

Many ways to learn Rust

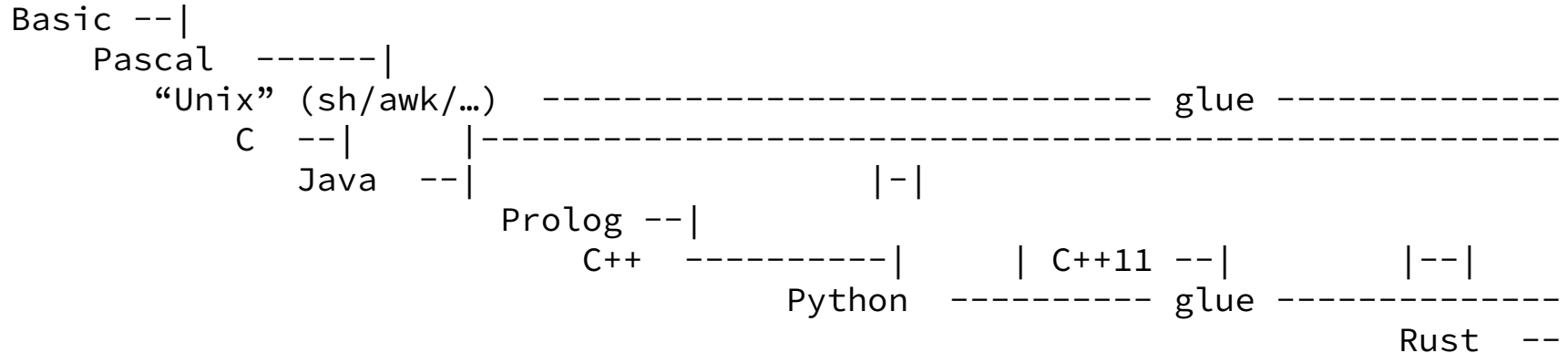
Daniel Thompson
Linaro



Biography: Daniel Thompson

I do many things for Linaro and one of them is that I get to wear the **training manager's** hat. **I learned Rust** whilst wearing that hat. That means **I'm carrying** a lot of **baggage**. However it also means **I have spent** a lot of **time thinking about** how to teach Rust and thinking about **how we can learn** it.

I'm not going to make any attempt to teach Rust today. Instead this talk contains my **reflections on learning Rust**.



What you might already know about Rust

In decreasing order of the probability that you do, in fact, already know it:

Rust is...

... memory-safe

... performant

... hard to learn

... productive

What you might already know about Rust

In decreasing order of probability (that you, in fact, do already know it):

Rust is...

... memory-safe

... performant

... **hard to learn?**

... productive

Dogmatic compiler

People told you scary things about the borrow checker

Rust is commonly introduced to solve hard problems

It's different (a.k.a. C made you think type annotations come first and that integer promotion is a good thing)

Traits and generics are intertwined

Rust is different

Being different is not the same as being hard to learn... but if you have spend the last [five|ten|twenty|forty] years using only languages with strong C influences it will seem that way.

Rust is strongly influenced by OCaml and inherits several elements of OCaml's look and feel (including it's type annotations).

Rust brings many things that feel alien to those with a C background:

- Variables are immutable by default
- Data structures move by default
 - Function arguments are passed neither by-value nor by-reference
- Variables can be rebound
 - Same name - different types
- References can be cloned
 - Same name - different variable
- Variety of error handling syntax

The big three - rust-lang.org/learn

“the book”

The Rust Programming Language is a narrative introduction to Rust. It is well structured and introduces most, but not all, of the language.

It can be accessed in HTML form (free) and in paperback or eBook format (paid).

rustlings

Introduces Rust features in the form of bite sized puzzles: typically fixing a compiler error or writing a function that passes a test.

Helps you get used to the syntax and the compiler.

Rust by example

Well structured collection of Rust code examples. Mostly organized by language or standard library features.

Makes a great companion for the book. Especially when you can't remember how to spell whatever it was that you read in the book a month ago.

