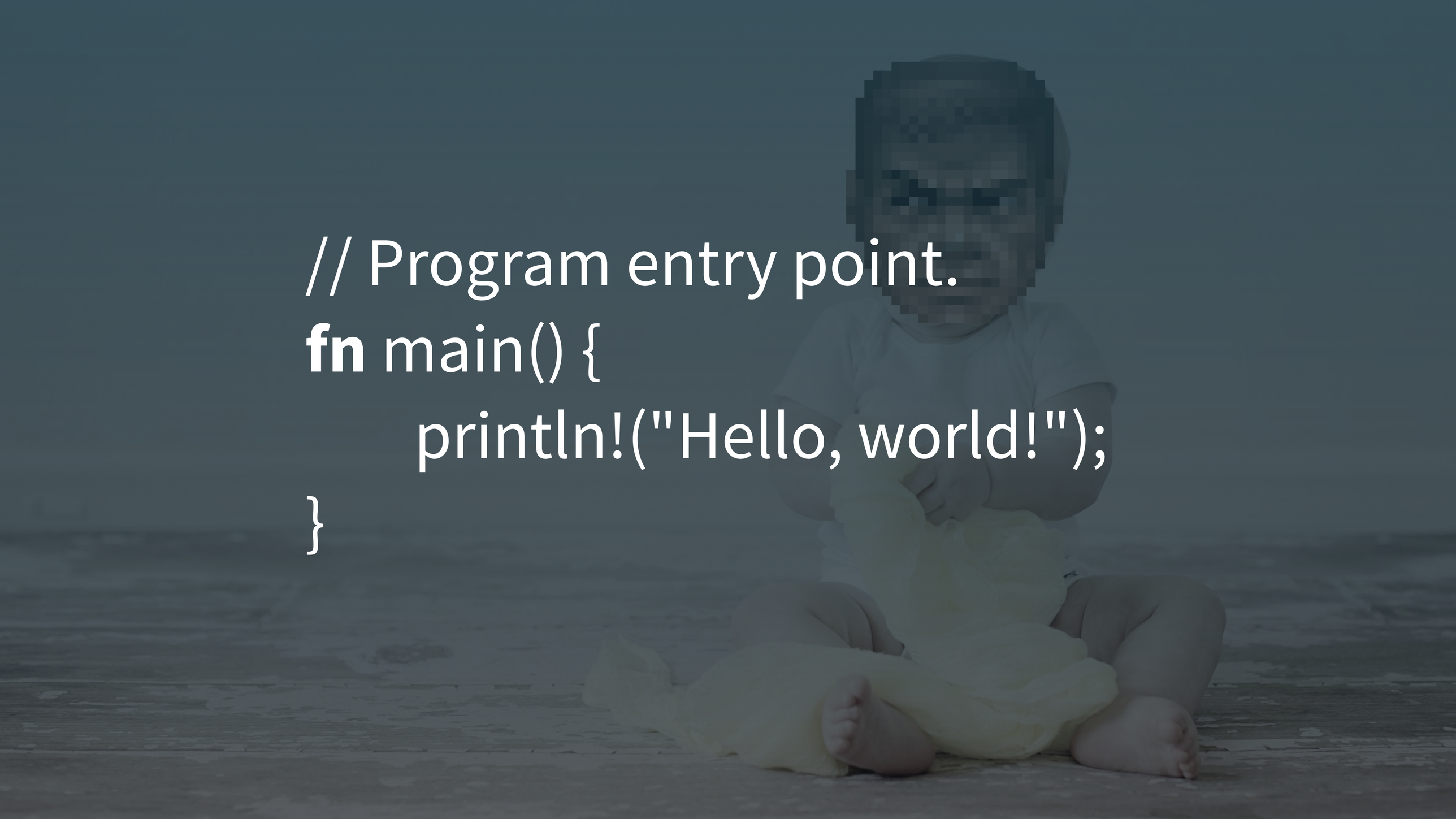
The Rust Language

Hello world



```
// Program entry point.
fn main() {
    println!("Hello, world!");
}
```

https://play.rust-lang.org

https://rustup.rs

https://github.com/ilammy/rust-talk

```
$ cd 1/hello-world-1
$ cargo run
Compiling hello-world-1 v0.1.0 (file:///Users/ilammy/Documents/dev/rust-talk/1/hello-world-1)
  Finished dev [unoptimized + debuginfo] target(s) in 0.51s
  Running `/Users/ilammy/Documents/dev/rust-talk/target/debug/hello-world-1`
Hello, world!
```

