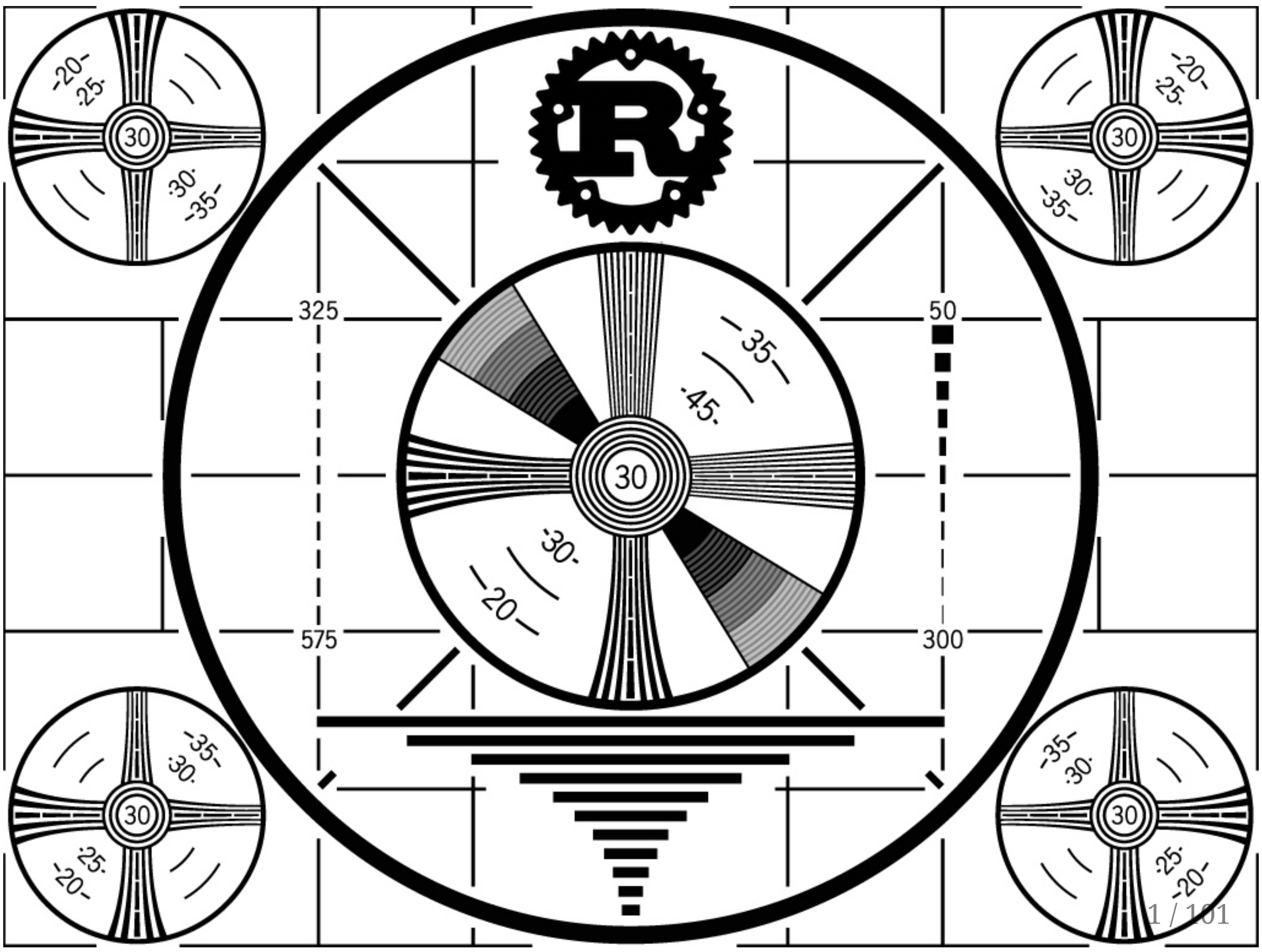

-20-
-25-
30
-30-
-35-
325
575
50
300
-45-
-35-
-20-

Introduction to Rust

Goal

Have a good chance of being able to read and understand some Rust code.

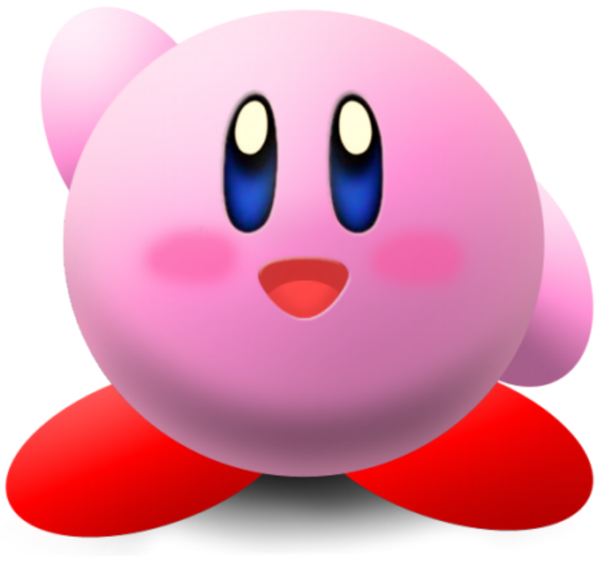
Agenda

1. What's not that unique about Rust
2. What makes Rust somewhat unique
3. What makes Rust really unique
4. Beyond the language
5. Beyond the code
6. Who is using Rust