Let the tools teach you Rust

Put simply, you **must** integrate rust-analyzer into your editor (or change editors).

rust-analyzer is a “Rust compiler front-end for IDEs”. It provides check-as-you-type, type information (as tooltips or inlay hints), function signature info, auto-complete, ...

Rust is all about compile-time checking and there are lots of checks beyond borrow checking. The compiler is strict but it is on your side: the quality of its diagnostics is very good making the compiler a good teacher.

Configuring your editor to use rustfmt to format-on-save is also strongly recommended. Especially when you are starting out since becoming used to the default Rust style is useful.

Let clippy teach you Rust

Clippy is an a linting tool, that provided over 700 new checks to help you write better, more idiomatic Rust code. Try: `rustup component add clippy` and `cargo clippy`

```
5 fn sum(nums: &Vec<i32>) -> i32 {  
6     nums.iter().sum()  
7 }
```

↓

```
5 fn sum(nums: &[i32]) -> i32 {  
6     nums.iter().sum()  
7 }
```

```
> cargo clippy  
warning: writing `&Vec` instead of `&[_]` involves a new  
object where a slice will do  
--> src/main.rs:5:14  
|  
5 | fn sum(nums: &Vec<i32>) -> i32 {  
|                               ^^^^^^^^^^^ help: change this to: `&[i32]`  
|  
= help: for further information visit  
https://rust-lang.github.io/rust-clippy/master/index.html#  
ptr\_arg  
= note: `#[warn(clippy::ptr_arg)]` on by default
```


Let evcxr teach you rust

```
sh$ cargo install evcxr_repl
  Updating crates.io index
```

```
  ...
  Compiling evcxr_repl v0.14.2
  Finished release [optimized] target(s) in 2m 43s
```

```
sh$ evcxr
```

```
Welcome to evcxr. For help, type :help
```

```
>> let x = -5;
```

```
>> if x > 0 { x } else { -x } as u32
```

```
Error: expected expression, found `as`
```

```
[command:1:1]
1 | if x > 0 { x } else { -x } as u32
  |                                     ^ expected expression
```

```
>> (if x > 0 { x } else { -x }) as u32
```

```
5
```

Make Rust your scripting language

```
#!/usr/bin/env rustc          #!/usr/bin/env rust-script # Create a playground
// The rust compiler          // rust-script can be      cargo init $HOME/toybox
// treats #! on the first     // installed using cargo.  cd $HOME/toybox
// line as a comment.        // Special comments give    mkdir src/bin
// We only have the std       // rust-script scripts      mv src/main.rs \
// library so I'd have to     // access to crates:        src/bin/hello.rs
// write a CSV parser from    // cargo-deps: csv          # Run hello from toybox
// scratch.                   //                               cargo run --bin hello
                               # cargo auto-discovers
                               # all programs found in
                               # src/bin
                               cargo add csv
                               vim src/bin/csv2email.rs

use std::io;                  use csv::Reader;
                               use std::io;

fn main () {                  fn main () {
    ...
}                               }
```

Make Rust your scripting language

```
use serde::Deserialize;
use std::{error::Error, io};

#[derive(Debug, Deserialize)]
struct MailAddr {
    #[serde(rename = "Name")] name: String,
    #[serde(rename = "Email")] email: String,
}

fn main() -> Result<(), Box<dyn Error>> {
    let mut reader =
        csv::Reader::from_reader(io::stdin());
    for row in reader.deserialize() {
        let row: MailAddr = row?;
        println!("{}", row.name, row.email);
    }
    Ok(())
}
```

```
import csv
import sys

reader = csv.DictReader(sys.stdin)

for row in reader:
    name = row["Name"]
    email = row["Email"]
    print(f"{name} {email}")
```