Hello world II

```
fn fib(n: u32) -> u32 {
    let mut a = 1;
    let mut b = 1;
    for _ in 1..n {
        let t = a;
        a += b;
        b = t;
    }
    return a;
}
```

```rust
#[test]
fn fib_values() {
    let expected = vec![1, 1, 2, 3, 5, 8, 13];

    let actual = (0..7)
        .map(fib)
        .collect::<Vec<_>>();

    assert_eq!(actual, expected);
}
```

```rust
#[test]
fn fib_values() {
    let expected = vec![1, 1, 2, 3, 5, 8, 13];

    let actual = (0..7)
        .map(fib)
        .collect::<Vec<_>>();

    assert_eq!(actual, expected);
}
```

Turbofish!

```
$ cd 1/hello-world-2
$ cargo test
Compiling hello-world-2 v0.1.0 (file:///Users/ilammy/Documents/dev/rust-talk/1/hello-world-2)
   Finished dev [unoptimized + debuginfo] target(s) in 2.55s
   Running /Users/ilammy/Documents/dev/rust-talk/target/debug/deps/hello_world_2-d79f06e28c42c0e5

running 1 test
test tests::fib_values ... ok

test result: ok.
1 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out
```

```rust
fn main() {
    loop {
        let mut input = String::new();
        io::stdin()
            .read_line(&mut input)
            .expect("read_line() error");

        match input.trim().parse() {
            Ok(n) => println!("fib({}) = {}", n, fib(n)),
            Err(_) => eprintln!("not an integer: {}", input),
        }
    }
}
```

Hello world III

Cargo.toml:

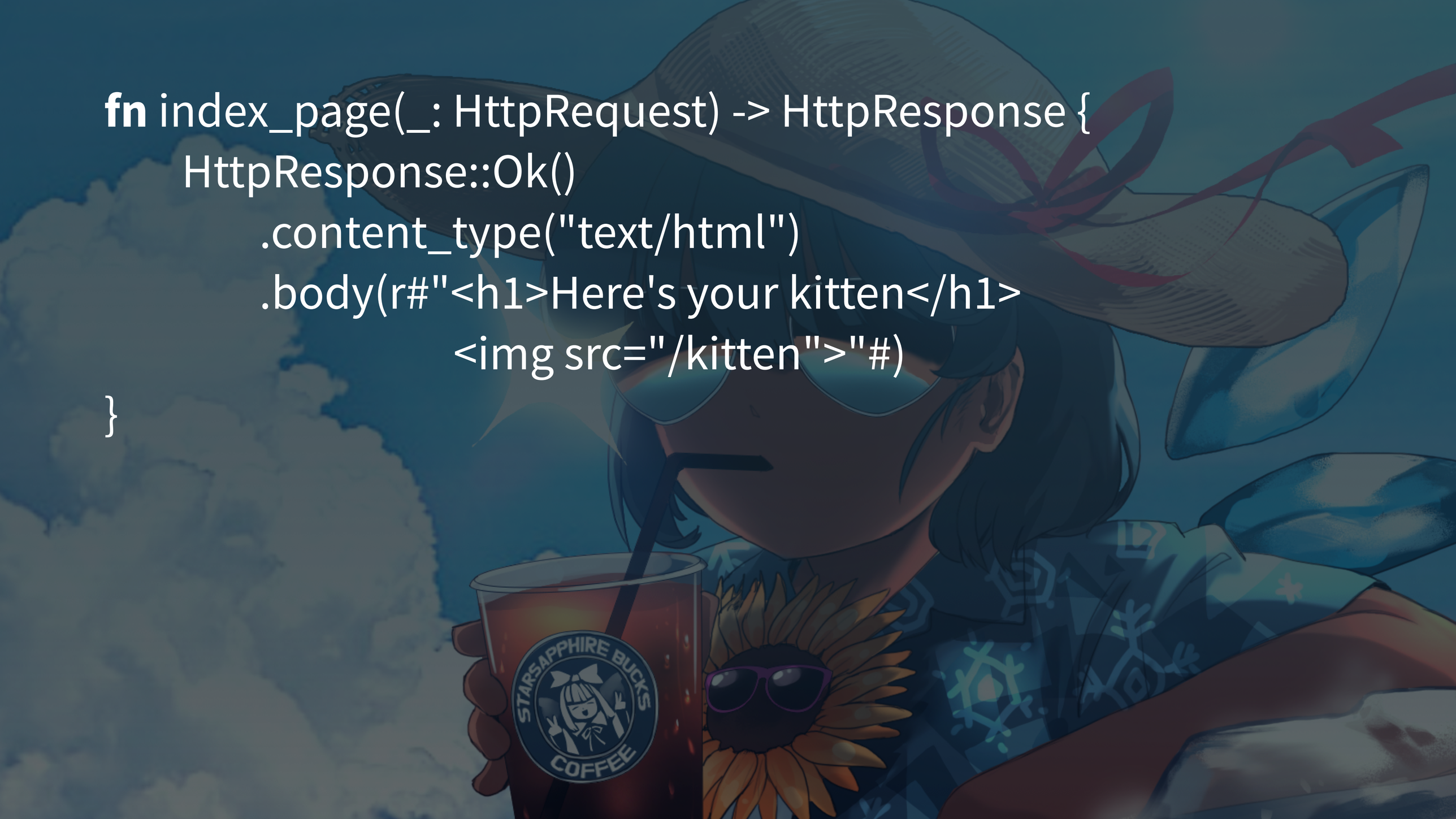
```
[dependencies]
actix-web = "^0.7.3"
```

