Rust

A 20-Minute Introduction

Brendan Dagys

2022-08-26

Outline

1. *What is Rust?*
2. *What is Rust used for?*
3. Introduction to key concepts
4. Cool stuff
5. Demo
6. More resources
7. Questions

What Is Rust?

- **General-purpose, cross-platform, multi-paradigm language**
- Statically typed
- Compiled language
- Emphasizes performance, type safety, and concurrency
- Enforces memory safety without a garbage collector

Multi-Paradigm Programming Languages

Language overview [edit]

List of multi-paradigm programming languages

Language	Number of Paradigms	Concurrent	Constraints	Data-flow	Declarative	Distributed	Functional	Meta-programming	Generic	Imperative	Logic	Reflection	Object-oriented	Pipe-lines	Visual	Rule--based	Other paradigms
Rust (version 1.0.0-alpha)	6	Yes[a 7]	No	No	No	No	Yes	Yes[111][112]	Yes[113]	Yes	No	No	Yes	No	No	No	linear, affline, and ownership types
Ruby	5	No	No	No	No	No	Yes	Yes	No	Yes	No	Yes	Yes[a 2]	No	No	No	No
R	4 (6)	Library[95]	No	No	No	Library[96]	Yes	No	No	Yes	No	Yes	Yes	Yes[97]	No	No	Array (multi-dimensional)
Python	5 (10)	Library[85][86]	Library[87]	No	No	Library[88]	Partial	Yes[89][90]	Yes[91][92]	Yes	Library[93]	Yes	Yes[a 2]	No	Editor[94]	No	structured
PHP[82][83][84]	4	No	No	No	No	No	Yes	No	No	Yes	No	Yes	Yes[a 2]	No	No	No	No
Perl[*citation needed*]	8 (9)	Yes[79]	Library[80]	Yes[81]	No	No	Yes	Yes	No	Yes	No	Yes[a 2]	Yes[a 2]	Yes	No	No	No
MATLAB	6 (10)	Toolbox[70]	Toolbox[71]	Yes[72]	No	Toolbox[73]	No	Yes[74]	Yes[75]	No	No	Yes[76]	Yes[77]	No	Yes[78]	No	Array (multi-dimensional)
Java	6	Yes	Library[60]	Library[61]	No	No	Yes	No	Yes	Yes	No	Yes	Yes[a 2]	No	No	No	No
Haskell	8 (15)	Yes	Library[54]	Library[55]	Yes	Library[56]	Yes (lazy)	Yes[57]	Yes	Yes	Library[58]	No	Immutable	Yes	Yes	Library[59]	literate, reactive, dependent types (partial)
Go	4	Yes	No	No	No	No	No	No	No	Yes	No	Yes	No	Yes	No	No	No
Fortran	4 (5)	Yes	No	No	No	No	Yes[a 13]	No	Yes[a 14]	No	No	No	Yes[a 2]	No	No	No	Array (multi-dimensional)
ECMAScript[46][47] (ActionScript, E4X, JavaScript, JScript)	4 (5)	partial (promises, native extensions) [a 8]	No	No	Library[48][49]	No	Yes	No	No	Yes	No	Yes	Yes[a 9]	Library[50][51]	Editor[52]	No	reactive,[a 10][53] event driven[a 11][a 12]
C++	7 (15)	Yes[7][8][9]	Library[10]	Library[11][12]	Library[13][14]	Library[15][16]	Yes	Yes[17]	Yes[a 3]	Yes	Library[18][19]	Library[20]	Yes[a 2]	Yes[21]	No	Library[22]	Array (multi-dimensional; using STL)
C#	6 (7)	Yes	No	Library[a 4]	No	No	Yes[a 5]	No	Yes	Yes	No	Yes	Yes[a 2]	No	No	No	reactive[a 6]

Figure 1. List of multi-paradigm programming languages (https://en.wikipedia.org/wiki/Comparison_of_multi-paradigm_programming_languages)

What Is Rust?

