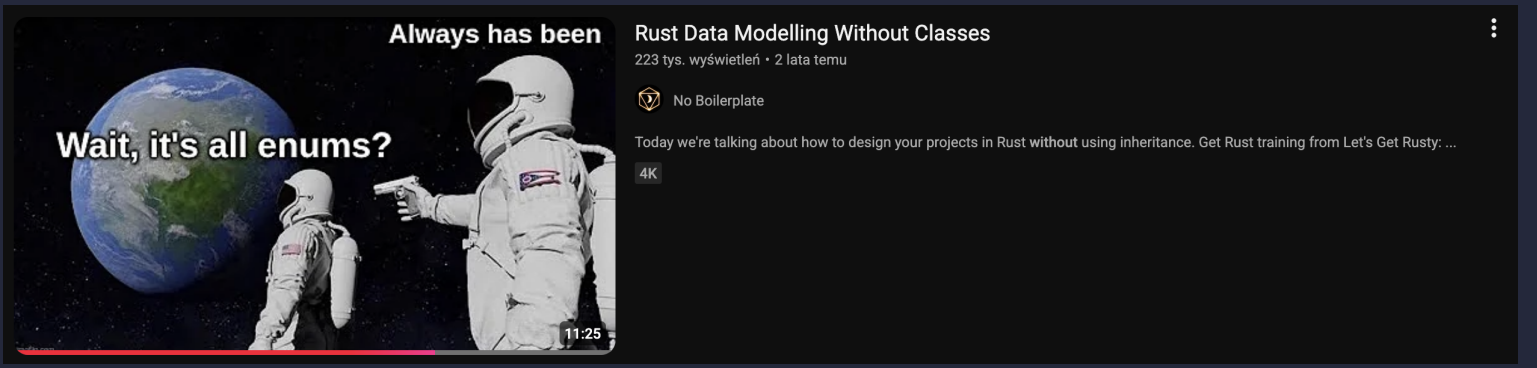
snippet +exec is disabled, run with -x to enable

Exhaustive pattern matching

```
enum WatchBrand {
  OMEGA,
  GRANDSEIKO,
  TAGHEUER,
  ROLLEYJEDEN,
  ROLLEYDRHUGI
}

let watch: WatchBrand = WatchBrand::GRANDSEIKO;

let answer = match watch {
  WatchBrand::OMEGA => "It's not Omega",
  WatchBrand::GRANDSEIKO => "ok ok",
  WatchBrand::TAGHEUER => "nice",
  WatchBrand::ROLLEYJEDEN | WatchBrand::ROLLEYDRHUGI => "feeling old yet?"
};

println!("{}", answer);
```

snippet +exec is disabled, run with -x to enable

More on the type system




```
1 echo
  https://www.youtube.com/watch?v=z-0-bbc80JM
  | qrencode -t utf8i
```

```
snippet +exec is disabled, run with -x to enable
```

What's next?

- A brief history of Rust ✅
- Typesystem ✅
- Memory Safety Core - Ownership, Borrowing, Lifetimes
- Heap and Stack (in short)
- Appliences
- How to start

Ownership introduction

How do programming languages manage heap allocated data?

Historically, you had two choices.

Garbage Collection

(JVM family, Python, Go, C#, JavaScript, TypeScript, Haskell, Lisp, Clojure, OCaml, F#, Erlang, Elixir, Ruby, PHP, Swift, D, Julia):

- constant memory scans
- conventional - you just don't care
- BUT unpredictable performance overhead

Manual Management

(C, C++, Zig):

- you must explicitly allocate and free memory
- you gain maximum control and performance
- it's prone to human error

Ownership

(Rust):

Rust uses RAII (Resource Acquisition Is Initialization).

When a variable goes out of scope, Rust automatically calls a special drop function to release the memory.

- you get memory safety for cost of learning curve
- memory, files, sockets are closed EXACTLY when you expect them to be
- doubly linked list
- long compile times (but there are no free lunches)

Ownership example

```rust
fn main() {
    // MAIN SCOPE BEGINS
    let s1 = String::from("Outer Scope"); // s1 is valid

    { //  INNER SCOPE STARTS
        let s2 = String::from("Inner Scope"); // s2 is valid

        println!("We can use both: {} and {}", s1, s2);

    } // INNER SCOPE ENDS, s2 is dropped

    println!("Back to outer: {}", s1); // s1 is still valid.
}
// Main scope ends, s1 is dropped
```

