A Case For Rust

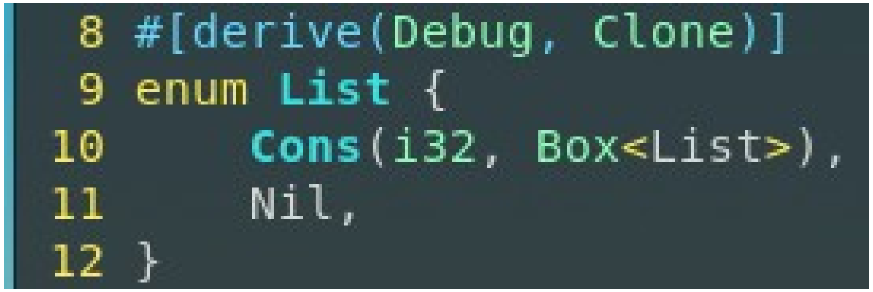
Oxidize your stack!

What is Rust?

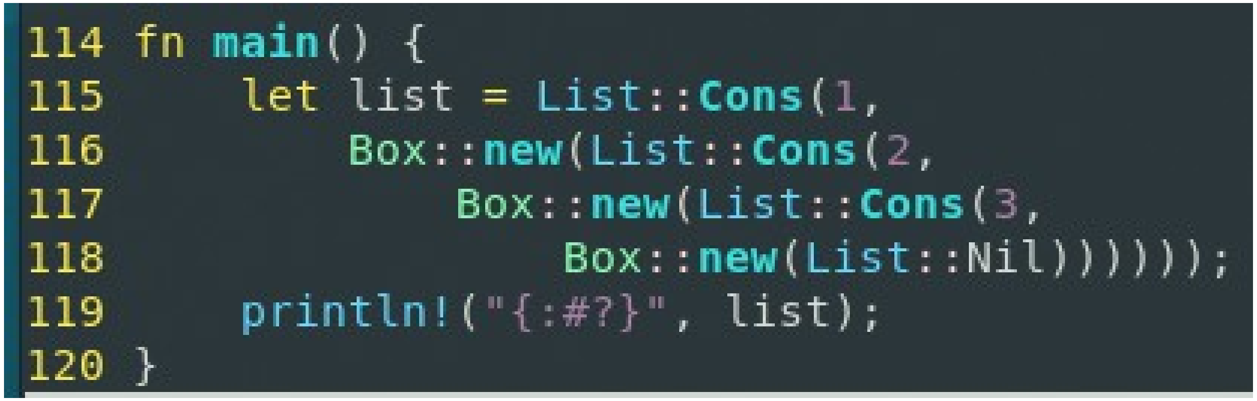
- **High level systems programming language**
  - No, this is not an oxymoron
- **High performance and highly productive**
- **Moves run time errors into compile time errors**
  - More on this later
- **Expressive type system**
- **Haskell in C++’s clothing**
- **Compiler enforces memory safety and disallows dataraces**
  - The compiler literally will refuse to compile code that has a datarace condition or potential memory corruption

Basic Language Features

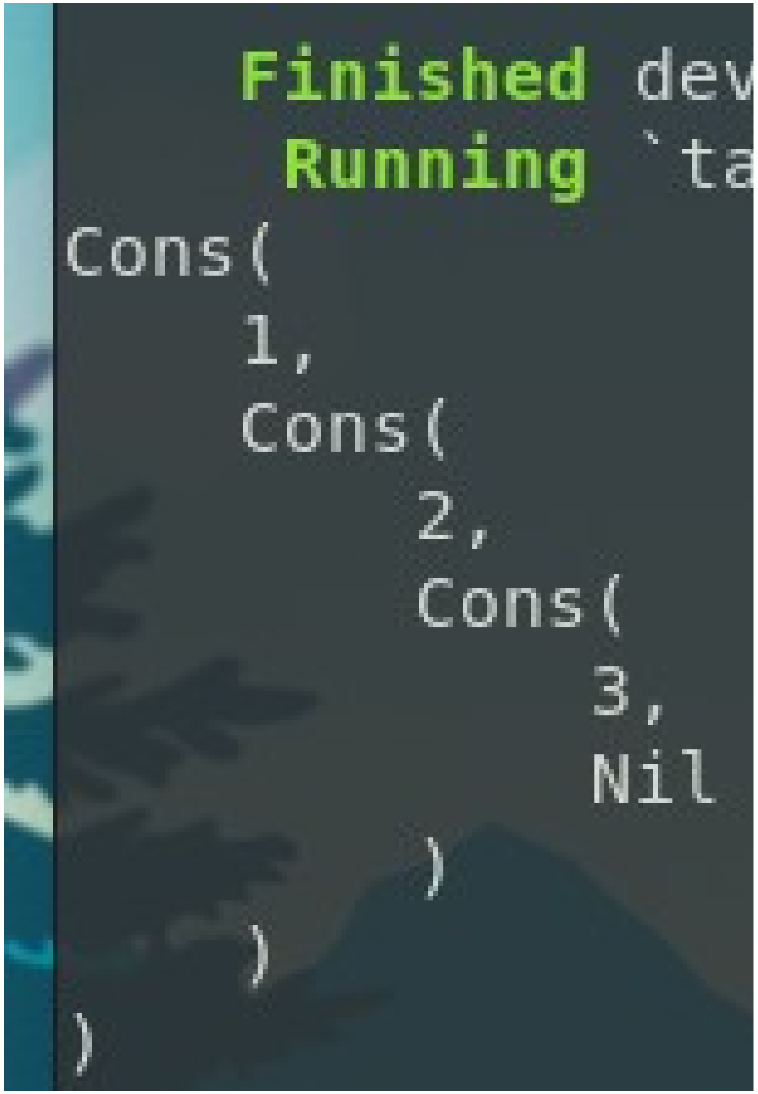
- **Of course, int, float, string…**
- **Also enum, but better!**
  - Rust enums are actually tagged unions



```rust
#[derive(Debug, Clone)]
enum List {
    Cons(i32, Box<List>),
    Nil,
}
```

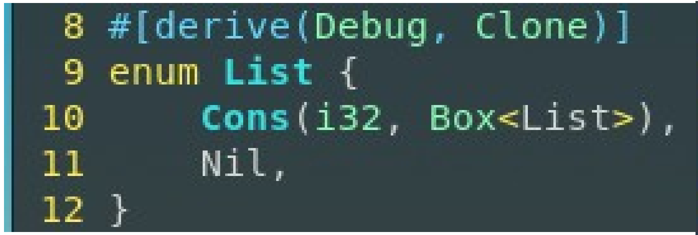


```rust
fn main() {
    let list = List::Cons(1,
        Box::new(List::Cons(2,
            Box::new(List::Cons(3,
                Box::new(List::Nil))))));
    println!("{:#?}", list);
}
```



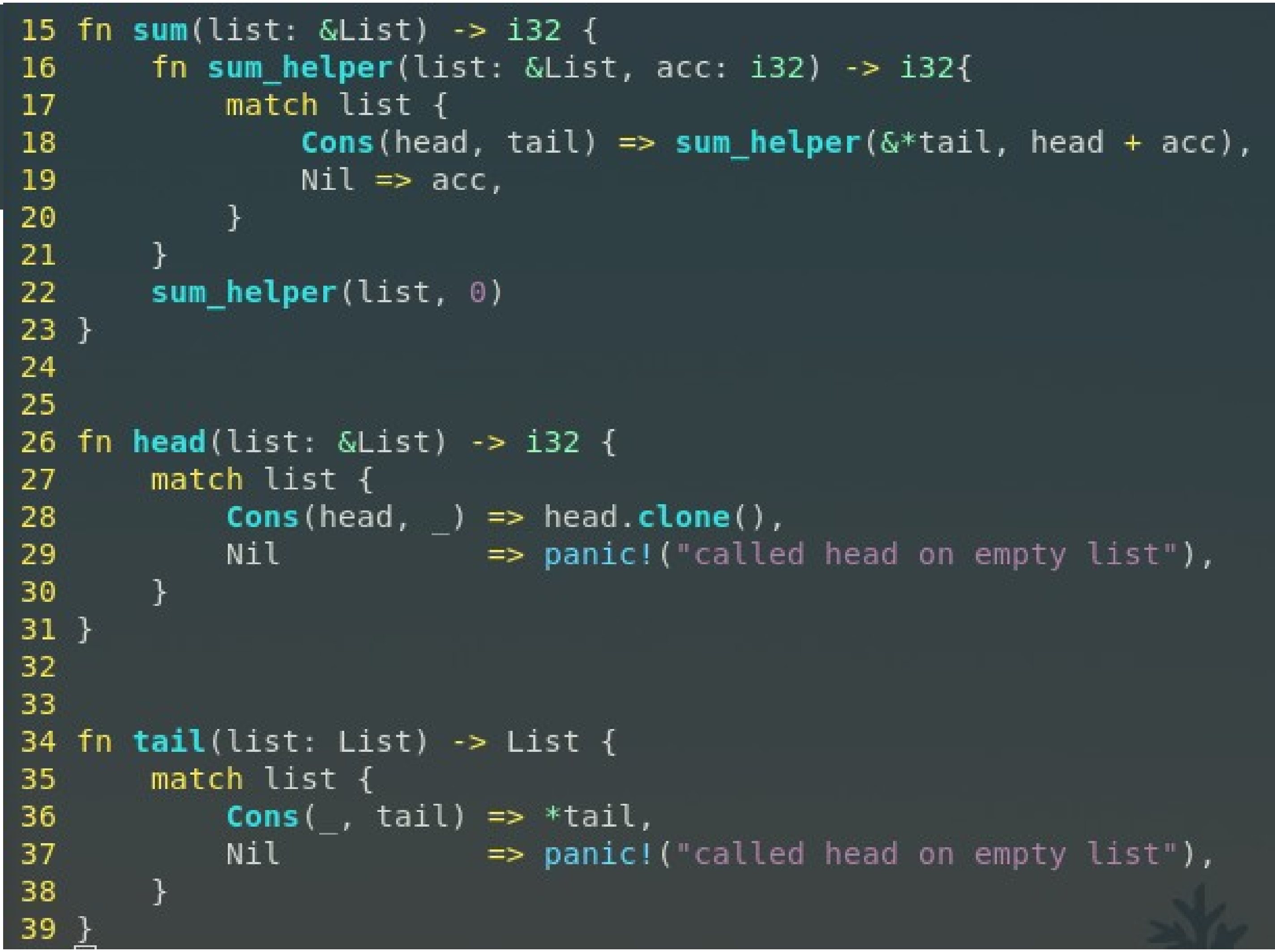
```
    Finished dev
     Running `ta
Cons(
    1,
    Cons(
        2,
        Cons(
            3,
            Nil
        )
    )
)
```

Enum and Pattern Matching



```
#[derive(Debug, Clone)]
enum List {
    Cons(i32, Box<List>),
    Nil,
}
```