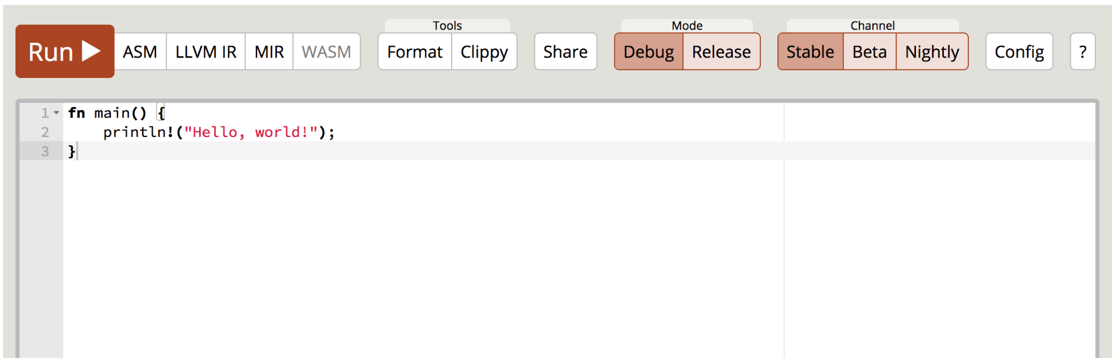
Who am I?

integer 32

Stack Overflow



Rust Playground



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

play.rust-lang.org

Jake Goulding

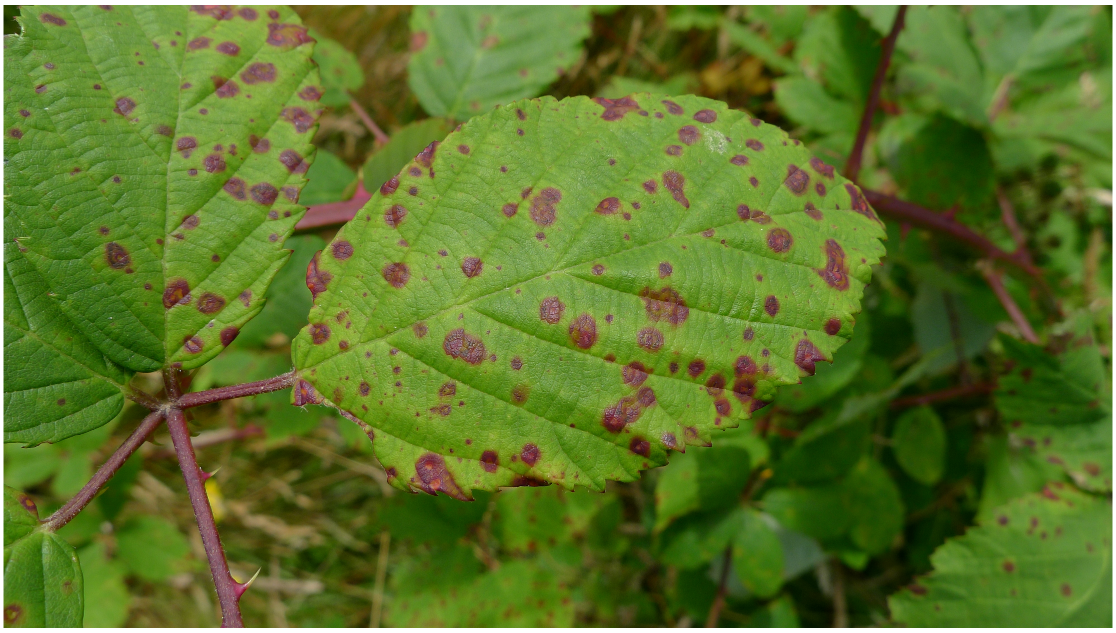
- Rust infrastructure team
- Working on a Rust video course for Manning
- A handful of crates
- Help out with AVR-Rust

Who are you?

What is Rust?

Metal thing

Fungus

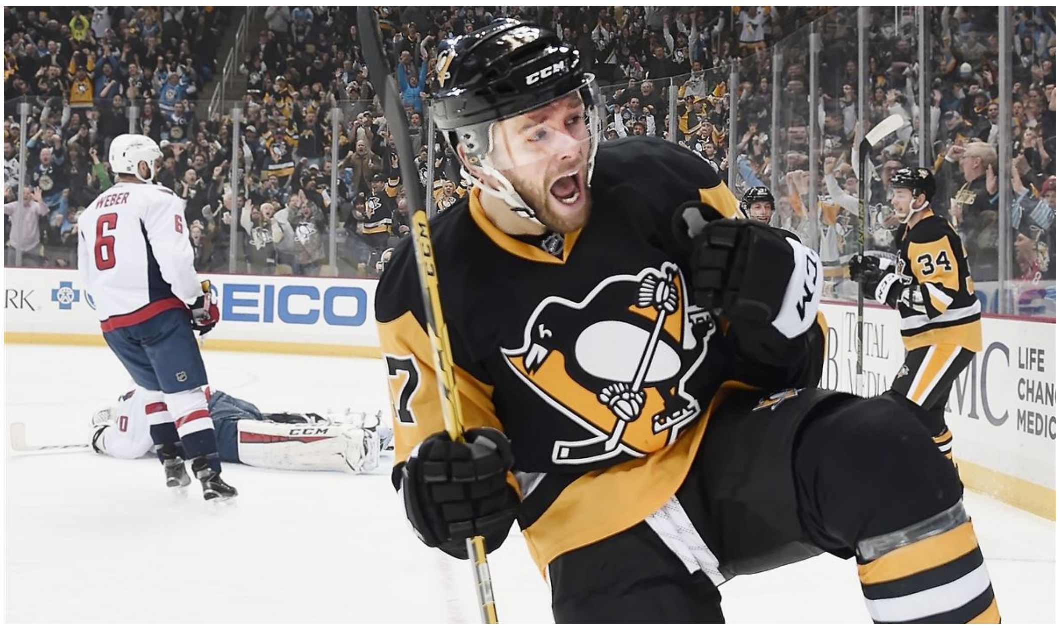


John Tann

Video Game

RUST

Back-to-back Stanley Cup winner



Peter Diana/Post-Gazette

What is Rust?

> Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety.

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

What's not that unique about Rust?

Variables

```
let age = 21;
let is_enabled = true;
let emoji = '🌍';
```

Built-in primitive types

- Signed integers: `i8`, `i16`, `i32`, `i64`, `i128`
- Unsigned integers: `u8`, `u16`, `u32`, `u64`, `u128`
- Floating point numbers: `f32`, `f64`
- Booleans: `bool`
- Unicode scalar values: `char`
- Fixed-sized collection of one type: `[T; N]` ("array")
- Fixed-size collection of arbitrary types: `(A, B)` ("tuple")

Statically, strongly typed

```
let a: u8 = 42;
let b: bool = a;
```

```
error[E0308]: mismatched types
  |
3 | let b: bool = a;
  |               ^ expected bool, found u8
  |
```

Control flow: `if`

```
if 1 < 2 {
    42.0
} else if 3 > 4 {
    -32.4
} else {
    99.0
}
```

Control flow: `while`

```rust
while 2 > 3 {
    println!("never printed");
}
```

Control flow: `for`

```rust
for i in 0..3 {
    println!("{}", i);
}
```

Control flow: `loop`

```
loop {
    println!("I go forever!");
}
```

Functions

```rust
fn is_teenager(age: u8) -> bool {
    age >= 13 && age <= 19
}

fn greeter(name: String, age: u8) {
    let next_age = age + 1;

    println!(
        "Hello, {}! Next year you will be {}.",
        name, next_age
    );

    if is_teenager(next_age) {
        println!("A teenager!");
    }
}
```

Structs

```rust
struct DungeonMonster {
    name: String,
    health: u32,
    damage_per_attack: f32,
}
```

Inherent methods

```rust
impl DungeonMonster {
    fn new() -> DungeonMonster {
        DungeonMonster {
            name: String::from("Grendel"),
            health: 42,
            damage_per_attack: 99.0,
        }
    }

    fn print_status(&self) {
        println!(
            "{} the monster has {} health",
            self.name, self.health
        )
    }
}
```

Calling methods

```rust
fn main() {
    let monster = DungeonMonster::new();
    monster.print_status();
}
```

Traits

```rust
trait Monster {
    fn damage(&self) -> f32;

    fn roar(&self) {
        println!("I roar!");
    }
}

impl Monster for DungeonMonster {
    fn damage(&self) -> f32 {
        self.damage_per_attack
    }
}

fn main() {
    let monster = DungeonMonster::new();
    monster.damage();
    monster.roar();
}
```

Generics

```rust
fn be_noisy<M>(monster: M)
where
    M: Monster,
{
    monster.roar();
}

fn main() {
    let monster = DungeonMonster::new();
    be_noisy(monster)
}
```

Trait objects

```
fn be_noisy(monster: Box<Monster>) {
    monster.roar();
}

fn main() {
    let monster = DungeonMonster::new();
    be_noisy(Box::new(monster))
}
```

Questions?

What makes Rust somewhat unique?

Variables are immutable by default

```rust
let a = 1;
a += 1;
```

```
error[E0384]: cannot assign twice to immutable variable `a`
  |
2 | let a = 1;
  |     - first assignment to `a`
3 | a += 1;
  | ^^^^^^ cannot assign twice to immutable variable
```

Variables are immutable by default

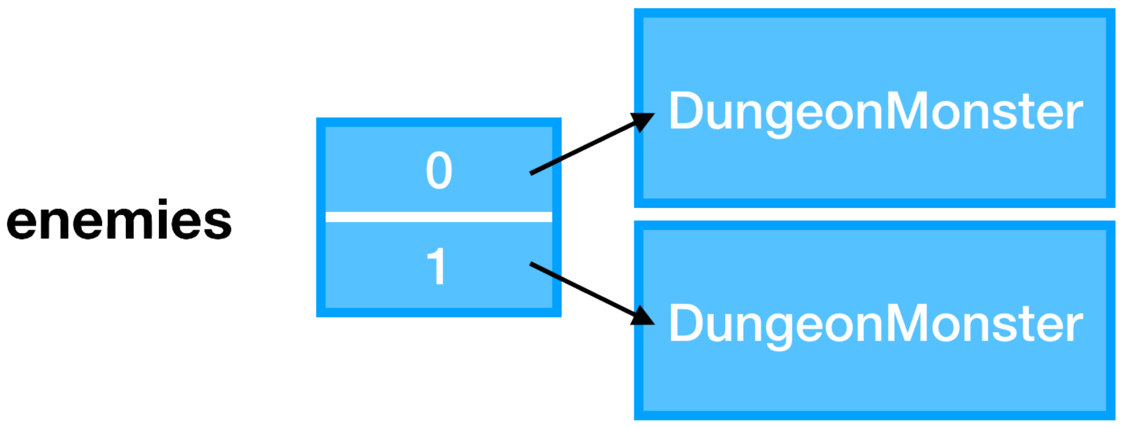
```
let mut a = 1;
a += 1;
```

Values not automatically placed on the heap

```
let enemies = [DungeonMonster::new(), DungeonMonster::new()];
```