Ownership example

```rust
fn main() {
    // MAIN SCOPE BEGINS
    let s1 = String::from("Outer Scope");

    { //  INNER SCOPE STARTS
        let s2 = String::from("Inner Scope"); // s2 is valid
        println!("We can use both: {} and {}", s1, s2);

    } // INNER SCOPE ENDS

    println!("{}", s2); // LETS HAVE FUN!

    println!("Back to outer: {}", s1);
}
```

snippet +exec is disabled, run with -x to enable

Borrowing

```
let mut cust = Customer {
  name: "Rust Enjoyer".to_owned(),
  age: 21,
};

let age = &cust.age; // first immutable borrow
let second_age = &cust.age; // second immutable borrow

// let r3 = &mut cust; this makes uncompilable code :(

if age >= &10 && *second_age <= i32::MAX {
  println!("{} and {}", age, second_age);
} // r1 and r2 "die" here after their last use
  // cust is still valid

let r3 = &mut cust; // no other active borrows
r3.name = "Johnny Bravo".to_string();

println!("mutable reference {:?}", r3);
println!("accessing directly {:?}", cust);
```

snippet +exec is disabled, run with -x to enable

Lifetimes pt.1

```
fn longest(x: &str, y: &str) -> &str {
    if x.len() > y.len() {
        x
    } else {
        y
    }
}
```

snippet +exec is disabled, run with -x to enable

Lifetimes pt.2

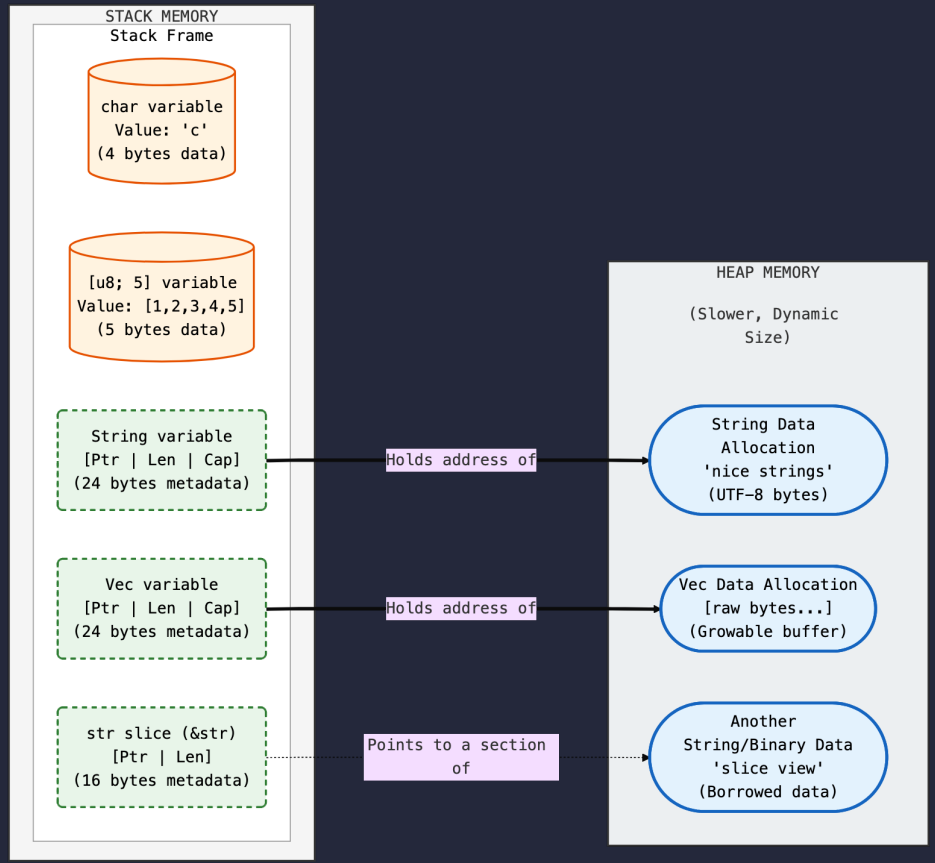
```
fn longest<'a>(x: &'a str, y: &'a str) -> &'a str {
    if x.len() > y.len() {
        x
    } else {
        y
    }
}
let x = "googoo gaga";
let y = "lady gaga";
println!("{}", longest(x, y));
```

snippet +exec is disabled, run with -x to enable

Tired yet?

