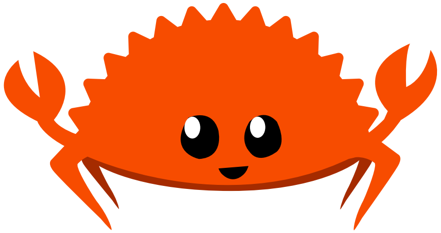
Getting started!

Rustup, Cargo & Hello World



Rustup

Installer for the Rust toolchain

Let’s install Rust!

https://rustup.rs

Afterwards:

The cargo command should be runnable
in your shell

Creating and running our first program

```
$ cargo new hello_world
     Created binary (application) `hello_world` package
$ cd testing
$ cargo run
   Compiling hello_world v0.1.0 (/tmp/testing)
    Finished dev [unoptimized + debuginfo] target(s) in 0.83s
     Running `target/debug/hello_world`
Hello, world!
```

src/main.rs

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

VScode

+ rust-analyzer

The Rust docs are very useful:

https://doc.rust-lang.org/std/index.html

src/main.rs

```rust
fn main() {
    for n in 0..10 {
        let n_is_even = is_even(n);
        if n_is_even {
            println!("{} is even", n);
        } else {
            println!("{} is odd", n);
        }
    }
}

fn is_even(num: i32) -> bool {
    num % 2 == 0
}
```

```
0 is even
1 is odd
2 is even
3 is odd
4 is even
5 is odd
6 is even
7 is odd
8 is even
9 is odd
```

src/main.rs

```rust
fn main() {
    for n in 0..10 {
        let n_is_even = is_even(n);
        if n_is_even {
            println!("{} is even", n);
        } else {
            println!("{} is odd", n);
        }
    }
}

fn is_even(num: i32) -> bool {
    num & 1 == 0
}
```

Let's do a FizzBuzz

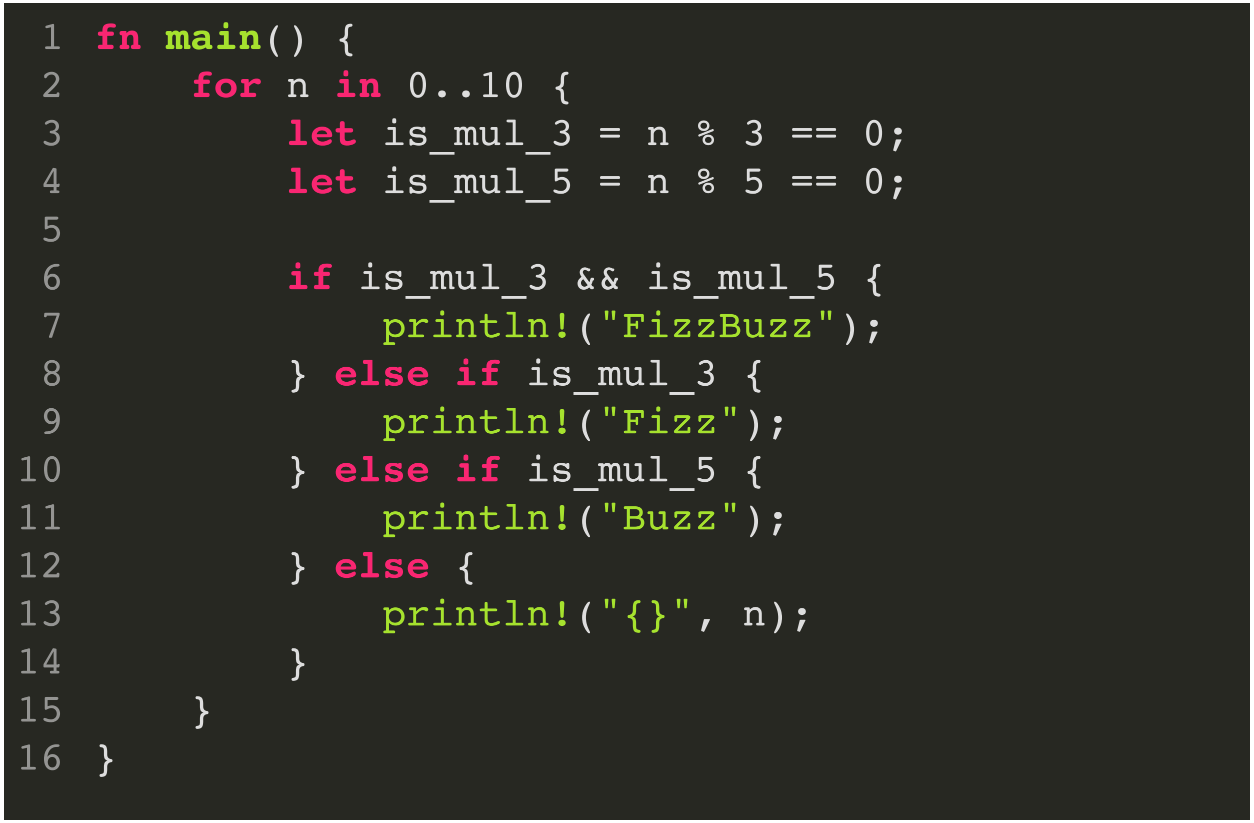
```rust
fn main() {
    for n in 0..10 {
        let n_is_even = is_even(n);
        if n_is_even {
            println!("{} is even", n);
        // } else if <condition> {
        } else {
            println!("{} is odd", n);
        }
    }
}

fn is_even(num: i32) -> bool {
    num % 2 == 0
}
```