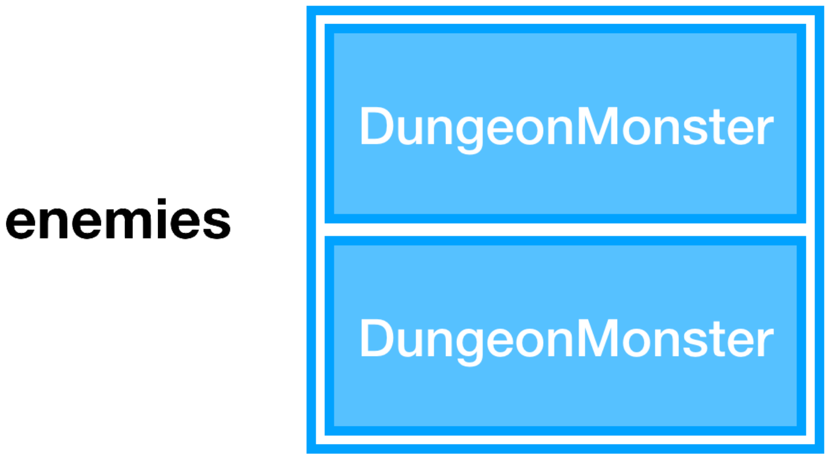
Values not automatically placed on the heap

```
let enemies = [DungeonMonster::new(), DungeonMonster::new()];
```



Values not automatically placed on the heap

```
let enemies = [DungeonMonster::new(), DungeonMonster::new()];
```



No garbage collector

```rust
fn do_tough_work() {
    // Allocate a vector of numbers
    let powers = vec![1, 2, 4, 8];
    // Memory is freed when variable goes out of scope
}
```

No NULL

There is no implicit `NULL`, `null`, `nil`, `None`, `undefined`, etc.

> 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.

Tony Hoare, 2009

Enums

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

Pattern matching

```rust
let name = Some("Vivian");
match name {
    Some(n) => println!("Hello, {}", n),
    None    => println!("Who are you?"),
}
```

Pattern matching

```
let name = Some("Vivian");
match name {
    Some(n) => println!("Hello, {}", n),
    None    => println!("Who are you?"),
}
```

```
let name = None;
if let Some(n) = name {
    println!("Hello, {}", n);
}
```

Error handling

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

Error handling

```rust
fn can_fail() -> Result<i32, String> { /* ... */ }

fn maybe_increment_1() -> Result<i32, String> {
    let val = can_fail()?;
    Ok(val + 1)
}

fn maybe_increment_2() -> Result<i32, String> {
    can_fail().map(|val| val + 1)
}

fn main() {
    match maybe_increment_1() {
        Ok(val) => println!("It worked: {}", val),
        Err(e) => println!("Something went wrong: {}", e),
    }
}
```

Error handling

```rust
fn can_fail() -> Result<i32, String> { /* ... */ }

fn main() {
    can_fail();
}
```

```
warning: unused `std::result::Result` which must be used
  |
4 |     can_fail();
  |     ^^^^^^^^^^^
  |
```

Iterators

```
pub trait Iterator {
    type Item;
    fn next(&mut self) -> Option<Self::Item>;

    // Many useful methods provided
}
```

Iterators

```
let iter_a = 0..3;
let iter_b = [-10, 0, 42].iter();
let iter_c = "hello, world".chars();

iter_a.max();
iter_b.min();

for c in iter_c {
    println!("{}", c);
}
```

Closures

```
fn sum_of_squares(start: i32, end: i32) -> i32 {
    (start..end)
        .map(|x| x * x)
        .sum()
}
```

Macros

```
macro_rules! add_3 {
    ($a:expr, $b:expr, $c:expr) => {
        ($a + $b + $c)
    }
}

fn main() {
    println!("{}", add_3!(1, 2, 3));
}
```

Questions?

What makes Rust really unique?

Ownership

```rust
fn do_tough_work() {
    let mut powers = vec![1, 2, 4, 8];
}
```

Borrowing

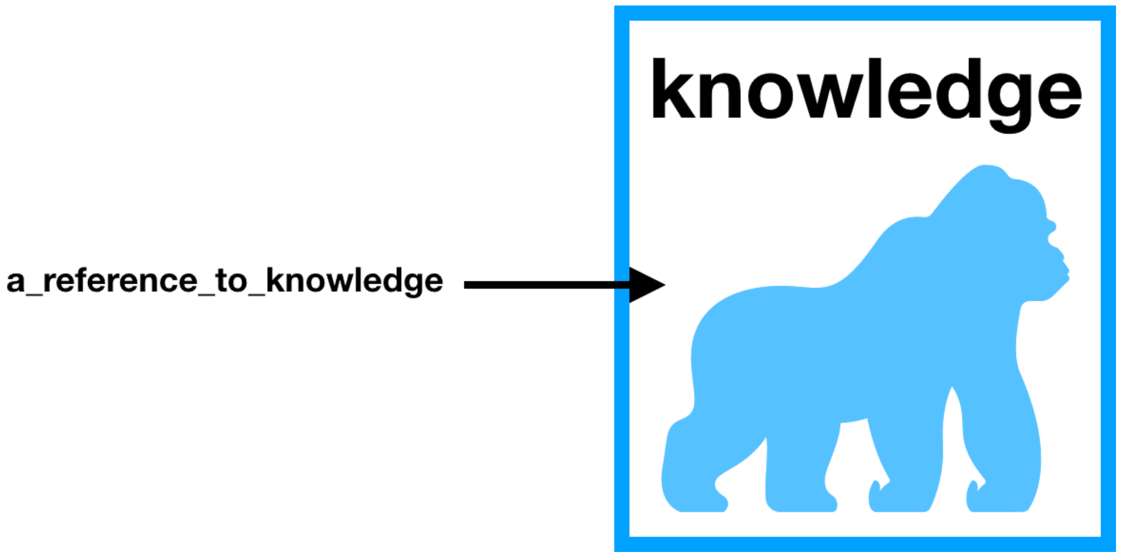
```
let knowledge = Wikipedia::download();
let a_reference_to_knowledge = &knowledge;
```

Ownership and borrowing

```
let knowledge = Wikipedia::download();
let a_reference_to_knowledge = &knowledge;
```

Ownership and borrowing

```
let knowledge = Wikipedia::download();
let a_reference_to_knowledge = &knowledge;
```