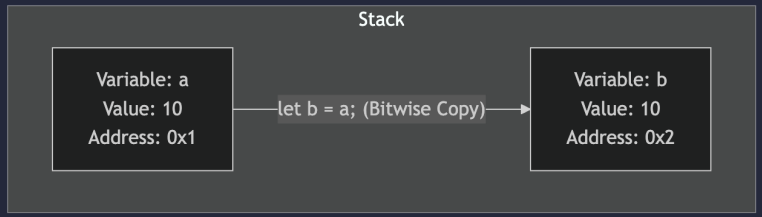
- A brief history of Rust ✅
- Typesystem ✅
- Memory Safety Core - Ownership, Borrowing, Lifetimes ✅
- Heap and Stack
- Appliences
- How to start

Heap and Stack



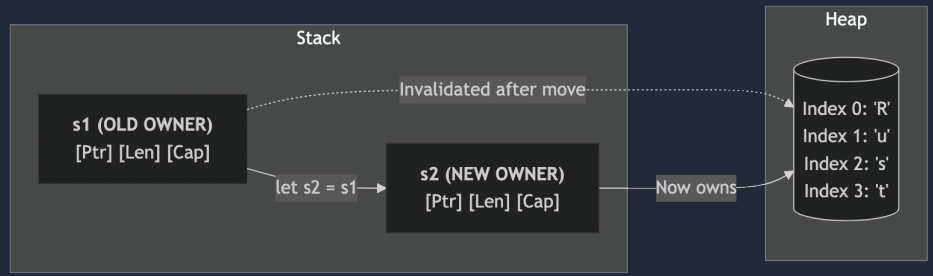
Stack memory allocation

```rust
let a: i32 = 10;
let b = a;
```



Heap memory allocation

```
let s1 = String::from("Rust");
let s2 = s1;
```



Please, enough with the code, why should I care?

- A brief history of Rust ✅
- Typesystem ✅
- Memory Safety Core - Ownership, Borrowing, Lifetimes ✅
- Heap and Stack ✅
- Appliences
- How to start

Appliences, part one

Async:

- tokio, de-facto standard async runtime, a true gem of Rust
- smol, small, fast, and easy to understand - great for CLI tooling
- embassy, primary choice for async on embedded devices

Web & Networking:

- axum, web framework maintained by the team behind tokio
- actix-web, actor based framework that consistently tops benchmarks
- hyper, http implementation, backbone of two above

Databases & Data Handling

- sqlx, my weapon of choice. Pure Rust, async, and compile time checked SQL queries.
- SeaORM, built on top of SQLx, for Django-like or TypeORM enjoyers
- mongodb, highly performant mongo driver

Appliences, part two

Data engineering:

- Polars, so called "pandas killer"
- Apache DataFusion, a powerful SQL query engine
- delta-rs, a native Rust interface for Delta Lake

Machine Learning:

- burn, PyTorch-like DL framework
- candle, lightweight ML framework
- limfa, the scikit-learn of rust
- tch-rs, a high level wrappers of C++ libtorch

Appliences, part three

WebAssembly and Fullstack

- Leptos, imagine SolidJS but running on native speed
- Dioxus, Multi-platform UI, Write once - run everywhere
- Yew, the long standing veteran, pure WASM framerwork
- Wasm-bindgen, the bridge between Rust and JS. It handles "the dirty work"

CLI

- Clap, argparse of Rust, you don't need much else

Appliences, part four

Ratatui

Terminal User Interface application, lately enabled to compile bare metal.

I say it's the future of embedded device development.

This presentation is written in presenterm, which utilizes Ratatui to interact with terminal.

Our friend Orhun Parmaksiz is the creator behind Ratatui, and posts amazing projects that people create using Ratatui on his LinkedIn feed. Besides, he livestreams Ratatui coding on youtube and recently created Tuitar, open source tuner for your electric guitar.

https://www.linkedin.com/in/orhunp/

Bevy

Open source game engine written in Rust. It's way too awesome. Please check it out.