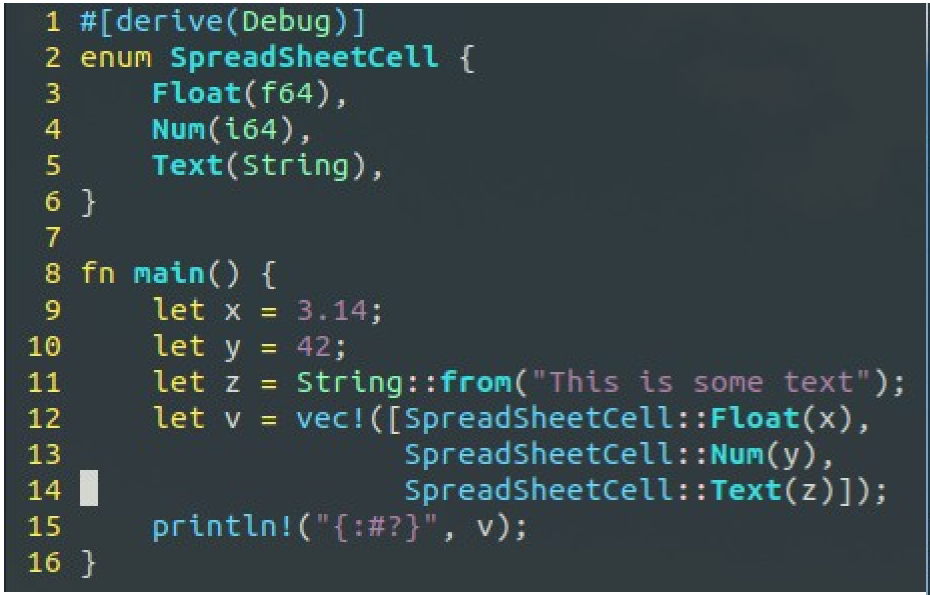
- Pattern matching on enums must be exhaustive, i.e., the compiler requires that you handle every case.
- Pattern matching allows you to access the fields of an enum



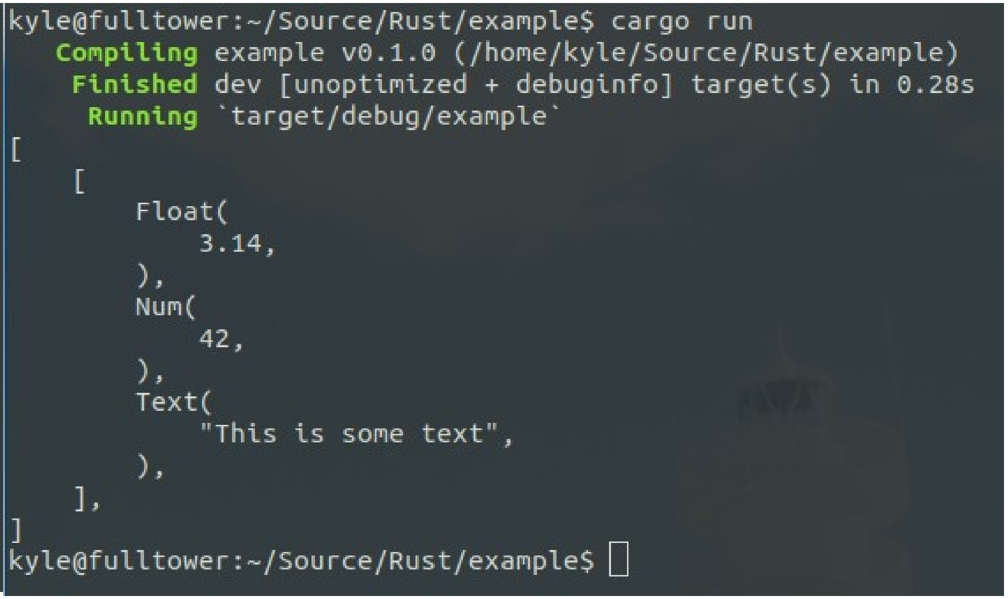
```
fn sum(list: &List) -> i32 {
    fn sum_helper(list: &List, acc: i32) -> i32{
        match list {
            Cons(head, tail) => sum_helper(&*tail, head + acc),
            Nil => acc,
        }
    }
    sum_helper(list, 0)
}


fn head(list: &List) -> i32 {
    match list {
        Cons(head, _) => head.clone(),
        Nil           => panic!("called head on empty list"),
    }
}


fn tail(list: List) -> List {
    match list {
        Cons(_, tail) => *tail,
        Nil           => panic!("called head on empty list"),
    }
}
```

Instead of Dynamic Typing...

- Rust is a strongly and staticly typed language
- One can however utilize enums in places where dynamic typing is useful, such as putting items into an array or vector
- One can then pattern match over the data to do the correct operation with the correct type



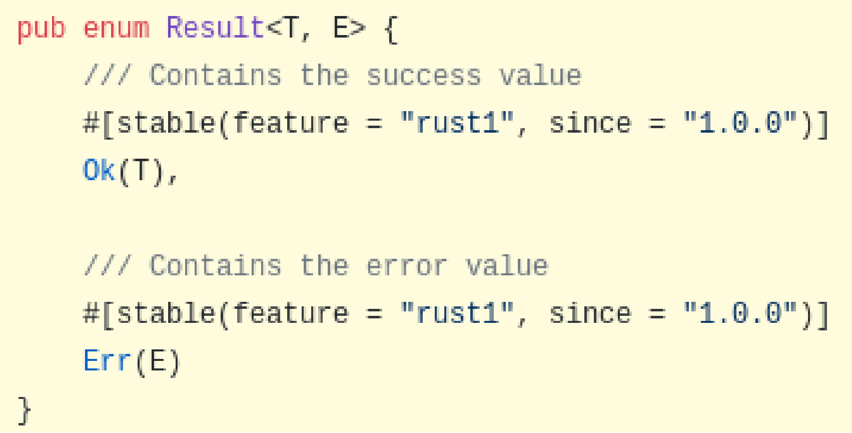
```rust
#[derive(Debug)]
enum SpreadSheetCell {
    Float(f64),
    Num(i64),
    Text(String),
}

fn main() {
    let x = 3.14;
    let y = 42;
    let z = String::from("This is some text");
    let v = vec!([SpreadSheetCell::Float(x),
                  SpreadSheetCell::Num(y),
                  SpreadSheetCell::Text(z)]);
    println!("{:#?}", v);
}
```



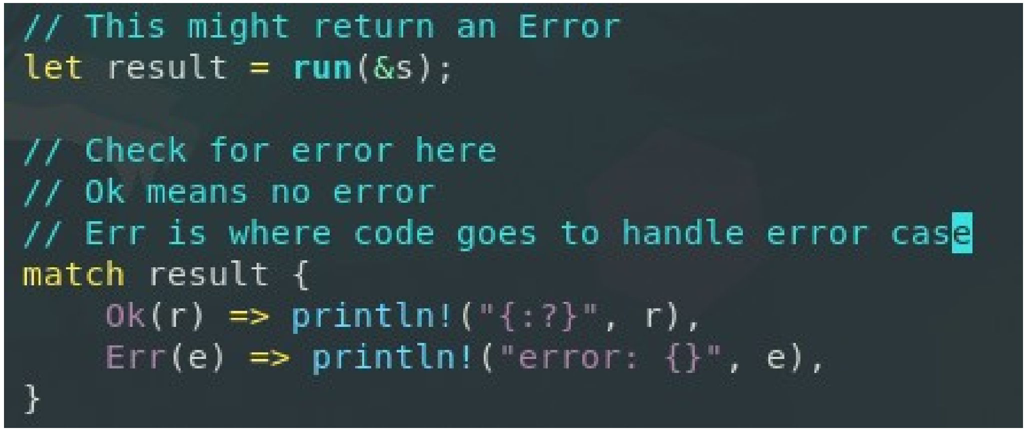
```
kyle@fulltower:~/Source/Rust/example$ cargo run
   Compiling example v0.1.0 (/home/kyle/Source/Rust/example)
    Finished dev [unoptimized + debuginfo] target(s) in 0.28s
     Running `target/debug/example`
[
    [
        Float(
            3.14,
        ),
        Num(
            42,
        ),
        Text(
            "This is some text",
        ),
    ],
]
kyle@fulltower:~/Source/Rust/example$
```

Error Handling

- **No Exceptions!**
- **Exceptions do not exist in the language, and neither does Null**
  - This means it is not possible to dereference a Null pointer!
- **Errors must be handled explicitly, or returned to the caller by using a Result type**



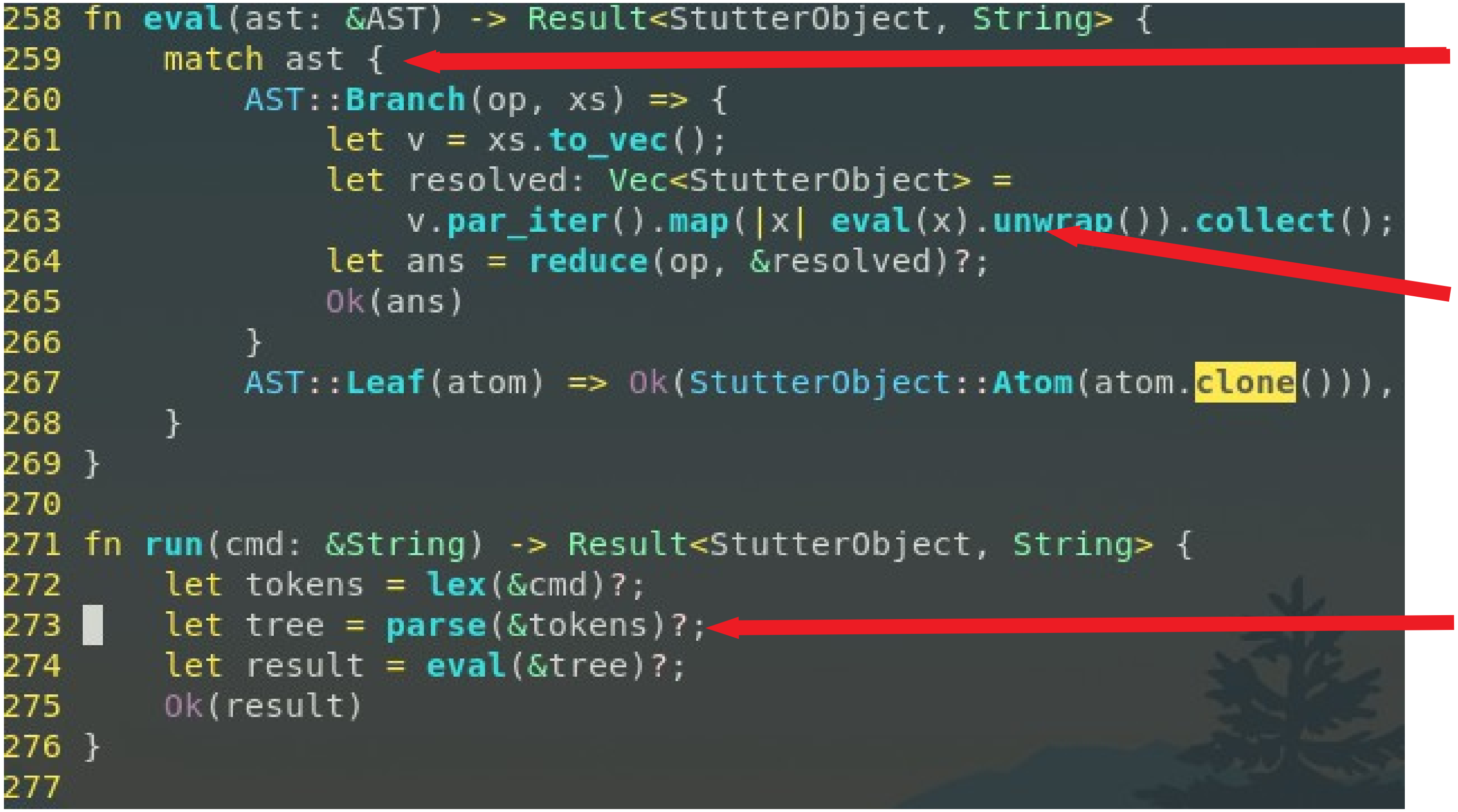
```
pub enum Result<T, E> {
    /// Contains the success value
    #[stable(feature = "rust1", since = "1.0.0")]
    Ok(T),

    /// Contains the error value
    #[stable(feature = "rust1", since = "1.0.0")]
    Err(E)
}
```



```
// This might return an Error
let result = run(&s);

// Check for error here
// Ok means no error
// Err is where code goes to handle error case
match result {
    Ok(r) => println!("{:?}", r),
    Err(e) => println!("error: {}", e),
}
```

Error handling and Patern Matching in Action!