Immutable and mutable borrowing

```
let a_book = String::new();
let reader = &a_book;
```

```
let mut a_book = String::new();
let author = &mut a_book;
```

Immutable and mutable borrowing

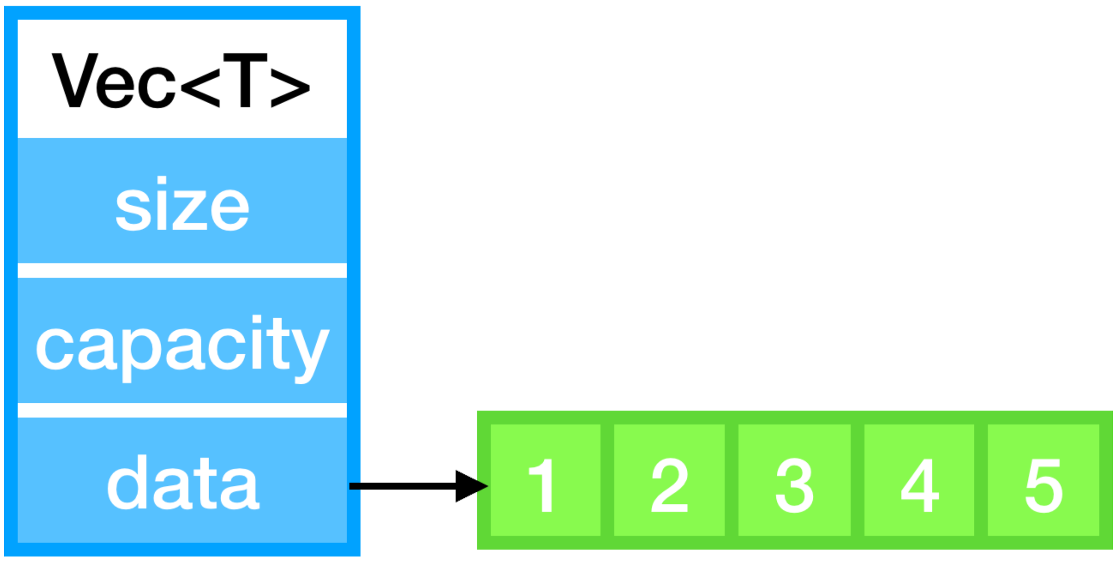
```rust
let mut a_book = String::new();
let author = &mut a_book;
let reader = &a_book;
```

```
error[E0502]: cannot borrow `a_book` as immutable because it
              is also borrowed as mutable
  |
3 | let author = &mut a_book;
  |                   ------ mutable borrow occurs here
4 | let reader = &a_book;
  |               ^^^^^^ immutable borrow occurs here
5 | }
  | - mutable borrow ends here
```

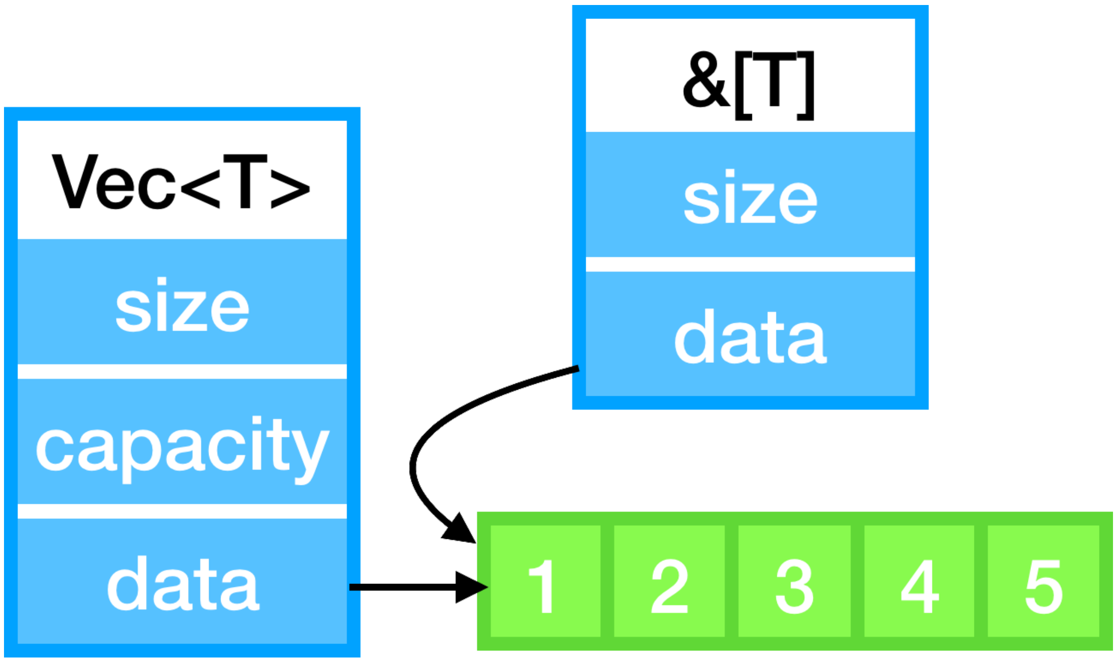
Slices

```
let scores = vec![1, 2, 3];
let some_scores = &scores[1..];
```