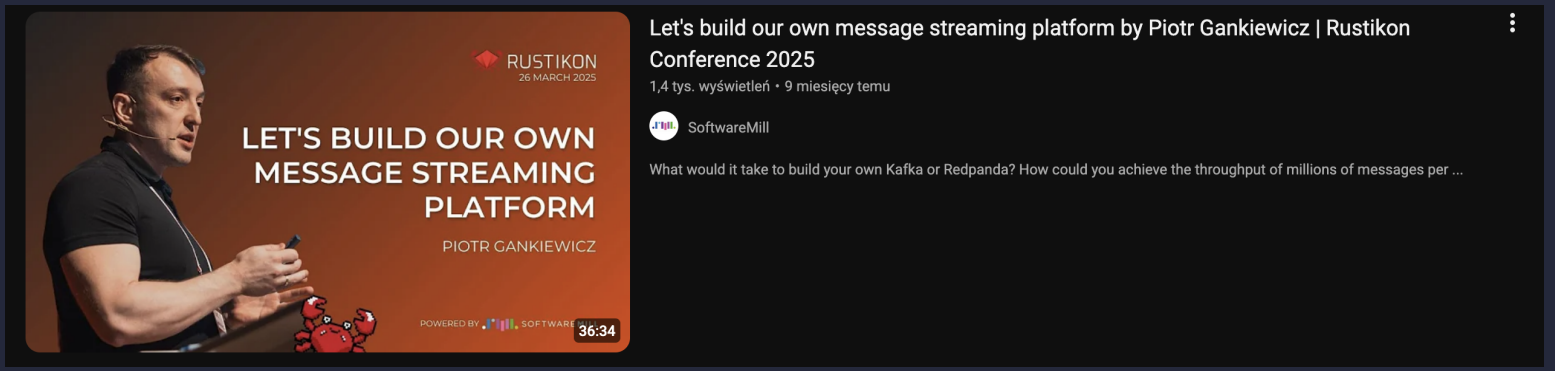
https://bevy.org/

Apache Iggy (iggy.apache.org)

I must mention Apache Iggy. Written by Piotr Gankiewicz, is a high-performance, persistent message streaming platform written in Rust.

I highly suggest familiarizing yourself with this platform, especially if you work with Kafka/redpanda.

Apache Iggy




```
1 echo https://www.youtube.com/watch?v=GkV306PyvqM | qrencode -t utf8i
```

```
snippet +exec is disabled, run with -x to enable
```

Python tools written in Rust

uv

(pip, poetry, virtualenv)

- replaces pip, enables workspaces and generates lockfile for better versioning
- heavily inspired by cargo, a Rust package manager

Ruff

(Flake8)

- Linter/formatter that processes code on every keystroke

ty

(mypy, Pyright, Pylance)

- LSP (Language Server Protocol)
- uses an incremental architecture to give real-time feedback 10–100x faster than mypy

polars

(pandas)

- Apache arrow + Rust concurrency, what else do you need?

pydantic

(pydantic)

- the core validation logic was moved to pydantic-core, making it 17x faster

cryptography

(cryptography)

- used by requests, ssh
- has migrated its low-level math to Rust for memory safety

JavaScript tools written in Rust

- SWC

(Babel)

  - A compiler that is ~20x faster than Babel, powers Next.js and Deno

- Turbopack

(webpack)

  - A successor

- biome

(prettier + ESLint)

  - single binary that formats and lints JS/TS projects instantly

- Tauri

(Electron) made the installers weight ~3MB instead of ~100MB

Companies openly using Rust in their backend

Global

1. Discord
2. Cloudflare
3. Dropbox
4. AWS
5. Google
6. Microsoft
7. Meta
8. Figma
9. Shopify

Our backyard

1. Neptune.ai
2. Air Space Intelligence
3. Gielda Papierow Wartosciowych
4. Synerise
5. Cosmose AI
6. Golem network
7. Codility
8. Anixe

and the number is growing :)

What's next?

- A brief history of Rust ✅
- Typesystem ✅
- Memory Safety Core - Ownership, Borrowing, Lifetimes ✅
- Heap and Stack ✅
- Appliences ✅
- How to start

So where to start my Rust journey?

The book

rustup doc --book

or

https://doc.rust-lang.org/book/

Rustlings

https://rustlings.rust-lang.org/

Rust by Example

https://doc.rust-lang.org/rust-by-example/

Get inspired

- No Boilerplate
- Code To The Moon

Allow compiler to be your friend - don't be mad if your code doesn't compile, it's a feature

How would I approach learning Rust again?

- Go through the book, get bored/excited with theory, do some rustlings
- Read the documentation - the answers are there
- Go solve some easy tasks on leetcode/hackerrank and familiraize with iterators
- Take your time!
- Create

Cargo


```
1 echo https://www.youtube.com/watch?v=wQ_OrmE_AEY | qrencode -t utf8i
```

```
snippet +exec is disabled, run with -x to enable
```

Thank you

stay curious

presentation repository

find me on linkedin

follow Rust Poland