Error handling as a part of the type system



```rust
fn eval(ast: &AST) -> Result<StutterObject, String> {
    match ast {
        AST::Branch(op, xs) => {
            let v = xs.to_vec();
            let resolved: Vec<StutterObject> =
                v.par_iter().map(|x| eval(x).unwrap()).collect();
            let ans = reduce(op, &resolved)?;
            Ok(ans)
        }
        AST::Leaf(atom) => Ok(StutterObject::Atom(atom.clone())),
    }
}

fn run(cmd: &String) -> Result<StutterObject, String> {
    let tokens = lex(&cmd)?;
    let tree = parse(&tokens)?;
    let result = eval(&tree)?;
    Ok(result)
}
```

Exhaustive Pattern Matching

Closures (AKA Anonymous functions, AKA Lambda functions)

Either handle errors as they come, or pass them down the line as a Result::Err type

No Segfaults

- **The Rust compiler checks for any potential memory issues such as**
  - Dangling pointers
  - Use after free
  - Buffer overflows
  - Race conditions while multithreading
- **The Rust compiler checks for these conditions and will not compile your code if any of them are present**

Easier Multithreading

- Even if you don’t write close to the metal code and use a GC’d language, everyone has to worry about race conditions in multithreading
- Rust will ensure that your code has no race conditions at compile time
- This eliminates an entire class of errors that can be very hard to detect and even harder to fix

Easier Multithreading

- **How does Rust ensure no race conditions?**
- **Rust uses something called a “Borrow Checker” to ensure that there is no more than 1 mutable reference to an object at any given time**
- **Rust supports message passing to communicate between threads**
- **Rust also supports shared memory, and requires that you uphold that no 2 threads can mutate the memory at once**
- **You can do this by wrapping a shared object in a *Arc* smart pointer, and using a mutex lock**
  - Arc is an **A**tomic **R**eference **C**ount smart pointer that uses hardware provided atomic primitives to ensure that only one thread can alter the reference count at a time

RAII Instead of GC

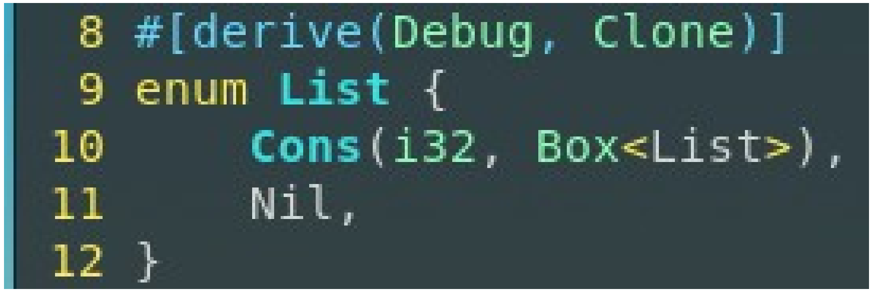
- **RAII**
  - Resource
  - Acquisition
  - Is
  - Initialization
- **This is a concept borrowed from C++, which means**
  - Allocate memory for an object when it comes in scope
  - Deallocate memory when the object goes out of scope

RAII Instead of GC

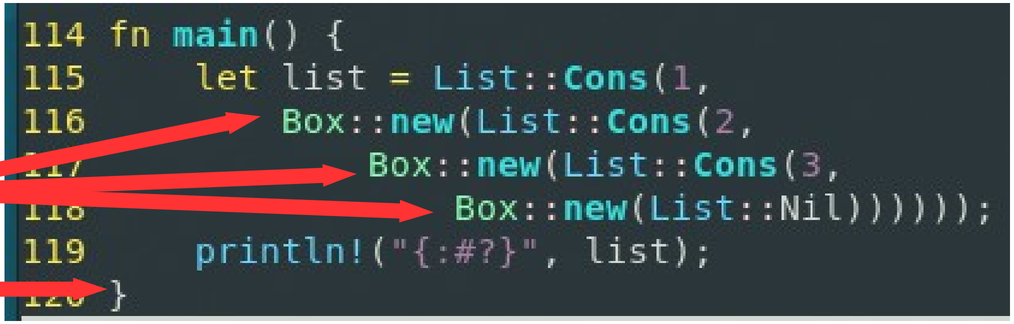
- Rust favors stack allocation over heap allocation, which cleans up memory for free
- Heap allocations are done using smart pointers, which free the memory when it goes out of scope
- Smart pointers mean you can `new` whatever you want without having to worry about calling `delete` later
- You also can call obj.drop() if you want to manage memory manually

RAII Instead of GC

- **Remember this code?**
  - Box<List> means this is a smart pointer
  - Box::new heap allocates an object within a smart pointer



```
#[derive(Debug, Clone)]
enum List {
    Cons(i32, Box<List>),
    Nil,
}
```



```
fn main() {
    let list = List::Cons(1,
        Box::new(List::Cons(2,
            Box::new(List::Cons(3,
                Box::new(List::Nil))))));
    println!("{:#?}", list);
}
```

- Memory Allocated here
- Memory Deallocated here

RAII Instead of GC

- This means you get all the benefits of manual memory managed systems without having to actually manage memory yourself!
- Rust has NO Garbage Collector, meaning you get performance on par with C

Performance

- **Move by default**
  - No copy constructor
  - Instead, if you want to copy, then you must explicitly .clone() your object
  - Pass by reference is encouraged by the compiler
- **Most checks for safety are done at compile time, so there is a minimal runtime environment (on par with C)**
- **Generics are done in a similar fashion to C++, i.e. through templates that are resolved at compile time**
  - This is in contrast to many high level languages such as Java and C# which use runtime indirection through pointers to achieve generics

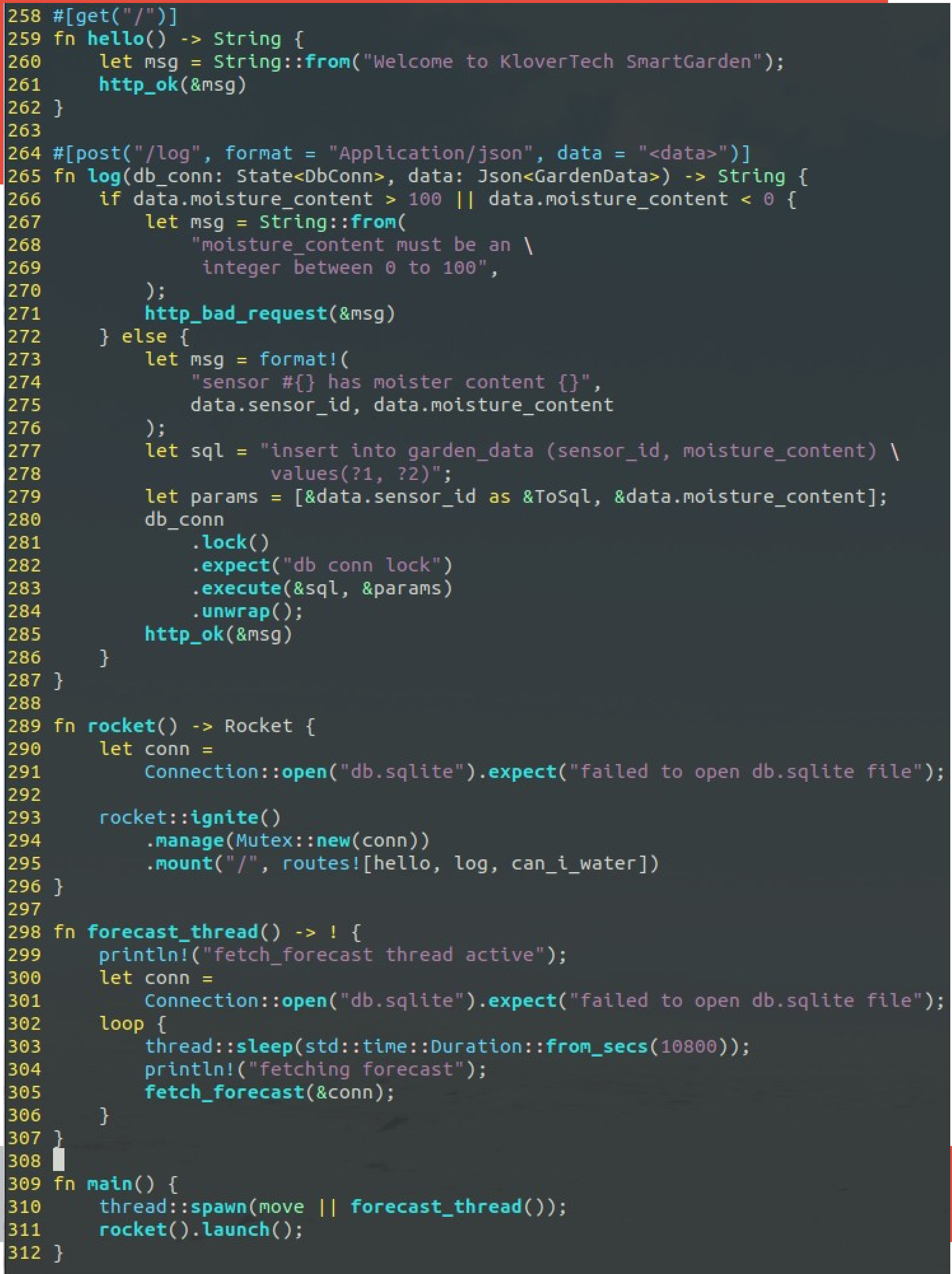
Ecosystem

- **Helpful compiler with easy to understand error messages**
  - Anyone who has made an error with C++ templates can understand how helpful good and concise error messages can be
- **Companies already using it**
  - Mozilla (Sponsor and largest contributor) – Firefox
  - Dropbox
  - Cloudflare
  - Sensirion – Embedded Sensors
  - Amazon – Firecracker
  - Google – Fuchsia
  - Facebook – Libra, Mononoke
  - Microsoft – IoT Edge