Make Rust your scripting language

```
use serde::Deserialize;
use std::{error::Error, io};

#[derive(Debug, Deserialize)]
struct MailAddr {
    #[serde(rename = "Name")] name: String,
    #[serde(rename = "Email")] email: String,
}

fn main() -> Result<(), Box<dyn Error>> {
    let mut reader =
        csv::Reader::from_reader(io::stdin());
    for row in reader.deserialize() {
        let row: MailAddr = row?;
        println!("{}", <{}>, row.name, row.email);
    }
    Ok(())
}
```

```
import csv
import sys
```

```
reader = csv.DictReader(sys.stdin)
for (lineno, row) in enumerate(reader):
    try:
        name = row["Name"]
        email = row["Email"]
    except KeyError as exc:
        raise ValueError(
            f"Bad line {lineno+2}") from exc
    print(f"{name} <{email}>")
```

Aside: the right amount of curious

Run towards the flames

If something doesn't feel right then figuring out why is an important part of learning.

A good example occurs when navigating data structures containing optional values.

Your code to handle the None case might feel bulky or like you are charring into the right margin. Consider that a clue to go and (re-)read [the reference documentation](#)

Clue: it's usually something that takes a closure.

Don't go down the rabbit hole

By making Rust your scripting language (or finding other reasons to practice learning the language) you are accepting a productivity trade-off versus \$OLDLANG. Be careful not to get sucked in.

For example, derive macros are super elegant and are part of what makes writing Rust code more ergonomic. You don't need to understand how they work on day 1 (or day 60).

Advent of Code

Advent of Code is collection of small programming puzzles, released once-per-day during December (but accessible throughout the year).

It's a great way to learn any new language.

In the case of Rust, taking part in Advent of Code it is a great way to better understand what sort of code is “hard” to borrow check.

```
Advent of Code [About] [Events] [Shop] [Settings] [Log Out] Daniel Thompson
0x0000|2023 [Calendar] [AoC++] [Sponsors] [Leaderboard] [Stats]

increment: The action of increasing or becoming greater.

Advent of Code is free to use.

However, building and running Advent of Code takes a significant amount of
my time and energy. If you like Advent of Code, would like more things like
it in the future, and are able to do so, please support Advent of Code.

Your contributions support things like:

- Infrastructure (hosting, bandwidth, etc)
- Time spent running Advent of Code events
- Time spent building puzzles
- More future projects
- My sushi addiction

Because you are logged in, if you proceed, you will get a supporter badge
next to your name on all Advent of Code 2023 pages. You can switch to a
different event on the [Events] page.

You can support Advent of Code via [PayPal] or, if you must, [Coinbase].

(Amounts under $1 barely reach me after fees and will not confer a badge.
Please be patient; some transactions, especially using cryptocurrencies,
can take an hour or two to complete.)

--- Supporter History ---

You have not yet supported any Advent of Code event.
```

Our sponsors help make Advent of Code possible:

DEPT - Code whisperers doubling as marketing mavens. We're the missing link between tech wonders and brand bling, bringing pixel-perfect dreams to life.

Embrace cargo

Cargo and the crates.io ecosystem allows Rust to keep the scope of the standard library small. There is no intent to extend the Rust standard library into a Python-like “batteries included” library.

When library code is required an experienced Rustacean will review crates.io looking for well-maintained libraries that can help.

An alternative way to view this is that cargo+crates.io are the included batteries.