If multiple of 3, print Fizz

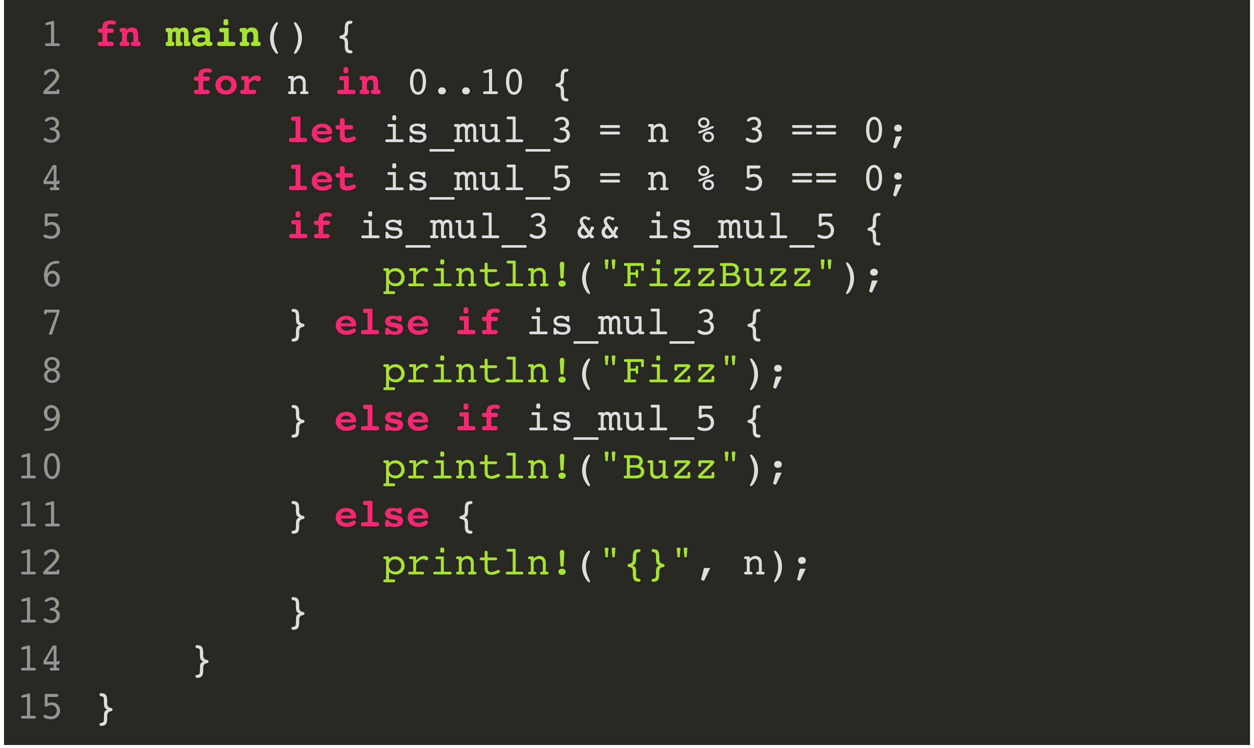
If multiple of 5, print Buzz

If multiple of both print, FizzBuzz instead

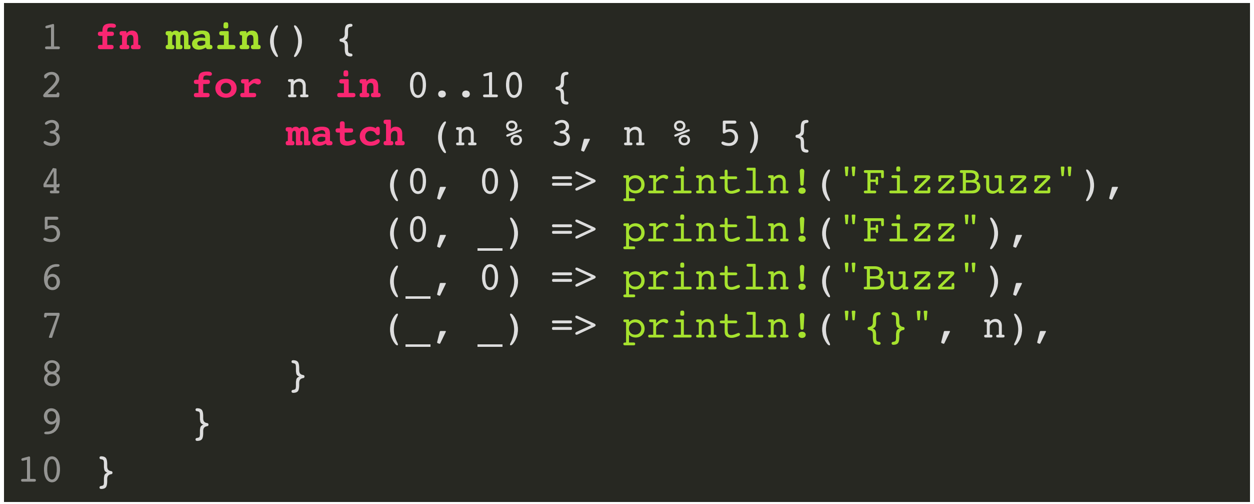
Else, print the number



```rust
fn main() {
    for n in 0..10 {
        let is_mul_3 = n % 3 == 0;
        let is_mul_5 = n % 5 == 0;

        if is_mul_3 && is_mul_5 {
            println!("FizzBuzz");
        } else if is_mul_3 {
            println!("Fizz");
        } else if is_mul_5 {
            println!("Buzz");
        } else {
            println!("{}", n);
        }
    }
}
```



```rust
fn main() {
    for n in 0..10 {
        let is_mul_3 = n % 3 == 0;
        let is_mul_5 = n % 5 == 0;
        if is_mul_3 && is_mul_5 {