Ecosystem

- Rust has existing frameworks and packages to support many use cases, including…
  - Web Frameworks
    - Rocket – (my favorite and similar to Flask)
    - Iron
    - Nickle
    - Warp
    - H2
    - Hyper
    - Tiny-http
  - Front end Web through WASM
    - smithy
  - Game Engines
    - Amethyst
    - Piston
    - SDL bindings
  - Embedded
    - STM32 (ARM)
    - AVR

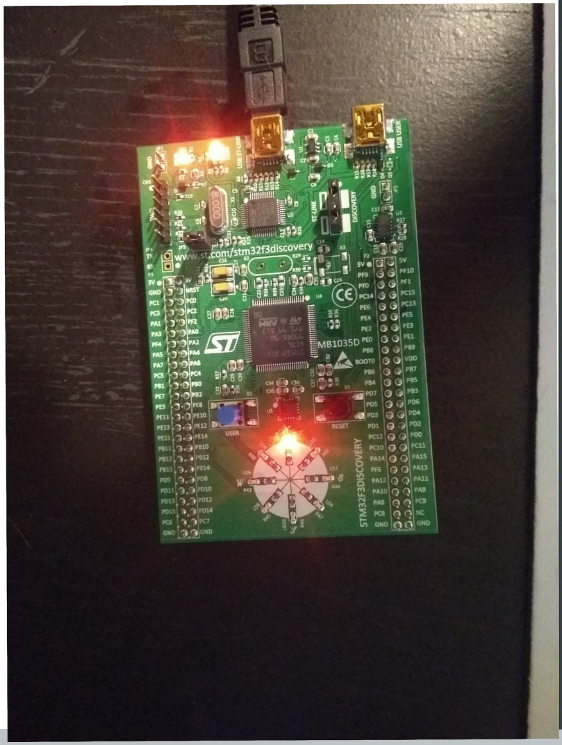
Web –
Rocket Framework

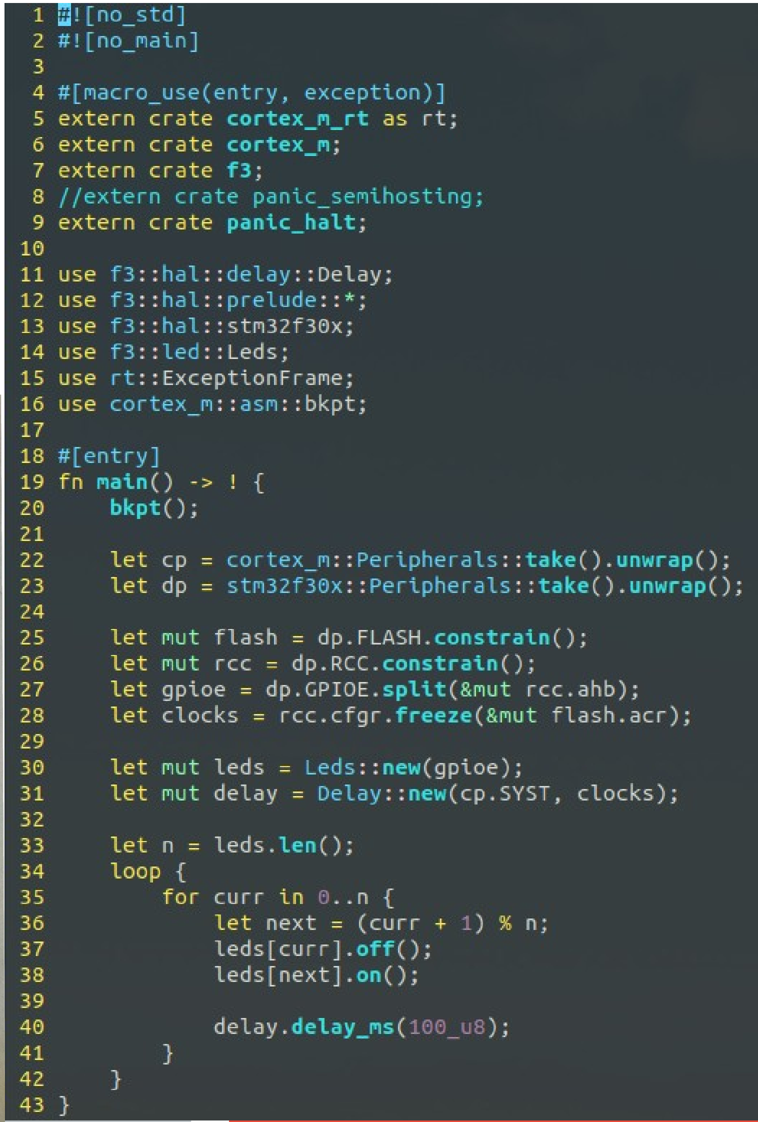


```
#[get("/")]
fn hello() -> String {
    let msg = String::from("Welcome to KloverTech SmartGarden");
    http_ok(&msg)
}

#[post("/log", format = "Application/json", data = "<data>")]
fn log(db_conn: State<DbConn>, data: Json<GardenData>) -> String {
    if data.moisture_content > 100 || data.moisture_content < 0 {
        let msg = String::from(
            "moisture_content must be an \
             integer between 0 to 100",
        );
        http_bad_request(&msg)
    } else {
        let msg = format!(
            "sensor #{} has moister content {}",
            data.sensor_id, data.moisture_content
        );
        let sql = "insert into garden_data (sensor_id, moisture_content) \
                   values(?1, ?2)";
        let params = [&data.sensor_id as &ToSql, &data.moisture_content];
        db_conn
            .lock()
            .expect("db conn lock")
            .execute(&sql, &params)
            .unwrap();
        http_ok(&msg)
    }
}

fn rocket() -> Rocket {
    let conn =
        Connection::open("db.sqlite").expect("failed to open db.sqlite file");

    rocket::ignite()
        .manage(Mutex::new(conn))
        .mount("/", routes![hello, log, can_i_water])
}

fn forecast_thread() -> ! {
    println!("fetch_forecast thread active");
    let conn =
        Connection::open("db.sqlite").expect("failed to open db.sqlite file");
    loop {
        thread::sleep(std::time::Duration::from_secs(10800));
        println!("fetching forecast");
        fetch_forecast(&conn);
    }
}

fn main() {
    thread::spawn(move || forecast_thread());
    rocket().launch();
}
```

Embedded

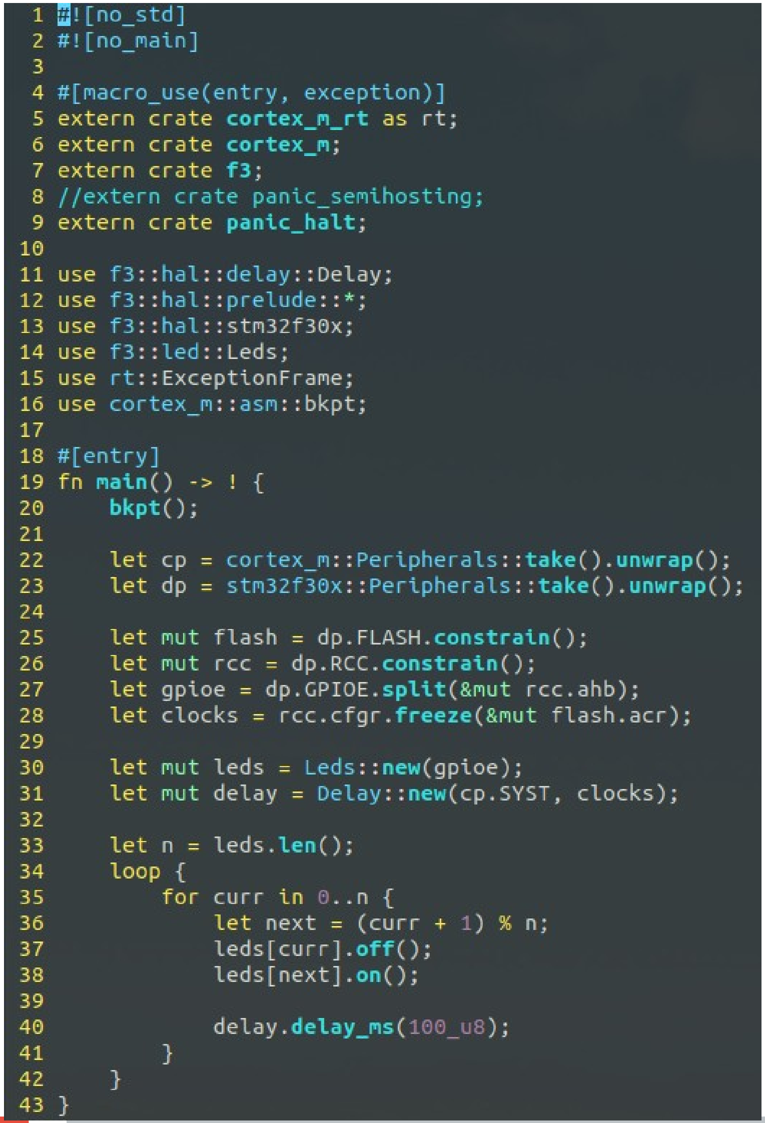
- This code runs on a microcontroller with no OS or runtime environment
  - 40 KB RAM
  - 32 bit
  - 72 MHz Clock