Slices



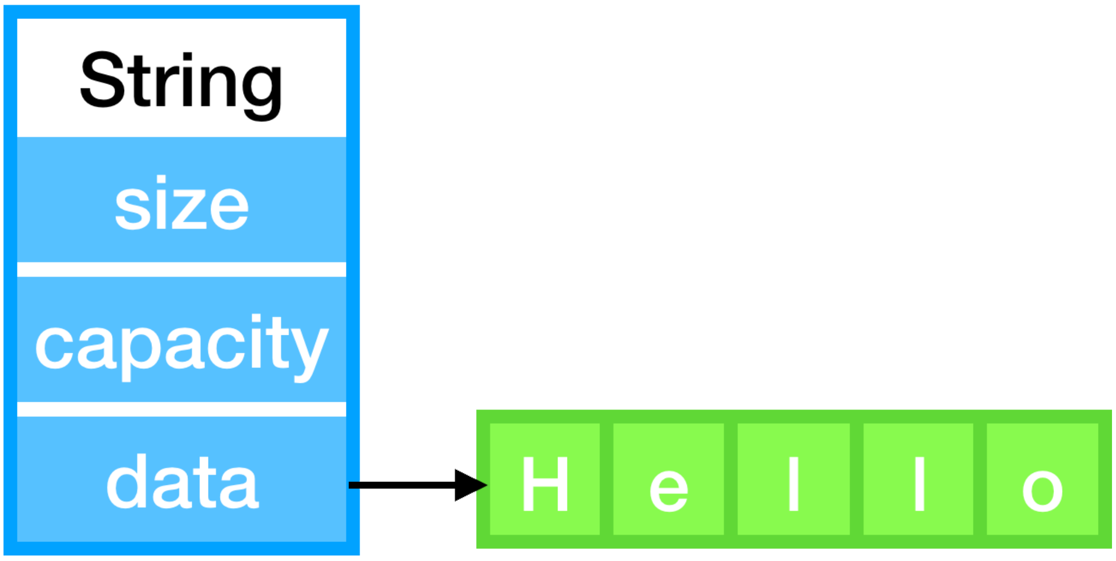
Slices



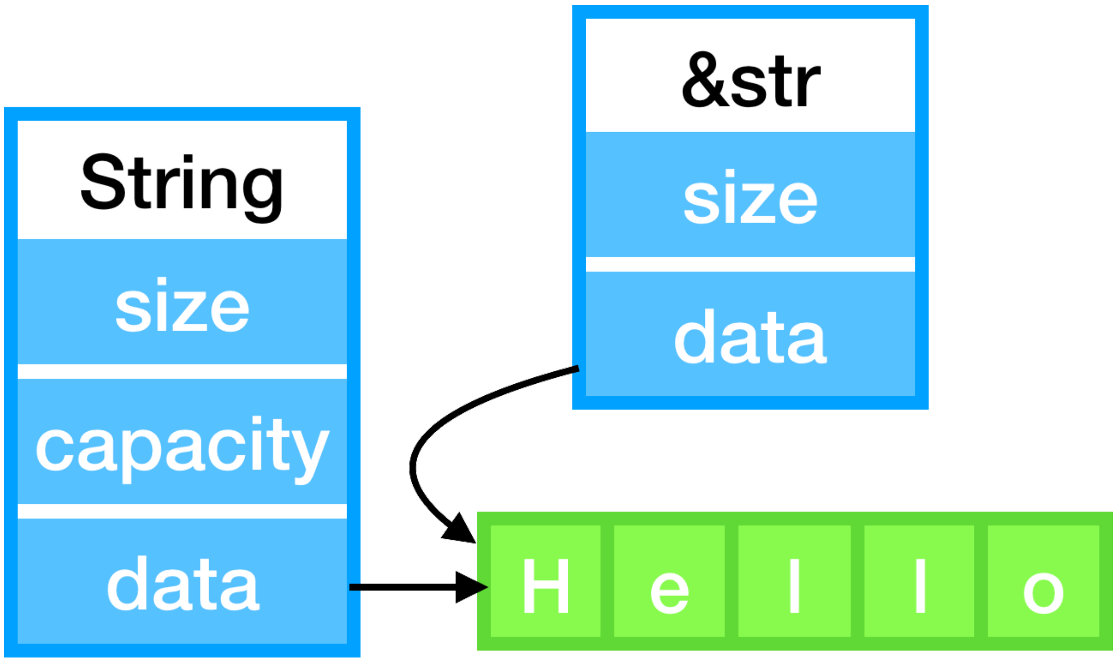
String slices

```
let novel = "hello, world!;
let chapter_1 = &novel[..5];
```

String slices



String slices



The borrow checker

```rust
let a_reference_to_the_book = {
    let a_book = String::new();
    &a_book
};
```

```
error[E0597]: `a_book` does not live long enough
  |
4 |     &a_book
  |      ^^^^^^ borrowed value does not live long enough
5 | };
  | - `a_book` dropped here while still borrowed
6 |
7 | }
  | - borrowed value needs to live until here
```

Move semantics

```rust
let a_book = String::new();
let ref_to_the_book = &a_book;
let a_moved_book = a_book;
```

```
error[E0505]: cannot move out of `a_book` because
              it is borrowed
  |
3 | let ref_to_the_book = &a_book;
  |                       ------ borrow of `a_book`
  |                              occurs here
4 | let a_moved_book = a_book;
  |     ^^^^^^^^^^^^ move out of `a_book` occurs here
```

Lifetimes

```rust
fn a_chapter_of_the_book<'a>(book: &'a str) -> &'a str {
    &book[100..]
}

let an_entire_book = String::new();
let a_chapter = a_chapter_of_the_book(&an_entire_book);
```

Beyond the language

Standard library

Collections

- Sequences: `Vec`, `VecDeque`, `LinkedList`
- Maps: `HashMap`, `BTreeMap`
- Sets: `HashSet`, `BTreeSet`
- Misc: `BinaryHeap`