- General-purpose, cross-platform, multi-paradigm language
- **Statically typed**
- **Compiled language**
- **Enforces memory safety without a garbage collector**
- **Emphasizes performance, type safety, and concurrency**

What Is Rust? - History to Present

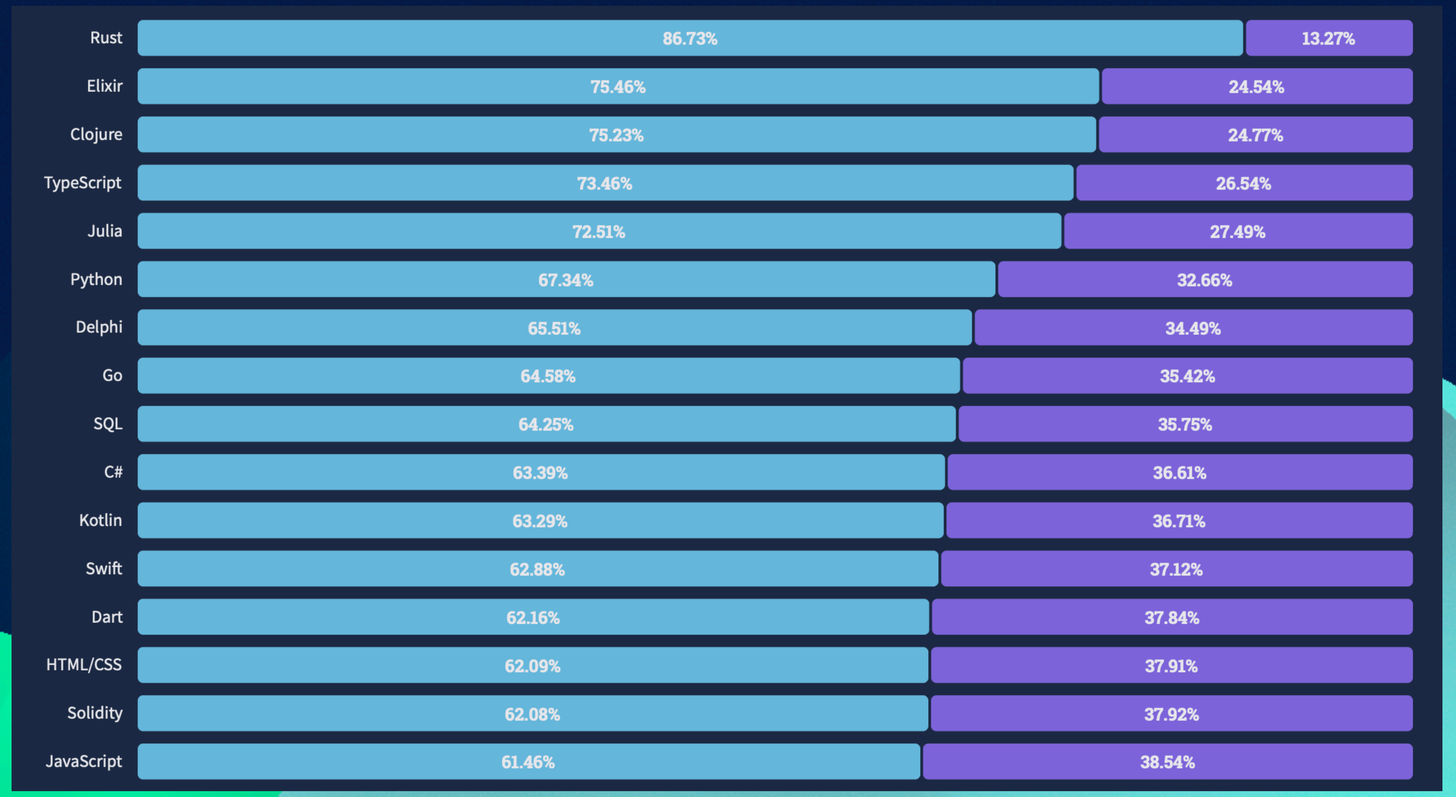
- Designed by Graydon Hoare at Mozilla Research in 2006
- First appeared in 2010; first stable release in 2014
- Used by:

#1

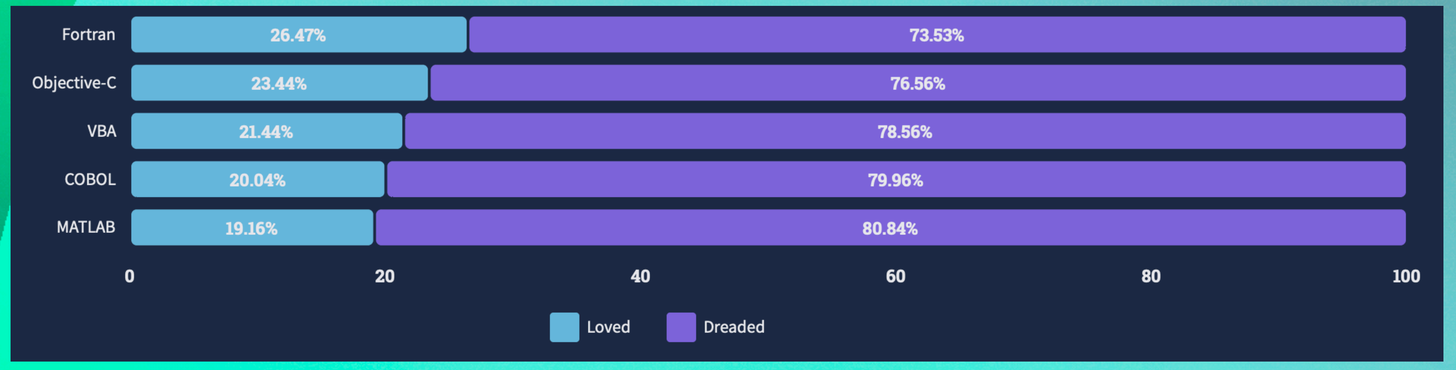
Most-loved language

2022 Stack Overflow Developer Survey

Loved



38 languages surveyed



Dreaded

Figure 3. Most-loved/dreaded programming languages, 2022 Stack Overflow Developer Survey (https://survey.stackoverflow.co/2022/#technology-most-loved-dreaded-and-wanted)

What Is Rust Used For?

- Systems programming
- Command-line interface (CLI) applications
- Scripting
- Embedded systems
- Web development
- WebAssembly

What Makes Rust Special?

- `Option` and `Result` enums
- Borrow checking
  - Memory safety without compromise
- Zero-cost abstractions
- Improved concurrency
- No garbage collector

Key Concepts

Built-in data types

Type	Description	Example
bool	Boolean value	`let my_bool: bool = true;`
i8, i16, i32, i64, i128	**Signed integers**	`let squared: i32 = -1;`
u8, u16, u32, u64, u128	**Unsigned integers**	`let fav_num: u32 = 7;`
usize, isize	Pointer-sized integers (size depends on platform)	`let fav_num: usize = 7;`
f32, f64	**Floating-point numbers**	`let total: f64 = 419.68;`
char	UTF-32 scalar value (4 bytes)	`let usv: char = '😂';`
&str	**String slice**	`let str: &str = "Hello, world!";`
(T1, T2)	Tuple	`let attribute: (&str, i32) = ("Age", 45);`

Key Concepts

Standard Library data types

Type	Description	Example
Box<T>	Pointer type for heap allocation	`let my_heap_pointer: Box<i32> = Box::new(5);`
String	**Dynamic, owned strings**	`let owned_str: String = String::from("Rust is great!");`
Vec<T>	**Dynamic arrays**	`let my_vec: Vec<[i32; 5]> = vec![[1, 2, 3, 4, 5]];`
Option<T>	`Option` enum	`let middle_name: Option<&str> = Some("John");` `let middle_name_2: Option<{unknown}> = None;`
Result<T, E>	`Result` enum	`Result::Ok(3);` `Result::Err("Something is wrong!");`

Key Concepts

Variables and mutability

- Define variables with the `let` keyword
- Variables are immutable by default (`mut` keyword to allow)
- Shadowing allows for simpler transformations, while retaining mutability afterward

Working with variables

```
fn main() {
    let x;
    x = 42;

    let y: i32 = 42;
}
```

Re-assigning a mutable variable

```
fn main() {
    let mut x = 5;
    println!("The value of x is: {x}");
    x = 6;
    println!("The value of x is: {x}");
}
```

Shadowing a variable

```
fn main() {
    let spaces = "   ";
    let spaces = spaces.len();
}
```

Key Concepts

Ownership, references, borrowing

- All programs must manage memory
- Rust does this by employing ownership rules for the compiler to enforce:
  1. Each value in Rust has an owner
  2. There can only be one owner at a time
  3. When the owner goes out of scope, the value will be dropped
- We can clone, obtain a reference to, or ‘move’ variables