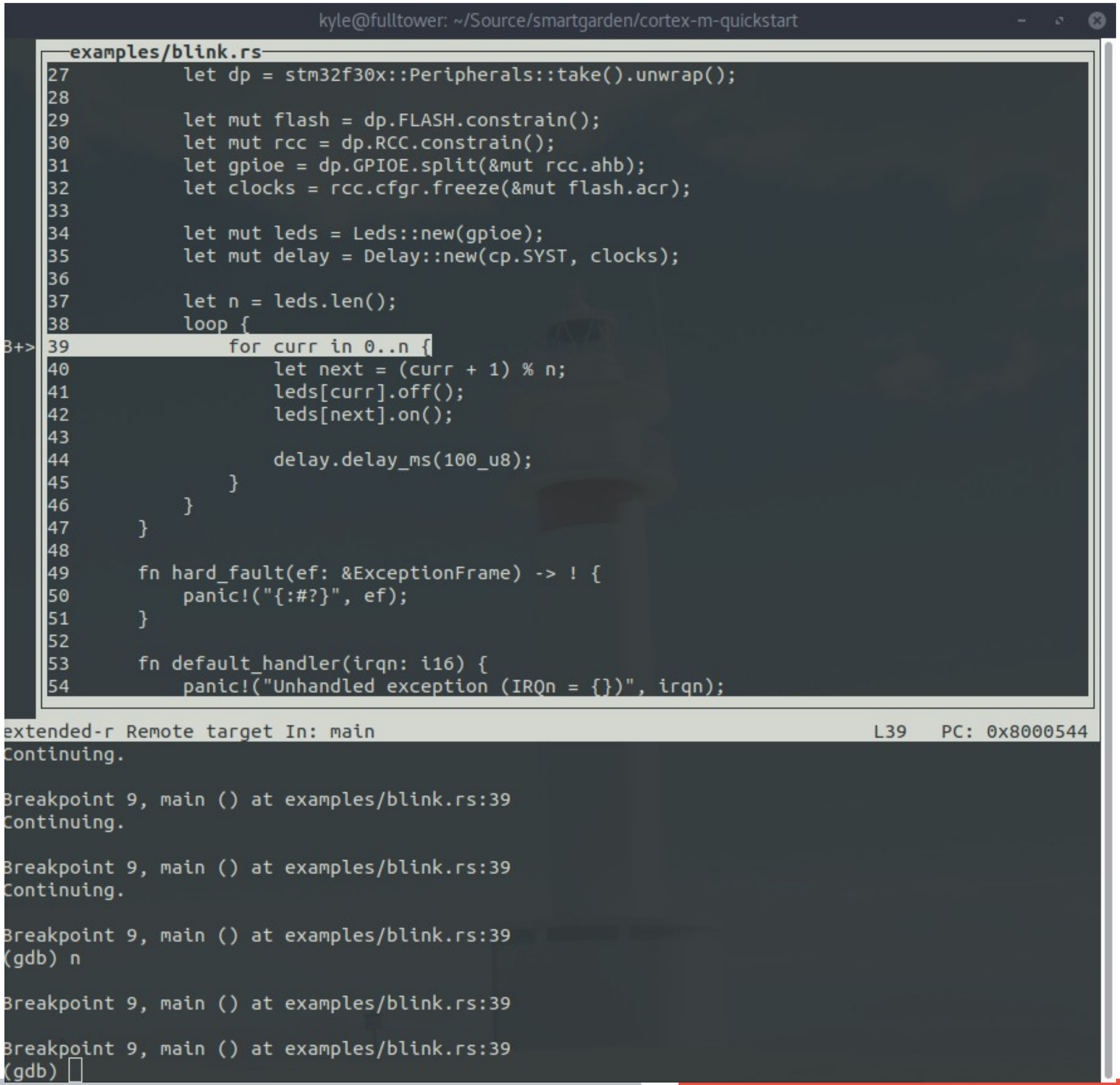


```
#![no_std]
#![no_main]

#[macro_use(entry, exception)]
extern crate cortex_m_rt as rt;
extern crate cortex_m;
extern crate f3;
//extern crate panic_semihosting;
extern crate panic_halt;

use f3::hal::delay::Delay;
use f3::hal::prelude::*;
use f3::hal::stm32f30x;
use f3::led::Leds;
use rt::ExceptionFrame;
use cortex_m::asm::bkpt;

#[entry]
fn main() -> ! {
    bkpt();

    let cp = cortex_m::Peripherals::take().unwrap();
    let dp = stm32f30x::Peripherals::take().unwrap();

    let mut flash = dp.FLASH.constrain();
    let mut rcc = dp.RCC.constrain();
    let gpioe = dp.GPIOE.split(&mut rcc.ahb);
    let clocks = rcc.cfgr.freeze(&mut flash.acr);

    let mut leds = Leds::new(gpioe);
    let mut delay = Delay::new(cp.SYST, clocks);

    let n = leds.len();
    loop {
        for curr in 0..n {
            let next = (curr + 1) % n;
            leds[curr].off();
            leds[next].on();

            delay.delay_ms(100_u8);
        }
    }
}
```

Full GDB Support



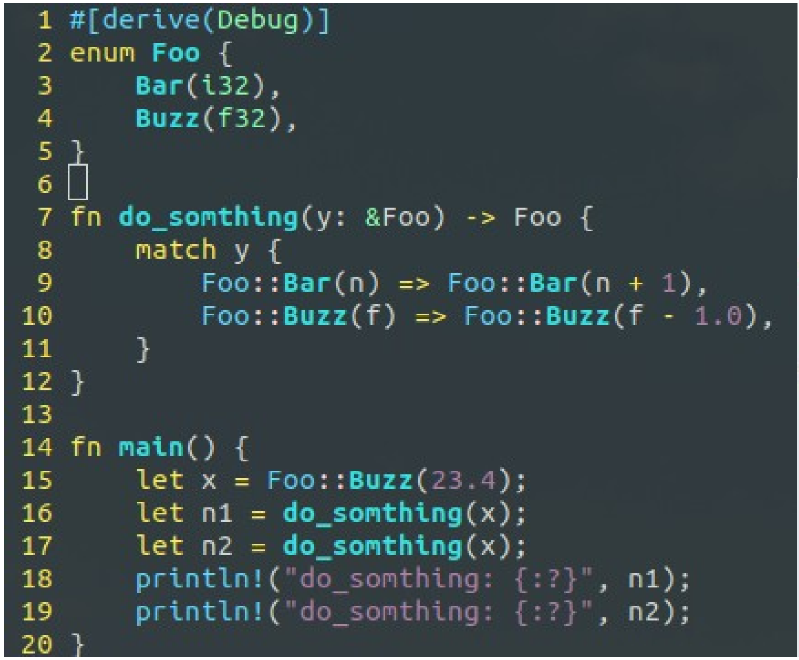
```
#![no_std]
#![no_main]

#[macro_use(entry, exception)]
extern crate cortex_m_rt as rt;
extern crate cortex_m;
extern crate f3;
//extern crate panic_semihosting;
extern crate panic_halt;

use f3::hal::delay::Delay;
use f3::hal::prelude::*;
use f3::hal::stm32f30x;
use f3::led::Leds;
use rt::ExceptionFrame;
use cortex_m::asm::bkpt;

#[entry]
fn main() -> ! {
    bkpt();

    let cp = cortex_m::Peripherals::take().unwrap();
    let dp = stm32f30x::Peripherals::take().unwrap();

    let mut flash = dp.FLASH.constrain();
    let mut rcc = dp.RCC.constrain();
    let gpioe = dp.GPIOE.split(&mut rcc.ahb);
    let clocks = rcc.cfgr.freeze(&mut flash.acr);

    let mut leds = Leds::new(gpioe);
    let mut delay = Delay::new(cp.SYST, clocks);

    let n = leds.len();
    loop {
        for curr in 0..n {
            let next = (curr + 1) % n;
            leds[curr].off();
            leds[next].on();

            delay.delay_ms(100_u8);
        }
    }
}
```



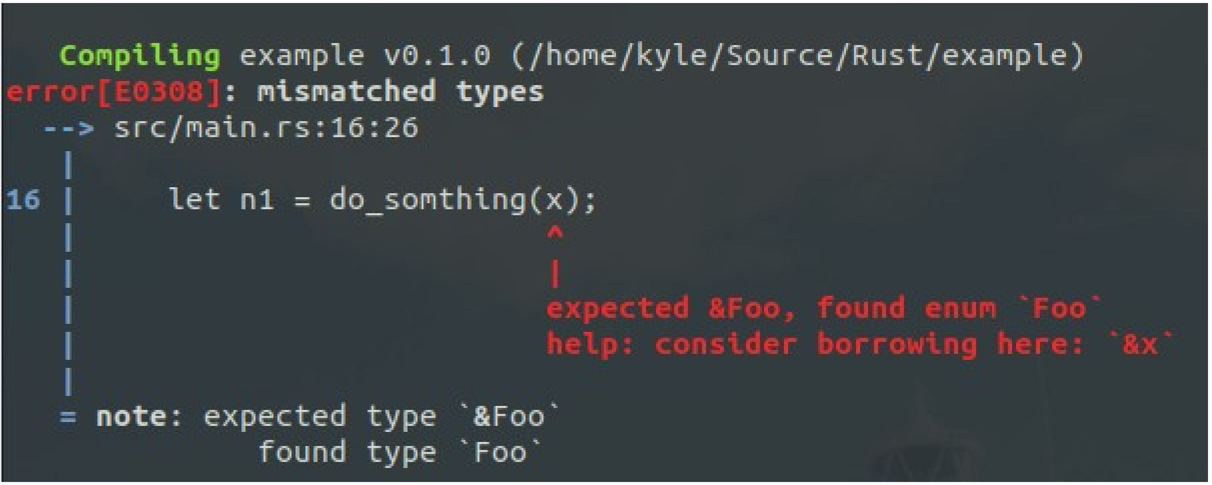
```
examples/blink.rs
27          let dp = stm32f30x::Peripherals::take().unwrap();
28
29          let mut flash = dp.FLASH.constrain();
30          let mut rcc = dp.RCC.constrain();
31          let gpioe = dp.GPIOE.split(&mut rcc.ahb);
32          let clocks = rcc.cfgr.freeze(&mut flash.acr);
33
34          let mut leds = Leds::new(gpioe);
35          let mut delay = Delay::new(cp.SYST, clocks);
36
37          let n = leds.len();
38          loop {
B+> 39              for curr in 0..n {
40                  let next = (curr + 1) % n;
41                  leds[curr].off();
42                  leds[next].on();
43
44                  delay.delay_ms(100_u8);
45              }
46          }
47      }
48
49      fn hard_fault(ef: &ExceptionFrame) -> ! {
50          panic!("{:#?}", ef);
51      }
52
53      fn default_handler(irqn: i16) {
54          panic!("Unhandled exception (IRQn = {})", irqn);
extended-r Remote target In: main                         L39   PC: 0x8000544
Continuing.

Breakpoint 9, main () at examples/blink.rs:39
Continuing.

Breakpoint 9, main () at examples/blink.rs:39
Continuing.

Breakpoint 9, main () at examples/blink.rs:39
(gdb) n

Breakpoint 9, main () at examples/blink.rs:39

Breakpoint 9, main () at examples/blink.rs:39
(gdb) 
```

Helpful Compiler

- The Rust compiler produces very helpful error messages and often will tell you what code to copy and paste to make your program compile properly



```
 1 #[derive(Debug)]
 2 enum Foo {
 3     Bar(i32),
 4     Buzz(f32),
 5 }
 6 
 7 fn do_somthing(y: &Foo) -> Foo {
 8     match y {
 9         Foo::Bar(n) => Foo::Bar(n + 1),
10         Foo::Buzz(f) => Foo::Buzz(f - 1.0),
11     }
12 }
13 
14 fn main() {
15     let x = Foo::Buzz(23.4);
16     let n1 = do_somthing(x);
17     let n2 = do_somthing(x);
18     println!("do_somthing: {:?}", n1);
19     println!("do_somthing: {:?}", n2);
20 }
```