Standard library

Building blocks

- `Box`
- `String`
- `Option<T>`
- `Result<T, E>`
- `Iterator`

Standard library

Platform abstractions and I/O

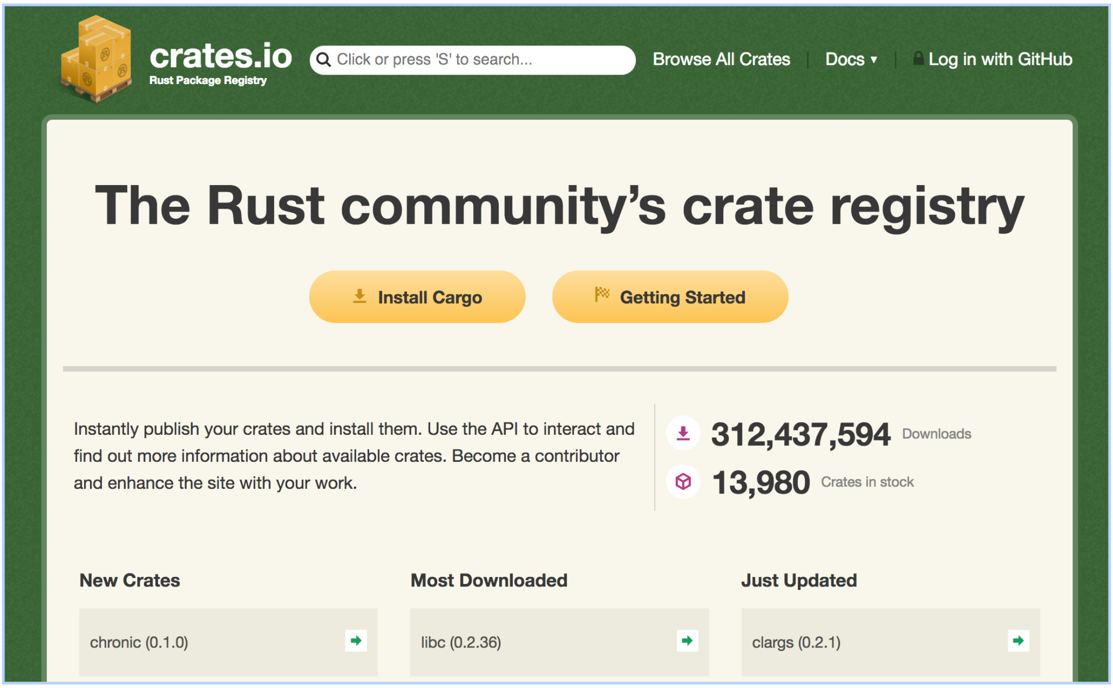
- Files
- TCP
- UDP
- Threading
- Shared memory primitives
- Subprocesses
- Environment

Cargo

A build and dependency managment tool in one.

- **new**
- **build**
- **run**
- **test**
- **doc**
- **publish**

Crates



The Rust community's crate registry

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.

312,437,594 Downloads
13,980 Crates in stock

New Crates	Most Downloaded	Just Updated
chronic (0.1.0)	libc (0.2.36)	clargs (0.2.1)

Testing

Built-in basic testing framework and assertions

```rust
#[test]
fn addition_works() {
    assert_eq!(2, 1 + 1);
}
```

Fuzzing

- Provide invalid, unexpected, or random data as input
- Two main implementations
    - cargo-fuzz
    - afl.rs

```
fuzz_target!(|data: &[u8]| {
    if let Ok(s) = std::str::from_utf8(data) {
        let _ = fuzzy_pickles::parse_rust_file(s);
    }
});
```

Property-based testing

- Generates data structures based on properties
- Shrinking reduces found test cases to manageable size

Property-based testing: QuickCheck

```
fn reverse<T: Clone>(xs: &[T]) -> Vec<T> { /* ... */ }
```

```
quickcheck! {
    fn quickcheck_example(xs: Vec<u32>) -> bool {
        xs == reverse(&reverse(&xs))
    }
}
```

Property-based testing: Proptest

```
fn reverse<T: Clone>(xs: &[T]) -> Vec<T> { /* ... */ }
```

```
use proptest::collection::vec;
use proptest::num::u32;

proptest! {
    #[test]
    fn proptest_example(ref xs in vec(u32::ANY, 0..100)) {
        xs == &reverse(&reverse(xs))
    }
}
```

Documentation

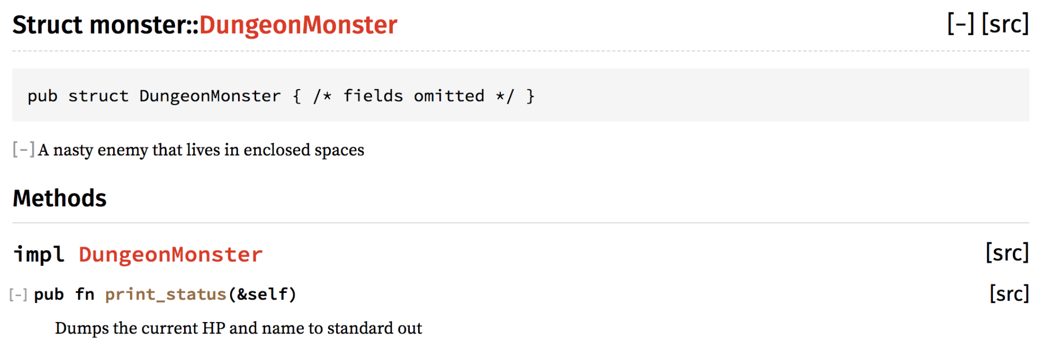
```rust
/// A nasty enemy that lives in enclosed spaces
struct DungeonMonster { /* ... */ }

impl DungeonMonster {
    /// Dumps the current HP and name to standard out
    fn print_status(&self) { /* ... */ }
}
```

Documentation

```
/// A nasty enemy that lives in enclosed spaces
struct DungeonMonster { /* ... */ }

impl DungeonMonster {
    /// Dumps the current HP and name to standard out
    fn print_status(&self) { /* ... */ }
}
```



Struct monster::DungeonMonster [-] [src]

```
pub struct DungeonMonster { /* fields omitted */ }
```

[-] A nasty enemy that lives in enclosed spaces

Methods

`impl DungeonMonster` [src]