Moved Variable Error

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

println!("{s1}");
```

Ownership

```
⊗ rust-demo % cargo build
   Compiling rust-demo v0.1.0 (/Users/brendan/Desktop/rust-demo)
warning: unused variable: `s2`
 --> src/main.rs:3:9
  |
3 |     let s2 = s1;
  |         ^^ help: if this is intentional, prefix it with an underscore: `_s2`
  |
  = note: `#[warn(unused_variables)]` on by default

error[E0382]: borrow of moved value: `s1`
 --> src/main.rs:5:16
  |
2 |     let s1 = String::from("hello");
  |         -- move occurs because `s1` has type `String`, which does not implement the `Copy` trait
3 |     let s2 = s1;
  |              -- value moved here
4 |
5 |     println!("{s1}");
  |                ^^ value borrowed here after move
  |
  = note: this error originates in the macro `$crate::format_args_nl` (in Nightly builds, run with -Z macro-backtrace for more info)

For more information about this error, try `rustc --explain E0382`.
warning: `rust-demo` (bin "rust-demo") generated 1 warning
error: could not compile `rust-demo` due to previous error; 1 warning emitted
```

Heap

Heap

Figure 3. `String` memory layout (https://doc.rust-lang.org/book/ch04-01-what-is-ownership.html)

Figure 4. `String` memory layout after ‘move’ (https://doc.rust-lang.org/book/ch04-01-what-is-ownership.html)

Key Concepts

Functions

- The `main()` function
- Asynchronous
- Comprised of statements and an optional ending expression
- Iterators

Iterators

```rust
fn main() {
    let x = vec![1, 3, 3, 7]
        .iter()
        .map(|x| x + 8)
        .fold(0, |x, y| x + y);
}
```

Asynchronous function

```rust
async fn fetch_data() {
    fetch_data().await;
    process_data();
    save_data().await;
}
```

Rust `main()` function

```rust
fn main() {
    println!("Hello, world!");
    let result = triple(140.23);
    println!("{}", result);
}

fn triple(x: f32) -> f32 {
    x * 3.0
}
```

Key Concepts

Enums

- Declare sets of related values
- Optionally contain data
- Can store almost any data!
- Can attach functionality!

```
fn main() {
    #[derive(Debug)]
    enum IpAddrKind {
        V4,
fn main() {
    #[derive(Debug)]
    enum IpAddr {
        V4(String),
        V6(String),
    }

    fn route(_ip_kind: IpAddr) {}

    route(_ip_kind: IpAddr::V4("127.0.0.1".to_string()));

    let home: IpAddr = IpAddr::V4("127.0.0.1".to_string());

    println!(
        "{:?}",
        match home {
            IpAddr::V4(address) => address,
            IpAddr::V6(address) => address,
        }
    );
}
        address: String::from("127.0.0.1"),
    };

    println!("{:?} {:?}", home.kind, home.address)
} fn main
```

Key Concepts

Option<T>

- Rust doesn’t use null pointers to represent a lack of data
- `Option<T>` enum represents an optional value
- Use cases:
  - Initial values
  - Optional function arguments
  - Optional struct fields
  - Nullable pointers

Option<T>

```
pub enum Option<T> {
    None,
    Some(T),
}
```

Option<T>

```
fn main() {
    let primary_language = "English";
    let secondary_language = Some("Mandarin");
    let tertiary_language: Option<&str> = None;
}
```

Key Concepts

Result<T, E>

- `Result<T, E>` enum is used for returning and propagating errors
  - `Ok(T)` - Represents success, and contains a value of type `T`
  - `Err(E)` - Represents error, and contains an error value of type `E`

- Used whenever errors are expected and recoverable

Result<T, E>

```
enum Result<T, E> {
    Ok(T),
    Err(E),
}
```

Result<T, E> Usage

```
use std::{fs::File, io};

fn main() {
    let file: Result<File, io::Error> = File::open("memo.txt");

    let file = match file {
        Ok(file) => file,
        Err(error) => panic!("Problem opening the file: {:?}", error),
    };
}
```

Key Concepts

Pattern matching