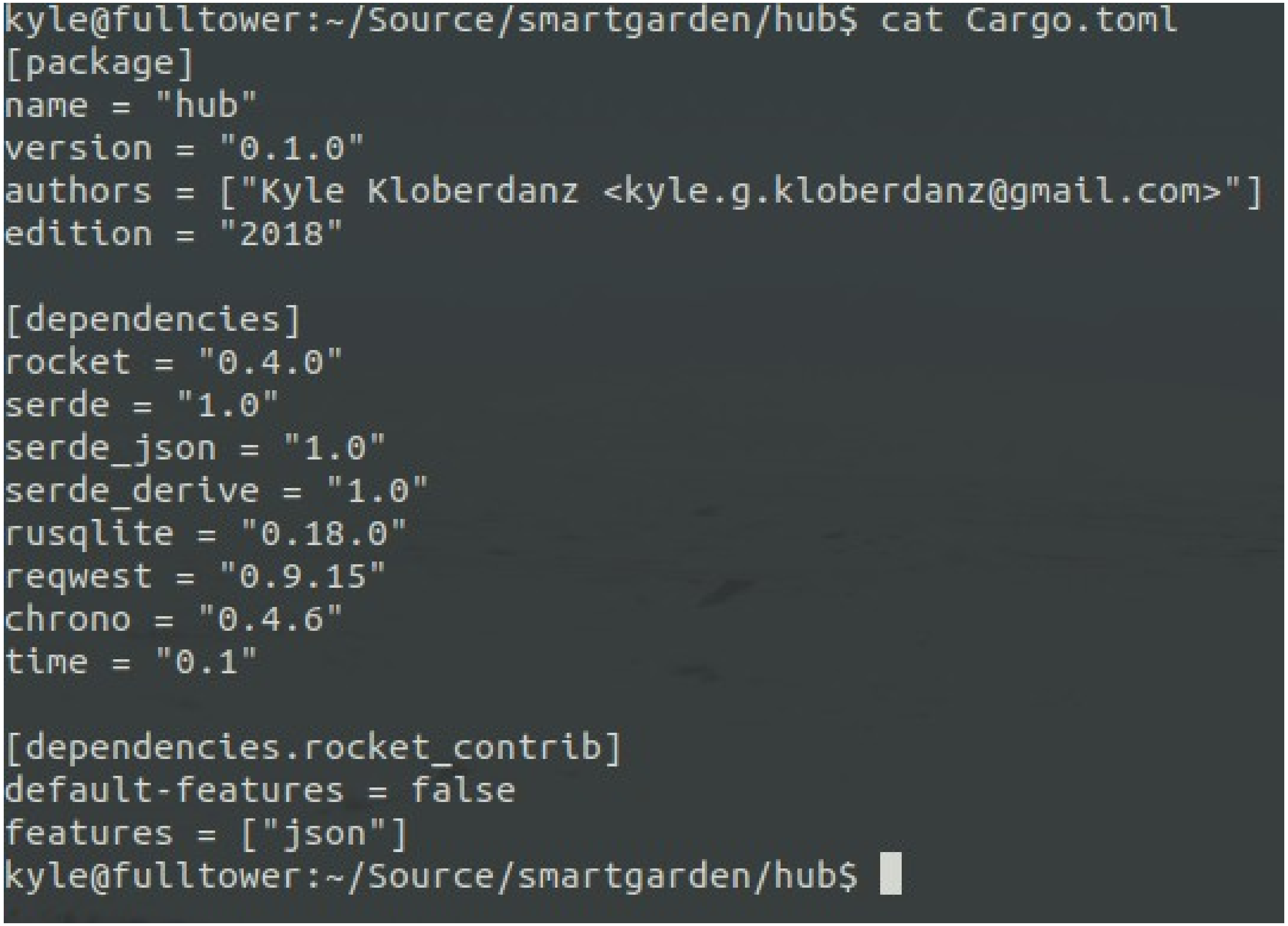


```
   Compiling example v0.1.0 (/home/kyle/Source/Rust/example)
error[E0308]: mismatched types
  --> src/main.rs:16:26
   |
16 |     let n1 = do_somthing(x);
   |                          ^
   |                          |
   |                          expected &Foo, found enum `Foo`
   |                          help: consider borrowing here: `&x`
   |
   = note: expected type `&Foo`
              found type `Foo`
```

Build System and Package Management

- Cargo is the build system and package manager
- All dependencies are pinned in the Cargo.toml file
- To build, you simply type `cargo build` and cargo will fetch the dependencies from Cargo.toml, compile them, then compile your project with just one command
- To build and run, simply type `cargo run`
- Builds are automatically done in parallel with however many cores you have available

Build System and Package Management



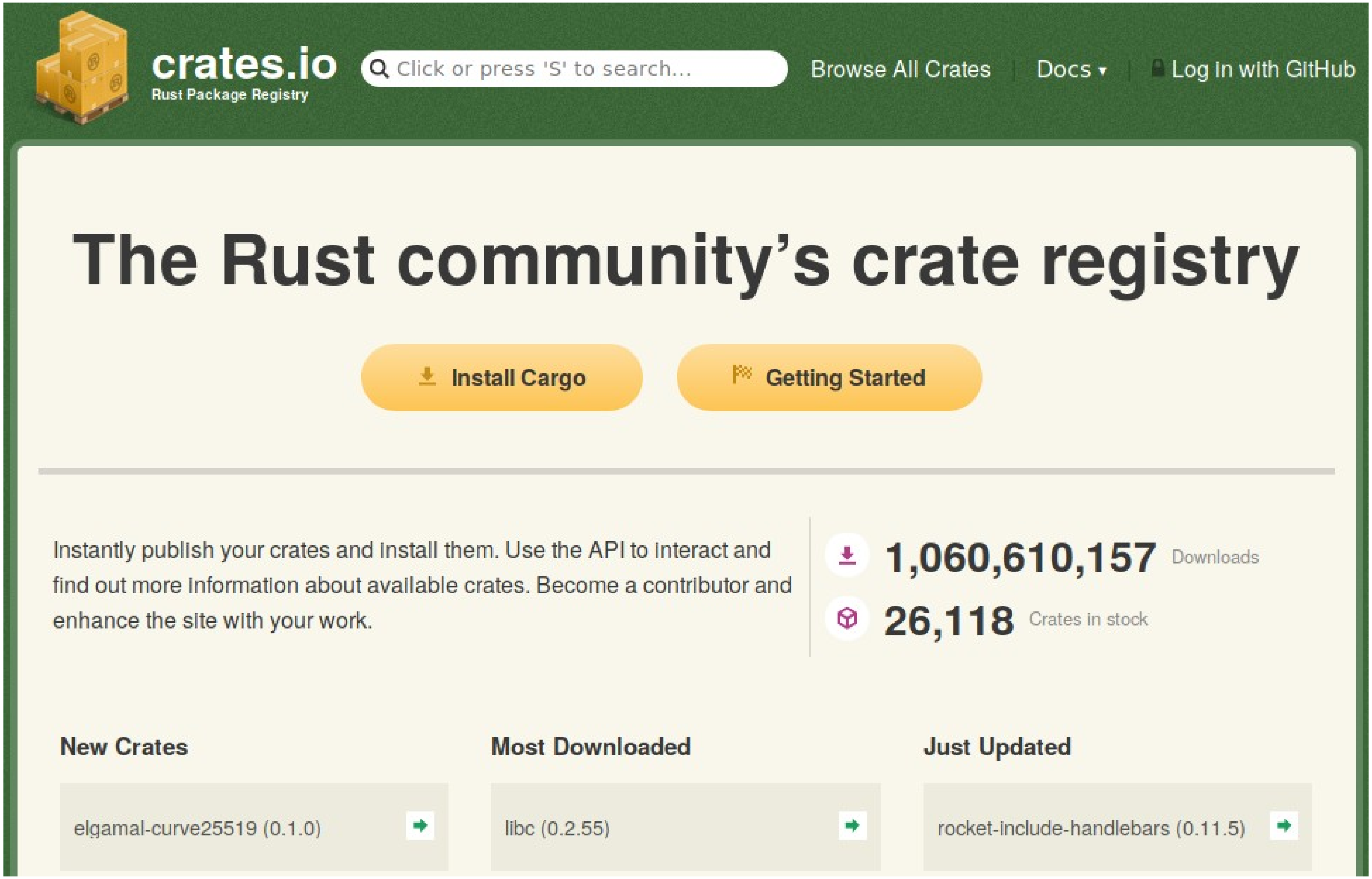
```
kyle@fulltower:~/Source/smartgarden/hub$ cat Cargo.toml
[package]
name = "hub"
version = "0.1.0"
authors = ["Kyle Kloberdanz <kyle.g.kloberdanz@gmail.com>"]
edition = "2018"

[dependencies]
rocket = "0.4.0"
serde = "1.0"
serde_json = "1.0"
serde_derive = "1.0"
rusqlite = "0.18.0"
reqwest = "0.9.15"
chrono = "0.4.6"
time = "0.1"

[dependencies.rocket_contrib]
default-features = false
features = ["json"]
kyle@fulltower:~/Source/smartgarden/hub$ 
```

Build System and Package Management

```
kyle@fulltower:~/Source/
smartgarden/hub$ cargo run
   Compiling autocfg v0.1.2
   Compiling libc v0.2.53
   Compiling semver-parser v0.7.0
   Compiling rand_core v0.4.0
   Compiling version_check v0.1.5
   Compiling proc-macro2 v0.4.29
...
```

```
   Compiling rocket_contrib v0.4.0
   Compiling hub v0.1.0
(/home/kyle/Source/smartgarden/hub)
    Finished dev [unoptimized + debuginfo] target(s) in
42.01s
     Running `target/debug/hub`
fetch_forecast thread active
□ Configured for development.
    => address: 0.0.0.0
    => port: 8080
    => log: debug
    => workers: 32
    => secret key: provided
    => limits: forms = 32KiB
    => keep-alive: 5s
    => tls: disabled
    => [extra] databases: { sqlite_db = { url = "db.sqlite" } }
□  Mounting /:
    => GET / (hello)
    => POST /log Application/json (log)
    => GET /can-i-water/<sensor_id> (can_i_water)
□ Rocket has launched from http://0.0.0.0:8080
```

Build System and Package Management



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.

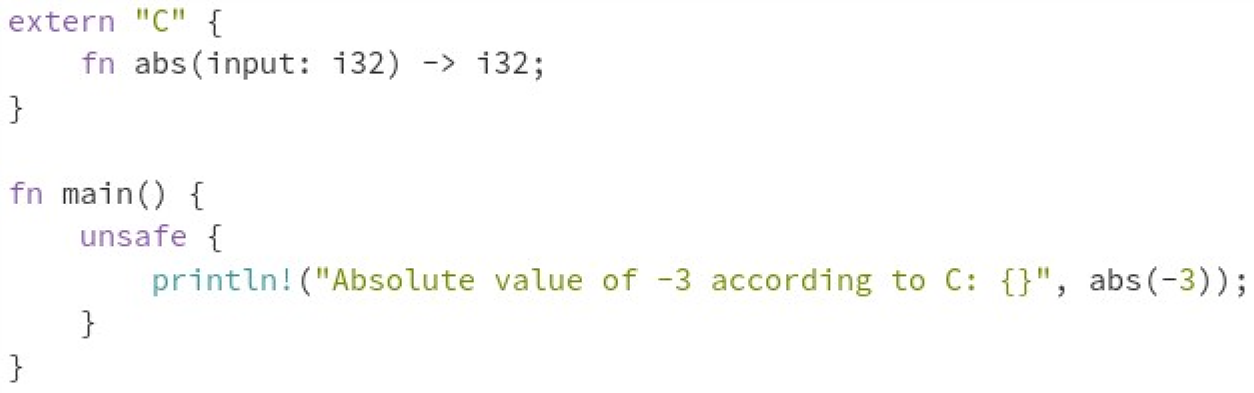
1,060,610,157 Downloads

26,118 Crates in stock

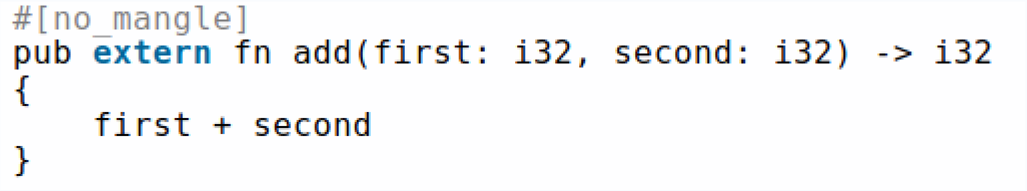
New Crates	Most Downloaded	Just Updated
elgamal-curve25519 (0.1.0)	libc (0.2.55)	rocket-include-handlebars (0.11.5)