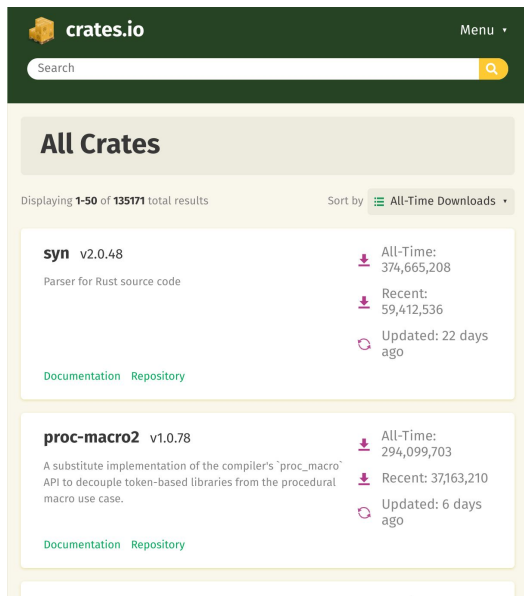
std



std + crates.io

Embrace cargo

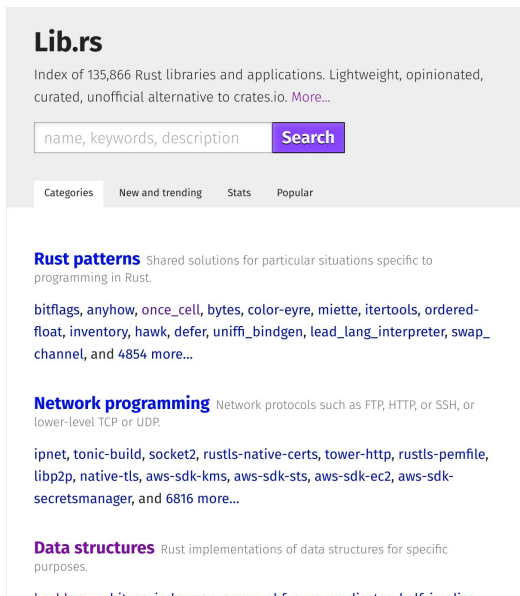
So... the batteries come from crates.io but how can I figure out which ones to use?



The screenshot shows the crates.io website interface. At the top, there is a search bar and a menu. Below the search bar, the text "All Crates" is displayed. Underneath, it says "Displaying 1-50 of 13571 total results" and "Sort by All-Time Downloads". Two crate entries are visible:

- syn v2.0.48**: Parser for Rust source code. All-Time: 374,665,208. Recent: 59,412,536. Updated: 22 days ago. Links: Documentation, Repository.
- proc-macro2 v1.0.78**: A substitute implementation of the compiler's 'proc_macro' API to decouple token-based libraries from the procedural macro use case. All-Time: 294,099,703. Recent: 37,163,210. Updated: 6 days ago. Links: Documentation, Repository.

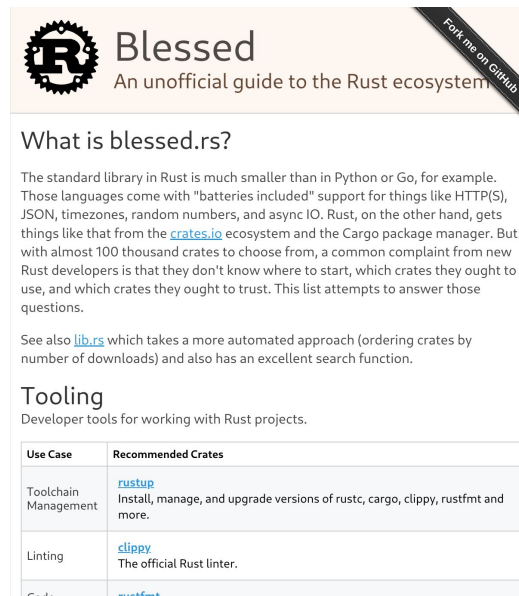
<https://crates.io>



The screenshot shows the lib.rs website interface. At the top, the title "Lib.rs" is displayed, followed by the description: "Index of 135,866 Rust libraries and applications. Lightweight, opinionated, curated, unofficial alternative to crates.io. More...". Below this is a search bar with the placeholder text "name, keywords, description" and a "Search" button. Underneath the search bar, there are tabs for "Categories", "New and trending", "Stats", and "Popular". Three categories are listed:

- Rust patterns**: Shared solutions for particular situations specific to programming in Rust. Examples: bitflags, anyhow, once_cell, bytes, color-eyre, miette, itertools, ordered-float, inventory, hawk, defer, uniffi_bindgen, lead_lang_interpreter, swap_channel, and 4854 more...
- Network programming**: Network protocols such as FTP, HTTP, or SSH, or lower-level TCP or UDP. Examples: ipnet, tonic-build, socket2, rustls-native-certs, tower-http, rustls-pemfile, libp2p, native-tls, aws-sdk-kms, aws-sdk-sts, aws-sdk-ec2, aws-sdk-secretsmanager, and 6816 more...
- Data structures**: Rust implementations of data structures for specific purposes. Examples: hashbrown, bitvec, indoc, once_cell, slab, smallvec, half, im-vec, and 1000 more...