```
match EXPRESSION {
    PATTERN => EXPRESSION | CODE BLOCK,
    PATTERN => EXPRESSION | CODE BLOCK,
    _ => println!(“My catch-all.”),
}
```

- Rust doesn’t use null pointers
- `match` expressions use patterns
- Exhaustive on the value being matched
- `if-let` expressions allow flexibility

`if-let` expressions

```rust
if let 4 = rng.gen_range(2..5) {
    println!("It was 4");
} else {
    // Optional
}
```

Pattern matching a random number

```rust
use rand::Rng;

fn main() {
    let mut rng = rand::thread_rng();
    let rand_num = rng.gen_range(0..10);
    match rand_num {
        0..=5 => println!("Less than 5."),
        5 | 6 => println!("5 or 6."),
        _ => println!("Between 7 and 9."),
    }
}
```

Key Concepts

Structs

- Hold related, *named* values/data (can be of different types)
- The struct itself, and any of its fields, can be private (default)
- Access fields with dot notation
- Functionality assigned via `impl` blocks

Structs

```
fn main() {
    struct User {
        active: bool,
        username: String,
        email: String,
        age: u32,
    }

    let mut hamid = User {
        active: true,
        username: "hamid_d".to_string(),
        email: String::from("hamid_d@example.com"),
        age: 45,
    };