main.rs:

```
extern crate actix_web;
use actix_web::{http, server, App,
    HttpRequest, HttpResponse};
```

```rust
fn main() {
    server::new(|| {
        App::new()
            .route("/", http::Method::GET, index_page)
            .route("/kitten", http::Method::GET, get_kitten)
    })
    .bind("0.0.0.0:8080")
    .expect("Can't listen on 8080 port")
    .run();
}
```

```
fn index_page(_: HttpRequest) -> HttpResponse {
    HttpResponse::Ok()
        .content_type("text/html")
        .body(r#"<h1>Here's your kitten</h1>
                <img src="/kitten">"#)
}
```



```
fn get_kitten(_: HttpRequest) -> HttpResponse {
    match get_kitten_image() {
        Ok(image_bytes) => HttpResponse::Ok()
            .content_type("image/jpeg")
            .body(image_bytes),

        Err(error) => HttpResponse::InternalServerError()
            .content_type("text/html")
            .body(format!("Error: {}", error)),
    }
}
```

```rust
fn get_kitten_image() -> Result<Vec<u8>, io::Error> {
    if let Some(path) = select_kitten_image() {
        let mut file = File::open(path)?;
        let mut bytes = Vec::new();
        file.read_to_end(&mut bytes)?;
        Ok(bytes)
    } else {
        Err(io::Error::new(io::ErrorKind::NotFound,
            "no kittens in current directory"))
    }
}
```

$ cd 1/hello-world-3

$ cargo run

```
  Compiling version_check v0.1.4
  Compiling libc v0.2.42
     (some 128 packages later)
  Compiling trust-dns-resolver v0.9.0
  Compiling actix v0.7.3
  Compiling hello-world-3 v0.1.0 (file:///Users/ilammy/Documents/dev/rust-talk/1/hello-world-3)
     Finished dev [unoptimized + debuginfo] target(s) in 1m 52s
     Running `/Users/ilammy/Documents/dev/rust-talk/target/debug/hello-world-3`
```

$ lsof -nP -i4TCP:8080

```
COMMAND      PID     USER    FD  TYPE DEVICE                SIZE/OFF  NODE  NAME
hello-world-3 51601 ilammy 3u  IPv4  0xd5fe20b3b4dacdd9  0t0       TCP    *:8080 (LISTEN)
```

localhost

Here's your kitten

ilammy — -bash — 80×24

```
✔ ~
12:11 $ curl -v http://localhost:8080
* Rebuilt URL to: http://localhost:8080/
*   Trying ::1...
* TCP_NODELAY set
* Connection failed
* connect to ::1 port 8080 failed: Connection refused
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 8080 (#0)
> GET / HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 200 OK
< content-length: 54
< content-type: text/html
< date: Sat, 18 Aug 2018 09:11:23 GMT
<
* Connection #0 to host localhost left intact
<h1>Here's your kitten</h1><img src="/random-kitten"/>✔ ~
12:11 $
```