            println!("FizzBuzz");
        } else if is_mul_3 {
            println!("Fizz");
        } else if is_mul_5 {
            println!("Buzz");
        } else {
            println!("{}", n);
        }
    }
}
```



```rust
fn main() {
    for n in 0..10 {
        match (n % 3, n % 5) {
            (0, 0) => println!("FizzBuzz"),
            (0, _) => println!("Fizz"),
            (_, 0) => println!("Buzz"),
            (_, _) => println!("{}", n),
        }
    }
}
```

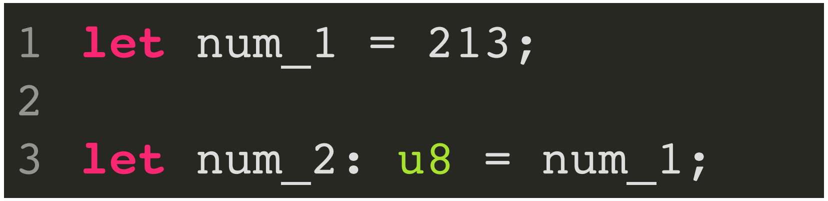
Intro Tour

C#	R
class/struct	struct
Methods	Associated Functions
enum	enum
(tagged union)	enum with fields
Interfaces	traits (ish)
Inheritance	Nope! (traits ish)
Polymorphism	Polymorphism