[-] `pub fn print_status(&self)` [src]

Dumps the current HP and name to standard out

Developer Tools: rustfmt

```
fn main( )
{
  std::iter::repeat(1).take(4).map(|x| x * x).map(|x| x + 4).
    println! ( "Hello, world!" );

}
```

Developer Tools: rustfmt

```rust
fn main( )
{
  std::iter::repeat(1).take(4).map(|x| x * x).map(|x| x + 4).
    println! ( "Hello, world!" );

}
```

```rust
fn main() {
    std::iter::repeat(1)
        .take(4)
        .map(|x| x * x)
        .map(|x| x + 4)
        .fold(0, |a, e| a + e);
    println!("Hello, world!");
}
```

Developer Tools: clippy

```
fn switcheroo(scores: &mut[i32], a: usize, b: usize) {
    let tmp = scores[a];
    scores[a] = scores[b];
    scores[b] = tmp;
}
```

Developer Tools: clippy

```rust
fn switcheroo(scores: &mut[i32], a: usize, b: usize) {
    let tmp = scores[a];
    scores[a] = scores[b];
    scores[b] = tmp;
}
```

```
warning: this looks like you are swapping elements of
         `scores` manually
  |
2 | /     let tmp = scores[a];
3 | |     scores[a] = scores[b];
4 | |     scores[b] = tmp;
  | |___________________^ help: try: `scores.swap(a, b)`
  |
  = note: #[warn(manual_swap)] on by default
```

Beyond the code

Community

- IRC
- Forums
- Reddit
- Stack Overflow
- Meetups
- Conferences

Code of Conduct

- Want the community to be welcoming to all
- Present since day one
- Enforced by moderators

Requests For Comments (RFCs)

Community-driven process for "substantial" changes to Rust

- Summary
- Motivation
- Guide-level explanation
- Reference-level explanation
- Drawbacks
- Rationale and alternatives
- Unresolved questions

Evolution of error handling

```
fn can_fail() -> Result<i32, String> { /* ... */ }

fn maybe_increment() -> Result<i32, String> {
    let val = match can_fail() {
        Ok(v) => v,
        Err(e) => return Err(e),
    };
    Ok(val + 1)
}
```

Evolution of error handling

```
fn can_fail() -> Result<i32, String> { /* ... */ }

fn maybe_increment() -> Result<i32, String> {
    let val = try!(can_fail());
    Ok(val + 1)
}
```

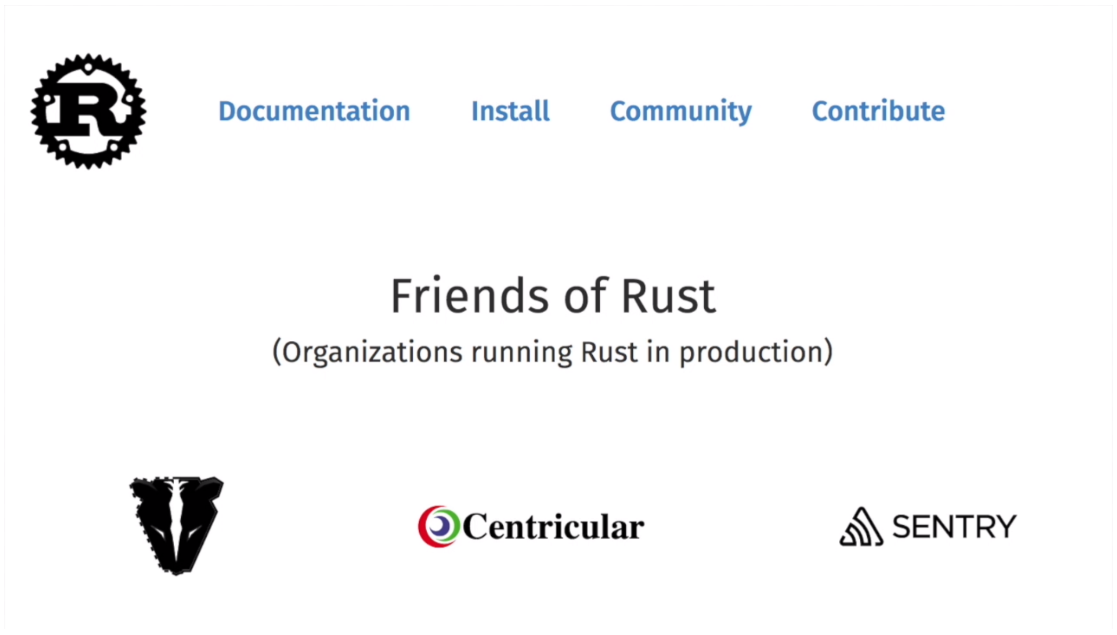
Evolution of error handling

```
fn can_fail() -> Result<i32, String> { /* ... */ }

fn maybe_increment() -> Result<i32, String> {
    let val = can_fail()?;
    Ok(val + 1)
}
```

Who is using Rust?

Lots of people



Friends of Rust

(Organizations running Rust in production)

Who is using Rust?

Who is using Rust?

CHUCKLEFISH

What's wrong with Rust?

- Slow compile times
- Doesn't support every platform C does
- Steeper learning curve
- IDE support still rudimentary
- Not a library for everything yet
- Not as many jobs as other languages

What doesn't Rust prevent?

- Deadlocks
- Non-data race conditions
- Leaking memory
- Failing to call destructors
- Crashing the program
- **Logic bugs**

Where do I go next?

- https://doc.rust-lang.org/
- IRC
- Session on how Rust helps with security
- Rust workshop

What didn't we talk about?

- Concurrency
  - Threading
  - Asynchronous
- C / FFI
- Embedded
- WebAssembly
- Nightly features
- Unsafe Rust

@jakegoulding

integer 32

integer32.com