Web Inspector — localhost

2 | 90.4 KB | 88.5ms | 0 | 0 | 0 | Search

Elements | Network | Resour... | Timelin... | Debug... | Storage | Canvas | Console

Filter Full URL | All | Document | CSS | Image | Font | JS | XHR | Other | Export

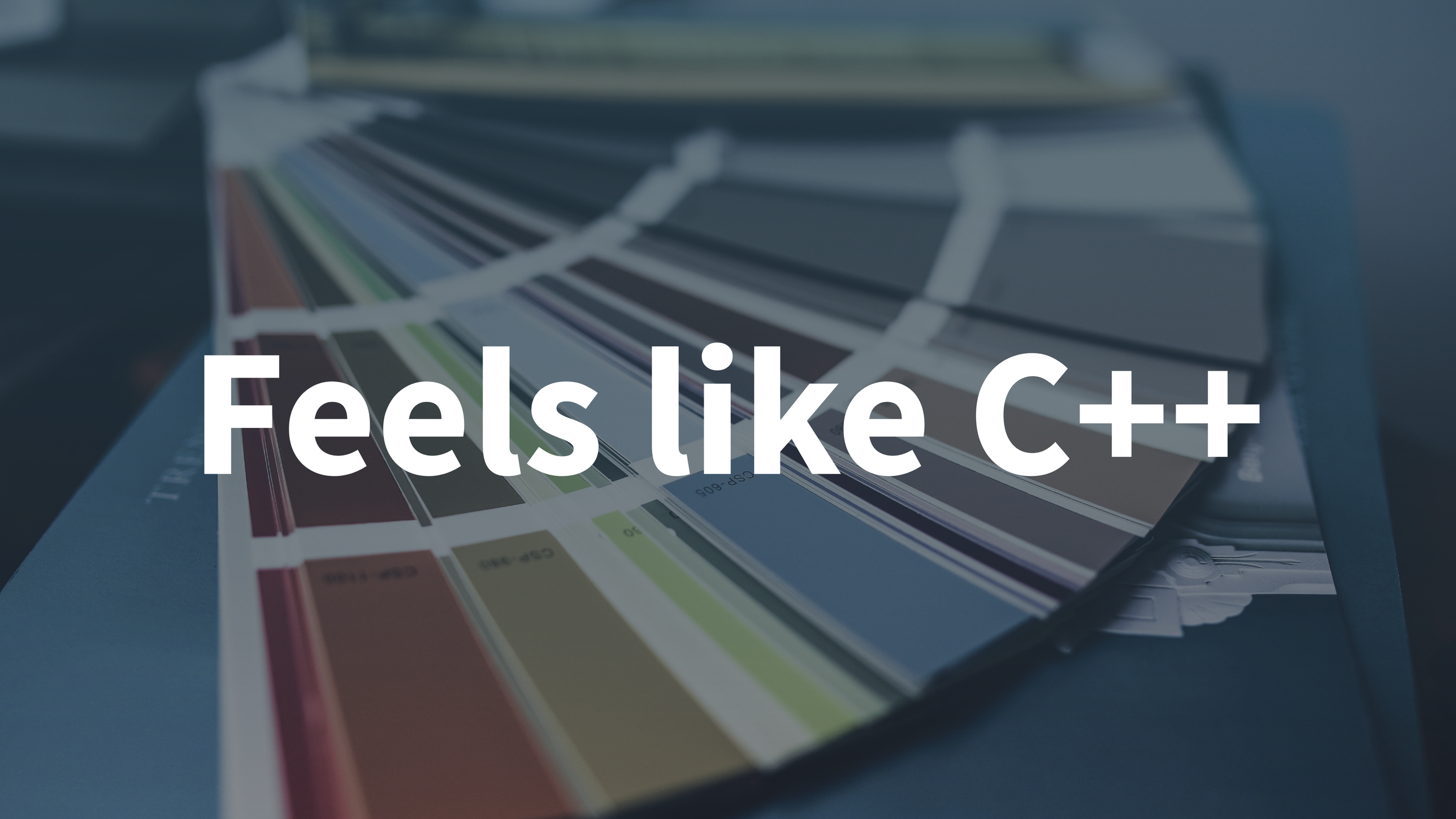
Name	Preview	Headers	Cookies	Sizes	Timing
localhost					
random-kitten					

Scheduling:
Queued 3.7ms

Response:
Waiting 15.4ms
Download 0.2ms

Totals:
Time to First Byte 19.1ms
Start to Finish 19.3ms

So why should I choose Rust?