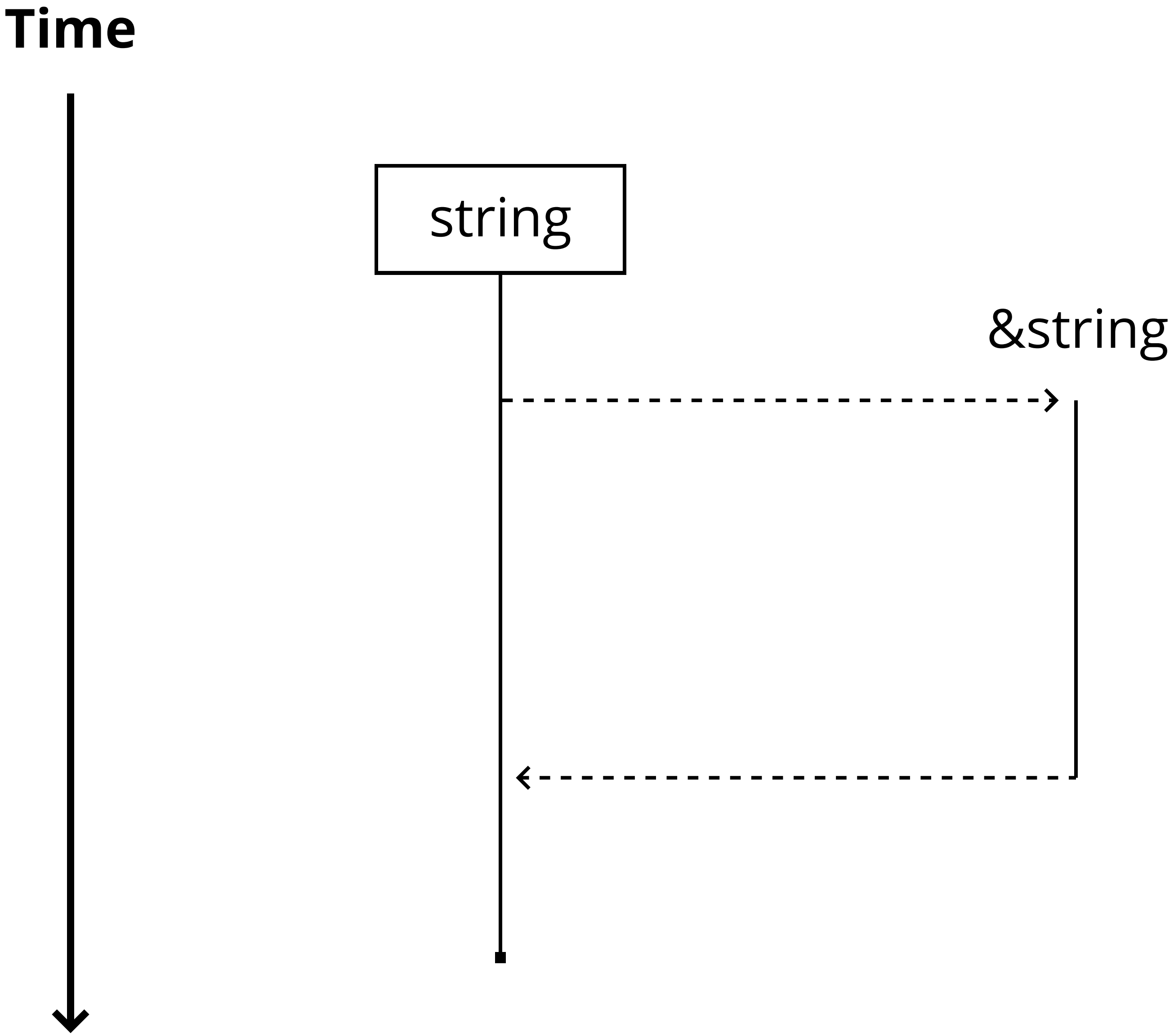
No nulls!

```
fn take_string(string: String) {
        ...
}
```

Type Inference



```
let num_1 = 213;

let num_2: u8 = num_1;
```


Time
string
&string

Traits

Sortable	→	implements Ord trait
Hashable	→	implements Hash trait

my_binary_crate

- root module
  - ui

↓

my_library_crate

- root module
  - api
  - data
  - utils

Basic Types

Numeric primitive types

Explicitly sized

```
i8    / u8
i16   / u16
i32   / u32
i64   / u64
i128  / u128
```