    hamid.age = 46;
}
```

Cool Stuff

Energy, speed, and memory

Figure 5. Programming language performance comparisons (https://greenlab.di.uminho.pt/wp-content/uploads/2017/09/paperSLE.pdf)

Table 1. CLBG corpus of programs.

Benchmark	Description	Input
n-body	Double precision N-body simulation	50M
fannkuch-redux	Indexed access to tiny integer sequence	12
spectral-norm	Eigenvalue using the power method	5,500
mandelbrot	Generate Mandelbrot set portable bitmap file	16,000
pidigits	Streaming arbitrary precision arithmetic	10,000
regex-redux	Match DNA 8mers and substitute magic patterns	fasta output
fasta	Generate and write random DNA sequences	25M
k-nucleotide	Hashtable update and k-nucleotide strings	fasta output
reverse-complement	Read DNA sequences, write their reverse-complement	fasta output
binary-trees	Allocate, traverse and deallocate many binary trees	21
chameneos-redux	Symmetrical thread rendezvous requests	6M
meteor-contest	Search for solutions to shape packing puzzle	2,098
thread-ring	Switch from thread to thread passing one token	50M

Table 4. Normalized global results for Energy, Time, and Memory

Total

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

	Time
(c) C	1.00
(c) Rust	1.04
(c) C++	1.56
(c) Ada	1.85
(v) Java	1.89
(c) Chapel	2.14
(c) Go	2.83
(c) Pascal	3.02
(c) Ocaml	3.09
(v) C#	3.14
(v) Lisp	3.40
(c) Haskell	3.55
(c) Swift	4.20
(c) Fortran	4.20
(v) F#	6.30
(i) JavaScript	6.52
(i) Dart	6.67
(v) Racket	11.27
(i) Hack	26.99
(i) PHP	27.64
(v) Erlang	36.71
(i) Jruby	43.44
(i) TypeScript	46.20
(i) Ruby	59.34
(i) Perl	65.79
(i) Python	71.90
(i) Lua	82.91

	Mb
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Hack	3.34
(v) Racket	3.52
(i) Ruby	3.97
(c) Chapel	4.00
(v) F#	4.25
(i) JavaScript	4.59
(i) TypeScript	4.69
(v) Java	6.01
(i) Perl	6.62
(i) Lua	6.72
(v) Erlang	7.20
(i) Dart	8.64