Feels like C++

Systems language

Modern Safety

- zero-cost abstractions
- move semantics
- guaranteed memory safety
- threads without data races
- trait-based generics
- pattern matching
- type inference
- minimal runtime
- efficient C bindings

Cargo

project/Cargo.toml
project/Cargo.lock

project/benches
project/benches/large-input.rs

project/examples
project/examples/simple.rs

project/tests
project/tests/integration-tests.rs

project/src
project/src/lib.rs
project/src/main.rs

project/src/bin
project/src/bin/some-tool.rs

project/src/db
project/src/db/mod.rs
project/src/db/api.rs
project/src/db/storage.rs

```toml
[package]
name = "kernel"
version = "0.1.33"
build = "build.rs"

[lib]
name = "kernel"
path = "src/lib.rs"
crate-type = ["staticlib"]

[dependencies]
bitflags = "1.0.3"
clippy = { version = "0.0.209", optional = true }
linked_list_allocator = "0.6.2"
raw-cpuid = "4.0.0"
redox_syscall = { path = "syscall" }
slab_allocator = {
    path = "slab_allocator", optional = true }
spin = "0.4.8"
```

Cargo.toml

crates.io

crates.io
Rust Package Registry

Click or press 'S' to search...

Browse All Crates | Docs ▾ | Alexei Lozovsky ▾

Fork me on GitHub

rand 0.5.5

Follow

Homepage Documentation Repository Dependent crates

Cargo.toml

```
rand = "0.5.5"
```

Rand

build unknown | build passing | crates.io v0.5.5 | docs 0.5.5 | rustc 1.22+

A Rust library for random number generation.

Rand provides utilities to generate random numbers, to convert them to useful types and distributions, and some randomness-related algorithms.

The core random number generation traits of Rand live in the rand_core crate; this crate is most useful when implementing RNGs.

API reference: master branch, by release.

Usage

Add this to your `Cargo.toml`:

Last Updated
a day ago

Authors

- The Rust Project Developers

License

MIT/Apache-2.0

Keywords

rng random

Categories

- Algorithms

Owners

Trait Rng

Provided Methods

gen
gen_range
sample
sample_iter
fill
try_fill
gen_bool
choose
choose_mut
shuffle
gen_iter
gen_weighted_bool
gen_ascii_chars

Implementors

rand

Modules

```
    println!("{:?}", thread_rng().sample_iter(&Standard).take(5)
                                .collect::<Vec<(f64, bool)>>());

    // Dice-rolling:
    let die_range = Uniform::new_inclusive(1, 6);
    let mut roll_die = rng.sample_iter(&die_range);
    while roll_die.next().unwrap() != 6 {
        println!("Not a 6; rolling again!");
    }
```

[-] `fn fill<T: AsByteSliceMut + ?Sized>(&mut self, dest: &mut T)`

Fill `dest` entirely with random bytes (uniform value distribution), where `dest` is any type supporting `AsByteSliceMut`, namely slices and arrays over primitive integer types (`i8`, `i16`, `u32`, etc.).

On big-endian platforms this performs byte-swapping to ensure portability of results from reproducible generators.

This uses `fill_bytes` internally which may handle some RNG errors implicitly (e.g. waiting if the OS generator is not ready), but panics on other errors. See also `try_fill` which returns errors.

Example

```
use rand::{thread_rng, Rng};

let mut arr = [0i8; 20];
thread_rng().fill(&mut arr[..]);
```

[-]
```
fn try_fill<T: AsByteSliceMut + ?Sized>(
    &mut self,
    dest: &mut T
) -> Result<(), Error>
```

Fill `dest` entirely with random bytes (uniform value distribution), where `dest` is any type supporting `AsByteSliceMut`, namely slices and arrays over primitive integer types (`i8`, `i16`, `u32`, etc.).