C Compatible Application Binary Interface

- **Rust can support the C ABI**
  - To use C in Rust:



```
extern "C" {
    fn abs(input: i32) -> i32;
}

fn main() {
    unsafe {
        println!("Absolute value of -3 according to C: {}", abs(-3));
    }
}
```

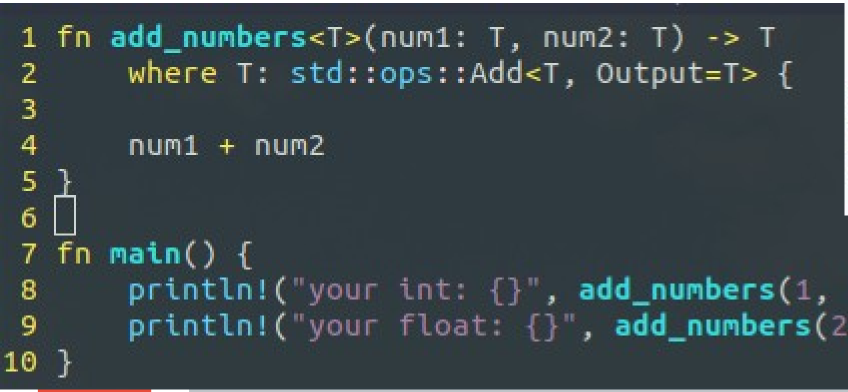


```
#[no_mangle]
pub extern fn add(first: i32, second: i32) -> i32
{
    first + second
}
```

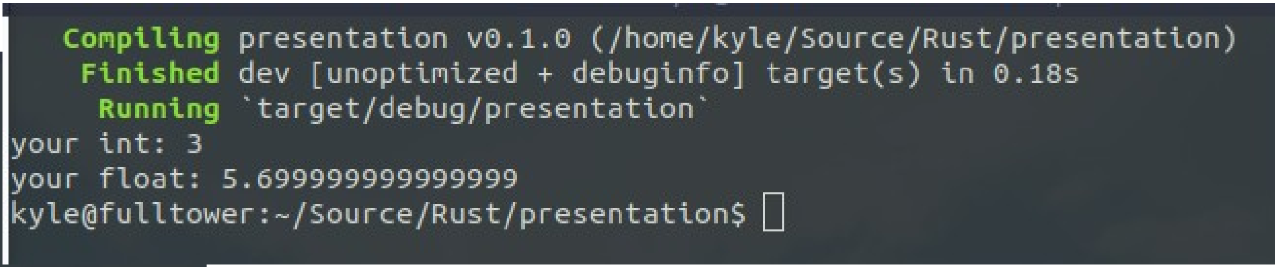
  - To use Rust in C:
    - #[no_mangle] will ensure that the machine code is C ABI compatible
    - Build a .h header
    - Compile as if it was C
  - Tools like bindgen can generate boiler plate for you

Generics

- **Compile time templates**
- **Unlike C++, templates are type checked before expanding**
    - Operations that can be performed on objects are a part of its type, so in order to pass something to the `add_numbers` function, the type signature must match what operations are done on the object within the function.



```rust
fn add_numbers<T>(num1: T, num2: T) -> T
    where T: std::ops::Add<T, Output=T> {

    num1 + num2
}

fn main() {
    println!("your int: {}", add_numbers(1, 2));
    println!("your float: {}", add_numbers(2.3, 3.4));
}
```

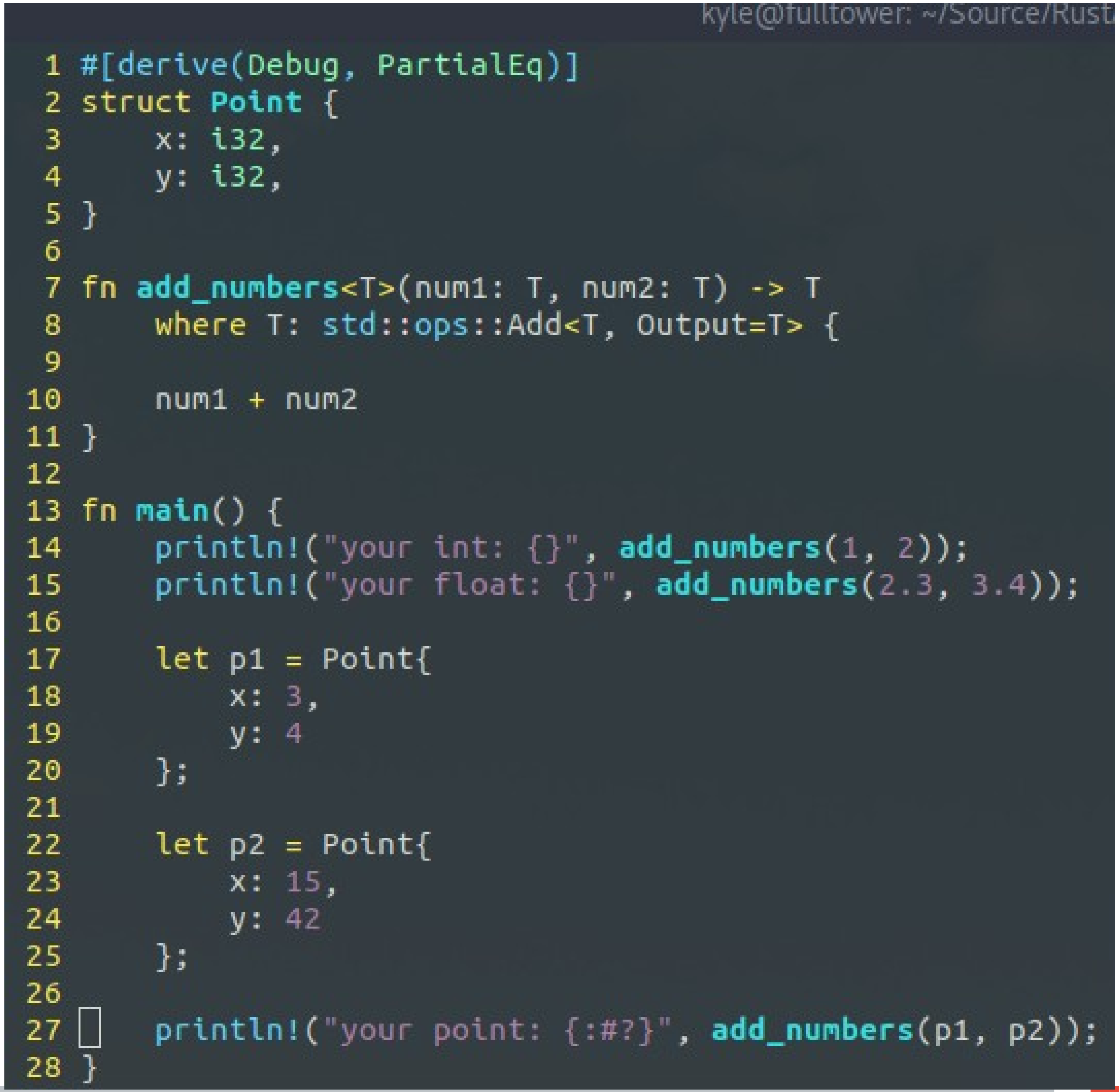


```
   Compiling presentation v0.1.0 (/home/kyle/Source/Rust/presentation)
    Finished dev [unoptimized + debuginfo] target(s) in 0.18s
     Running `target/debug/presentation`
your int: 3
your float: 5.699999999999999
kyle@fulltower:~/Source/Rust/presentation$
```

Generics

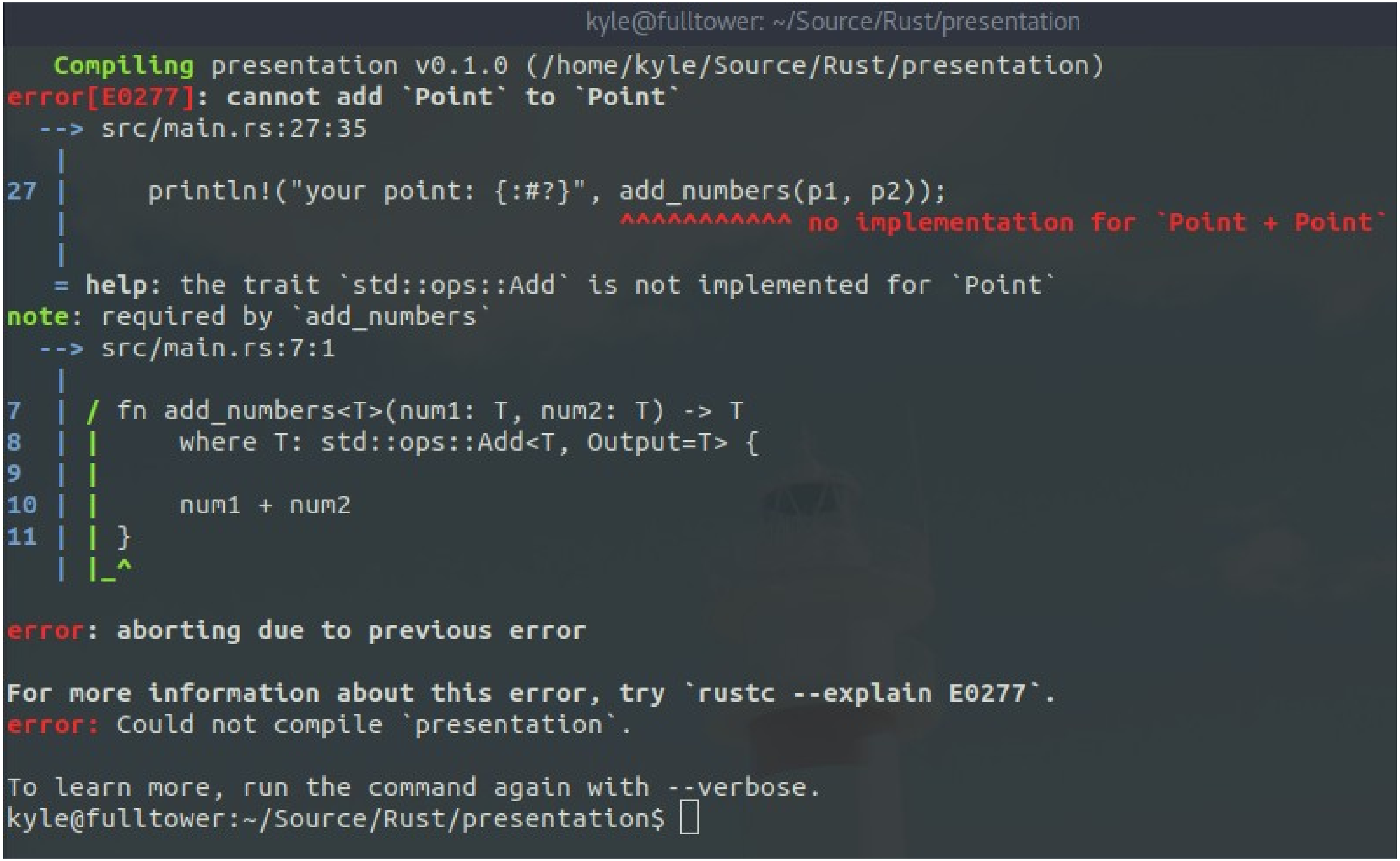
- **What if we want to make our own type work with the `+` operator?**