<https://lib.rs>



The screenshot shows the blessed.rs website interface. At the top, there is a logo for "Blessed" and the text "An unofficial guide to the Rust ecosystem". Below this is a section titled "What is blessed.rs?" with the following text: "The standard library in Rust is much smaller than in Python or Go, for example. Those languages come with 'batteries included' support for things like HTTP(S), JSON, timezones, random numbers, and async IO. Rust, on the other hand, gets things like that from the crates.io ecosystem and the Cargo package manager. But with almost 100 thousand crates to choose from, a common complaint from new Rust developers is that they don't know where to start, which crates they ought to use, and which crates they ought to trust. This list attempts to answer those questions." Below this text is a link to "lib.rs" which takes a more automated approach (ordering crates by number of downloads) and also has an excellent search function. Below the text is a section titled "Tooling" with the text "Developer tools for working with Rust projects." Below the text is a table with the following content:

Use Case	Recommended Crates
Toolchain Management	rustup Install, manage, and upgrade versions of rustc, cargo, clippy, rustfmt and more.
Linting	clippy The official Rust linter.
Code	rustfmt

<https://blessed.rs>

Make the right first steps

Rust “likes” ...

- ... chewing through strings and vectors
- ... other “easy” memory management
- ... functional programming
- ... you to build on best-in-class crates
- ... finding problems for you

Rust doesn't like...

- ... ~~GUI~~ libraries (maybe)

Rust loves...

- ... thread-safety
(so much so it might be easier to be single threaded when learning the syntax)

A miscellany

Write in Rust

Rust is not C or C++ or JavaScript or Python or Modula-2

Don't write C code using Rust syntax (c.f. [Run towards the flames](#))

Read some Rust

Browse production Rust code (and try to figure out if its good/bad/meh)

If you are a kernel programmer spend a little time in userspace

Listen and reflect

Cargo and crates.io as “a thing”, Rust code defaults to static linking, Rust has strict boundaries for dynamic libraries, Rust compile times are long, <insert your favourite *Rust sucks because here*>

Some great YouTube content

Channels

[Rust Talks by No Boilerplate](#)

[Crust of Rust by Jon Gjengs](#) - long-form tutorial videos by the author of the (excellent) [Rust for Rustaceans](#) book

Single videos

["Type-Driven API Design in Rust"](#) by Will Crichton

Online reading list

- [“the book”](#)... yes... again (especially chapter 15)

- Other important Rust “books”

- [Asynchronous Programming in Rust](#)
- [the Cargo book](#)
- [Command Line Applications in Rust](#)
- [“the nomicon”](#) (unsafe Rust)

- [The Rust Standard Library](#) and [docs.rs](#)

- [std::iter::Iterator](#) plus maybe the [itertools](#) crate
- [std::string::String](#)
 - Comparing String to [str](#) primitive is also useful to learn how to navigate docs

```
use itertools::Itertools;

fn main() {
    std::io::stdin()
        .lines()
        .filter_map(|ln| ln.ok())
        .sorted_by_key(|ln| ln.len())
        .for_each(|ln| println!("{}", ln));
}
```

How to teach yourself Rust

Read “**the book**” from cover to cover



Bring an **empty cup**



Use the **tools**



Make the right **first steps**



Be the right amount of **curious**

Get **idiomatic**



 Linaro Connect
MADRID 2024 | MAY 12-17 2024



support@linaro.org
training@linaro.org