(i) Jruby	19.84

Contrasting With TypeScript/JavaScript

	Rust	TypeScript
Adding dependencies	Add to `cargo.toml` file	Install with a package manager
Typing	Static	Static
`let`	Immutable by default	“Mutable” variable
Compiled into machine code?	Compiled	Interpreted
Learning curve	Steep	Not as steep
Garbage collected?	No	Yes

What We’ve Learned

- Rust emphasizes performance, type/memory safety, and concurrency
- Best of both worlds: high-level features with low-level control
- Used in systems programming, scripting, embedded systems, and Web Dev
- Data types, ownership, variables, functions, enums, pattern matching, structs
- Differences from common technology like TypeScript

Further Learning

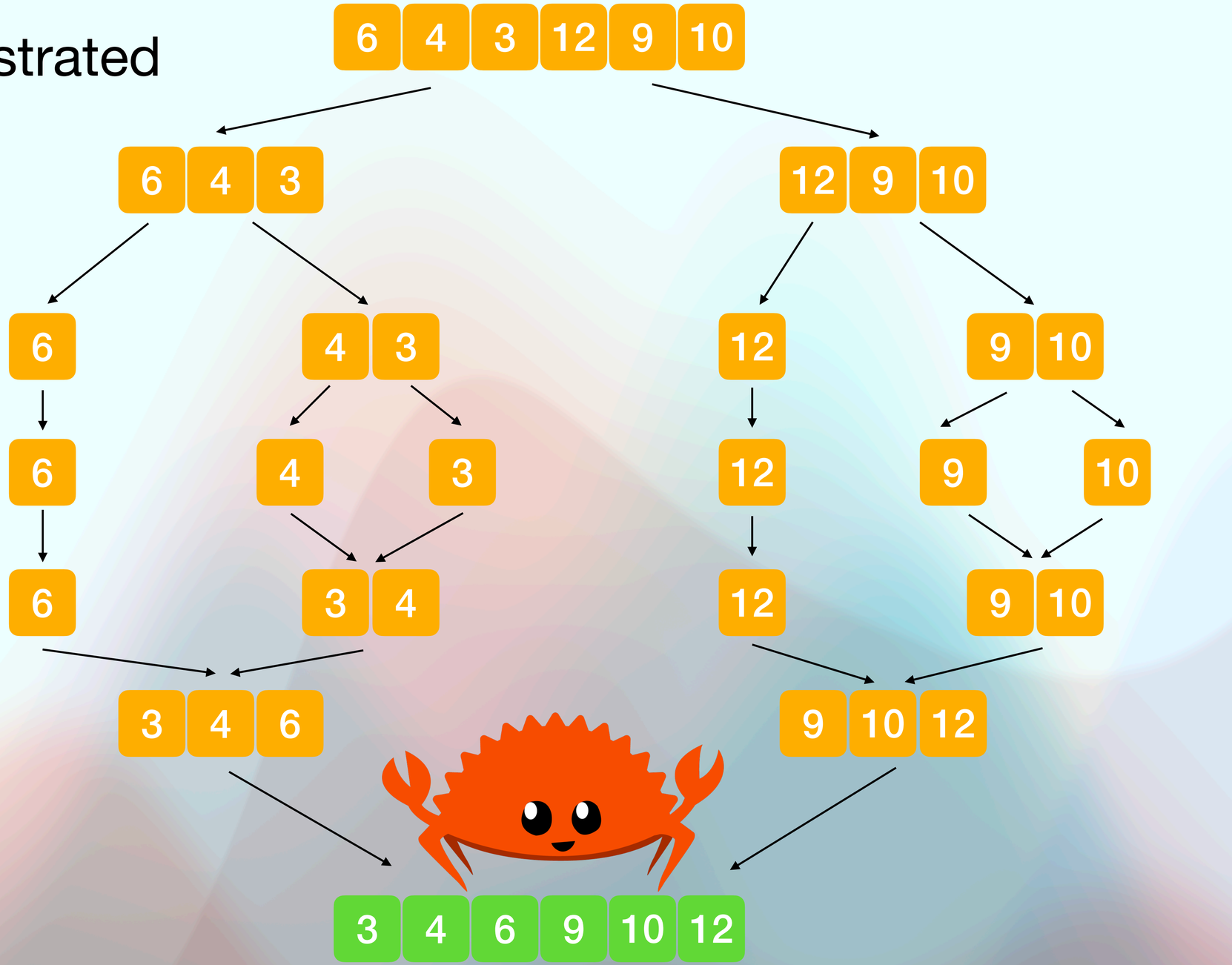
Things to look into…

- Traits
- Lifetimes
- Error-handling
- Testing
- Concurrent Rust
- Unsafe Rust
- Macros

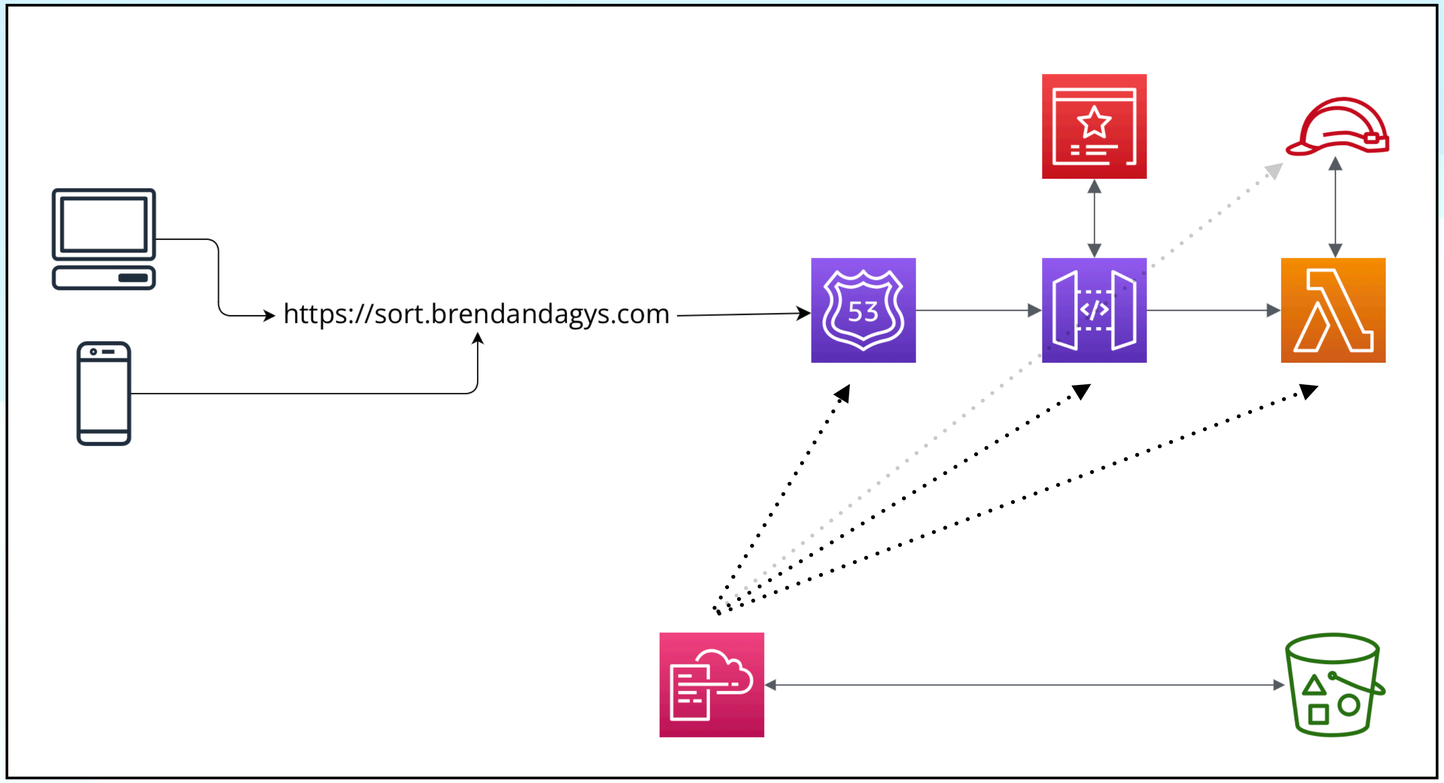
Demo

- We are going to implement merge-sort in Rust!
  - Divide-and-conquer algorithm
  - Recursive

Merge-Sort, Illustrated



AWS Deployment Architecture



Helpful Resources

Learn more…

- **The ‘Book’**
  - Between an article and the full documentation, the ‘book’ is a superb introduction to the key parts of the language you need to know to get started.
  - https://doc.rust-lang.org/book/
- **Wikipedia article**
  - https://en.wikipedia.org/wiki/Rust_(programming_language)
- **Reddit (Rust subreddit)**
  - Great questions, content, and updates on the growth of the Rust language.
  - https://www.reddit.com/r/rust/
- **YouTube (Let’s Get Rusty)**
  - A great channel that covers beginner to advanced concepts, in byte-sized videos. The creator, Bogdan, comes from a Web development background and fills in the gaps for those coming in with such experience.
  - https://www.youtube.com/c/LetsGetRusty/videos

Thank you!

Any questions?