Generics - Naive Approach



```
#[derive(Debug, PartialEq)]
struct Point {
    x: i32,
    y: i32,
}

fn add_numbers<T>(num1: T, num2: T) -> T
    where T: std::ops::Add<T, Output=T> {

    num1 + num2
}

fn main() {
    println!("your int: {}", add_numbers(1, 2));
    println!("your float: {}", add_numbers(2.3, 3.4));

    let p1 = Point{
        x: 3,
        y: 4
    };

    let p2 = Point{
        x: 15,
        y: 42
    };

    println!("your point: {:#?}", add_numbers(p1, p2));
}
```

Generics - Naive Approach

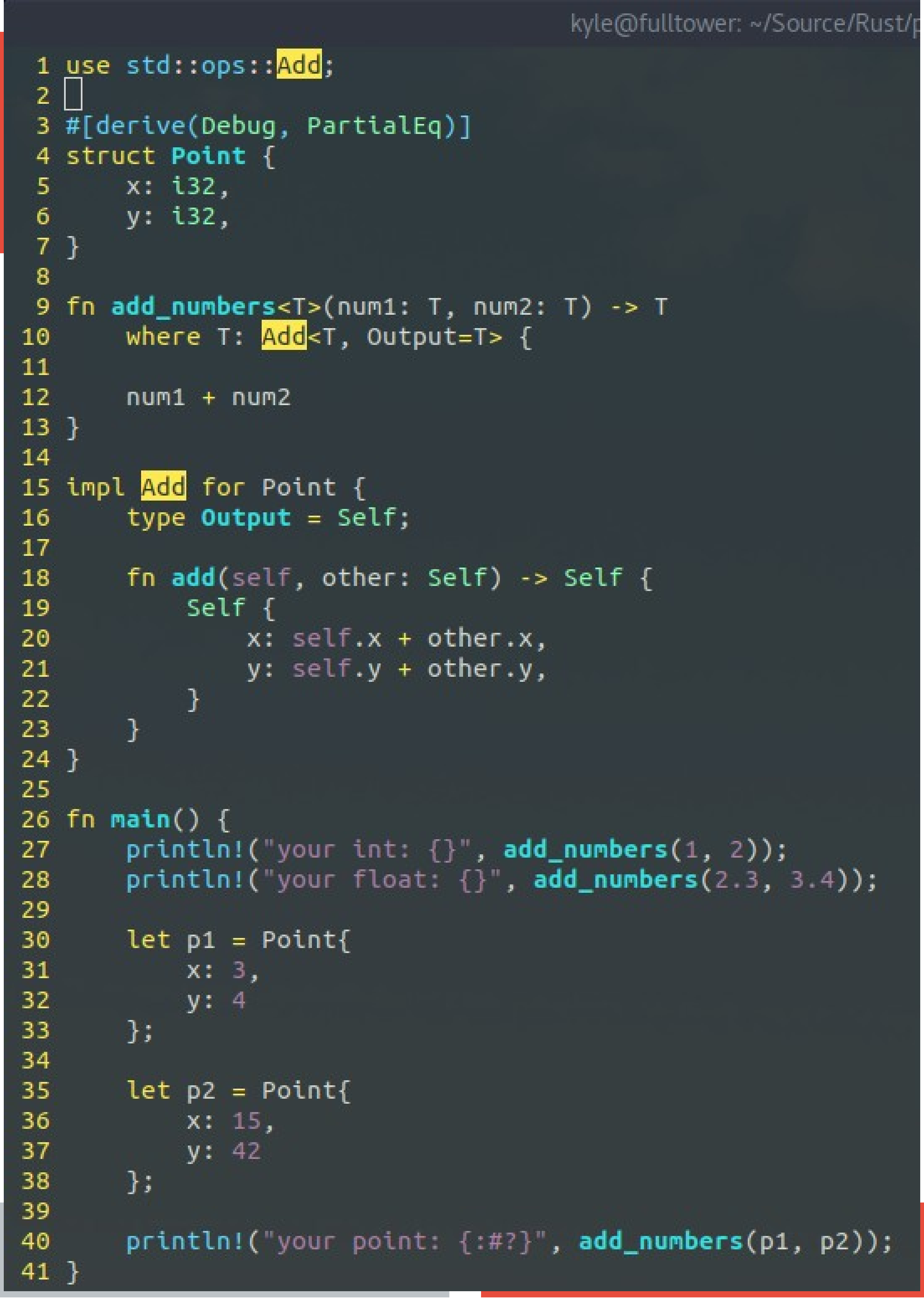


```
   Compiling presentation v0.1.0 (/home/kyle/Source/Rust/presentation)
error[E0277]: cannot add `Point` to `Point`
  --> src/main.rs:27:35
   |
27 |     println!("your point: {:#?}", add_numbers(p1, p2));
   |                                   ^^^^^^^^^^^ no implementation for `Point + Point`
   |
   = help: the trait `std::ops::Add` is not implemented for `Point`
note: required by `add_numbers`
  --> src/main.rs:7:1
   |
7  | / fn add_numbers<T>(num1: T, num2: T) -> T
8  | |     where T: std::ops::Add<T, Output=T> {
9  | |
10 | |     num1 + num2
11 | | }
   | |_^

error: aborting due to previous error

For more information about this error, try `rustc --explain E0277`.
error: Could not compile `presentation`.

To learn more, run the command again with --verbose.
kyle@fulltower:~/Source/Rust/presentation$
```

Traits (Type Classes)

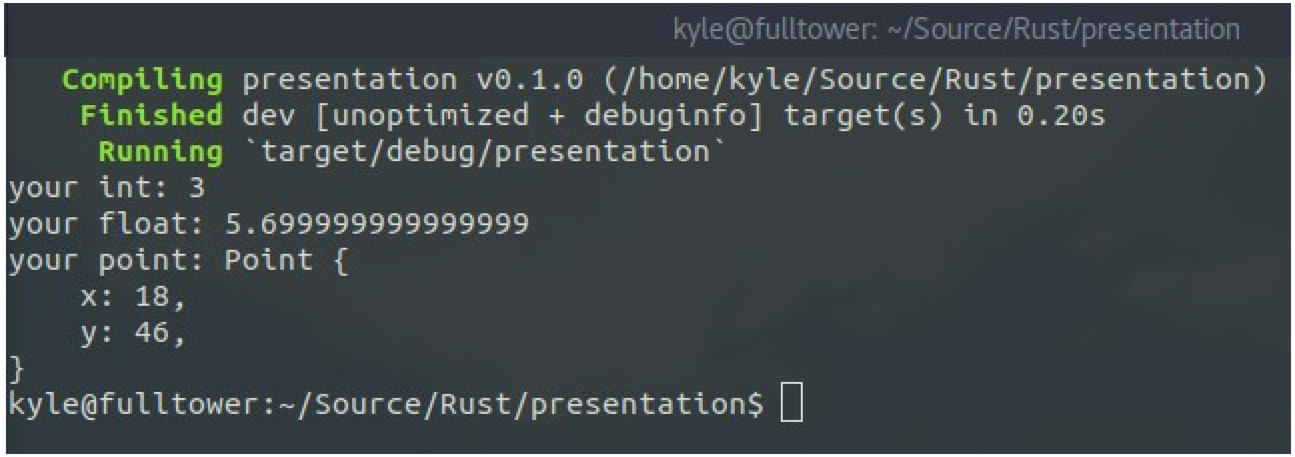
- **If you want to pass a custom object to add_numbers, then you must implement the trait std::ops::Add**

Generics - Fixed



```rust
use std::ops::Add;

#[derive(Debug, PartialEq)]
struct Point {
    x: i32,
    y: i32,
}

fn add_numbers<T>(num1: T, num2: T) -> T
    where T: Add<T, Output=T> {

    num1 + num2
}

impl Add for Point {
    type Output = Self;

    fn add(self, other: Self) -> Self {
        Self {
            x: self.x + other.x,
            y: self.y + other.y,
        }
    }
}

fn main() {
    println!("your int: {}", add_numbers(1, 2));
    println!("your float: {}", add_numbers(2.3, 3.4));

    let p1 = Point{
        x: 3,
        y: 4
    };

    let p2 = Point{
        x: 15,
        y: 42
    };

    println!("your point: {:#?}", add_numbers(p1, p2));
}
```

Generics - Fixed



kyle@fulltower: ~/Source/Rust/presentation

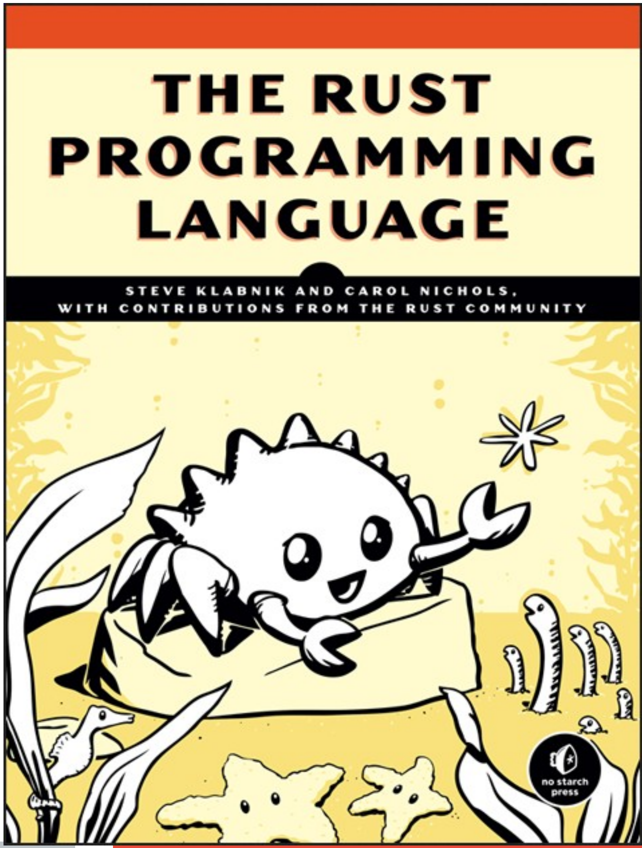
```
   Compiling presentation v0.1.0 (/home/kyle/Source/Rust/presentation)
    Finished dev [unoptimized + debuginfo] target(s) in 0.20s
     Running `target/debug/presentation`
your int: 3
your float: 5.699999999999999
your point: Point {
    x: 18,
    y: 46,
}
kyle@fulltower:~/Source/Rust/presentation$ 
```

How do I get Rust?

```
curl https://sh.rustup.rs -sSf | sh
```

How can I learn more?

- ***The Rust Programming Language*** **is freely available in online form and in print edition from No Starch Press**
  - https://nostarch.com/Rust
  - https://doc.rust-lang.org/book/


THE RUST PROGRAMMING LANGUAGE
STEVE KLABNIK AND CAROL NICHOLS, WITH CONTRIBUTIONS FROM THE RUST COMMUNITY
no starch press