On big-endian platforms this performs byte-swapping to ensure portability of results from reproducible generators.

cargo build
cargo test
cargo run

cargo bench

cargo doc

cargo fmt

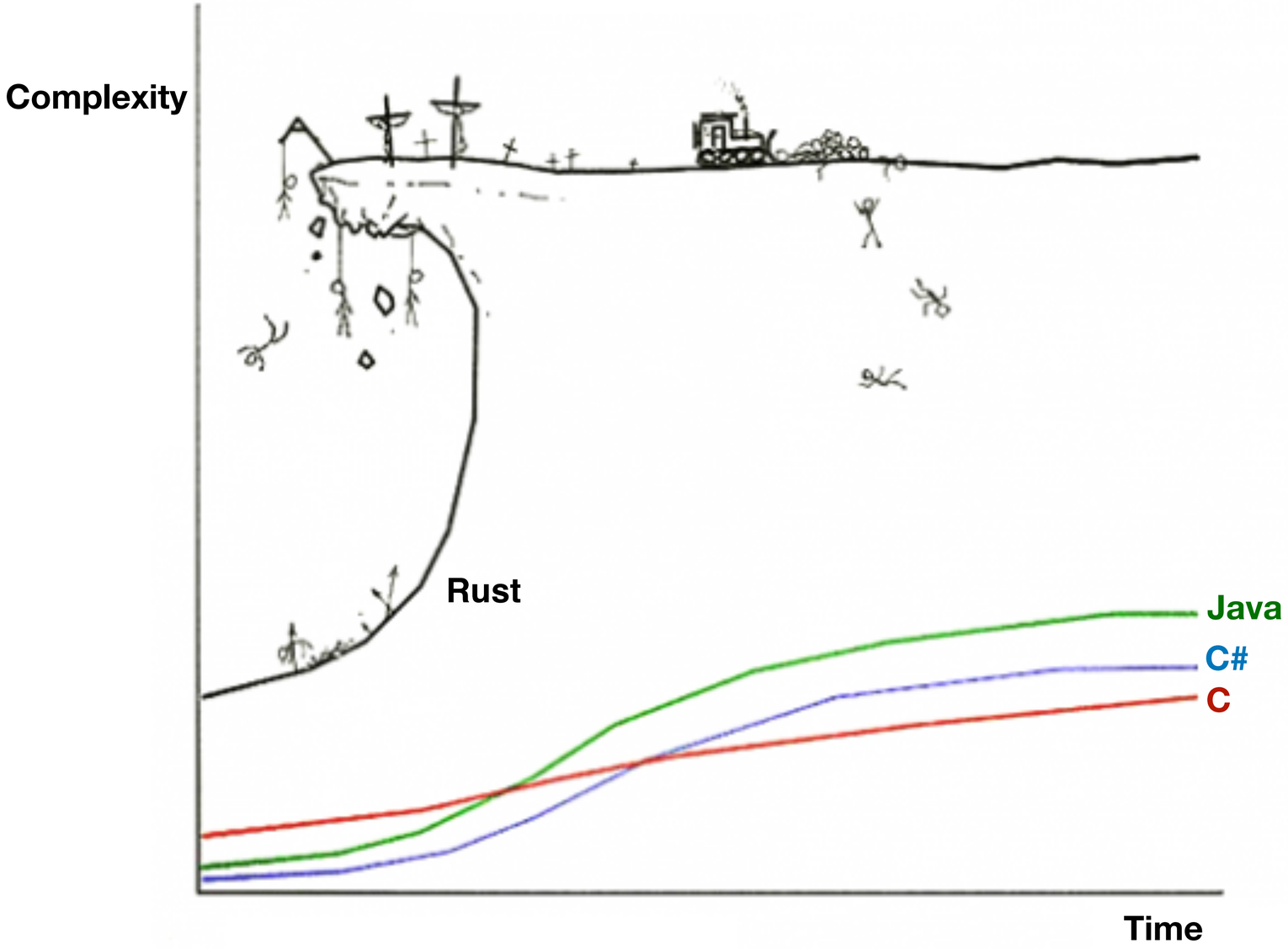
cargo audit
OLYMPUS
UPlanFL N
4x/0.13
∞/– /FN26.5

cargo publish

Opinionated
(i.e., easy to use)
build tools

SECURITY
CSS
HTML
QUIET
php
LARACON
git
BACKBONE.JS
nodc
NODE

Drawbacks



learning curve

community

stability

abandonment

node_modules

What’s next

“The Rust Programming Language”
https://github.com/sger/RustBooks
https://rustup.rs + IntelliJ IDEA