Floating point

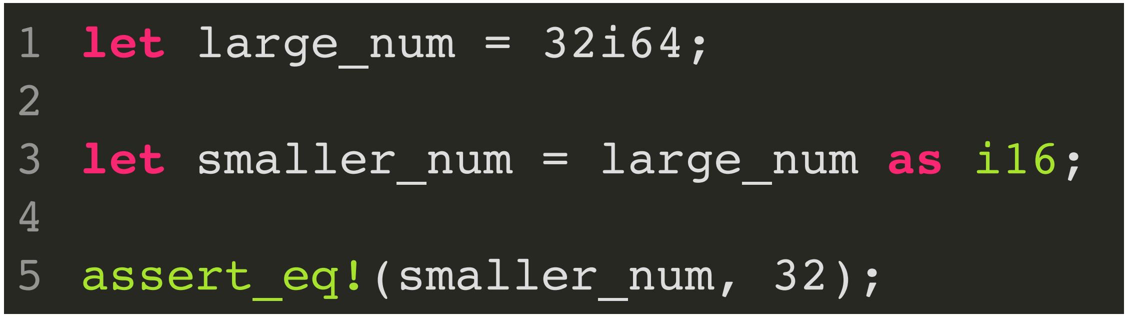
```
f32
f64
```

Pointer-sized

```
isize / usize
```

```
let num: i32 = 13;
```

```
let num = 13i32;
```



```rust
let large_num = 32i64;

let smaller_num = large_num as i16;

assert_eq!(smaller_num, 32);
```

Other primitive types

Boolean

`bool`

Unit type

`()`

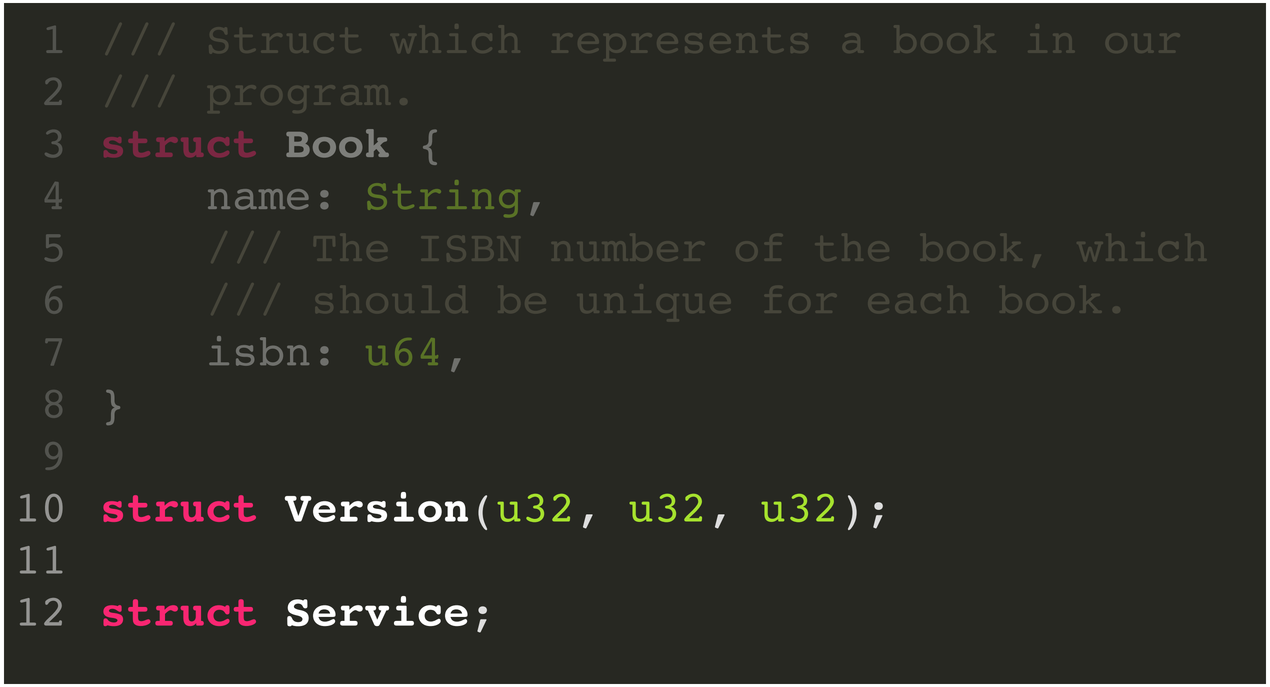
Unicode scalar

`char`

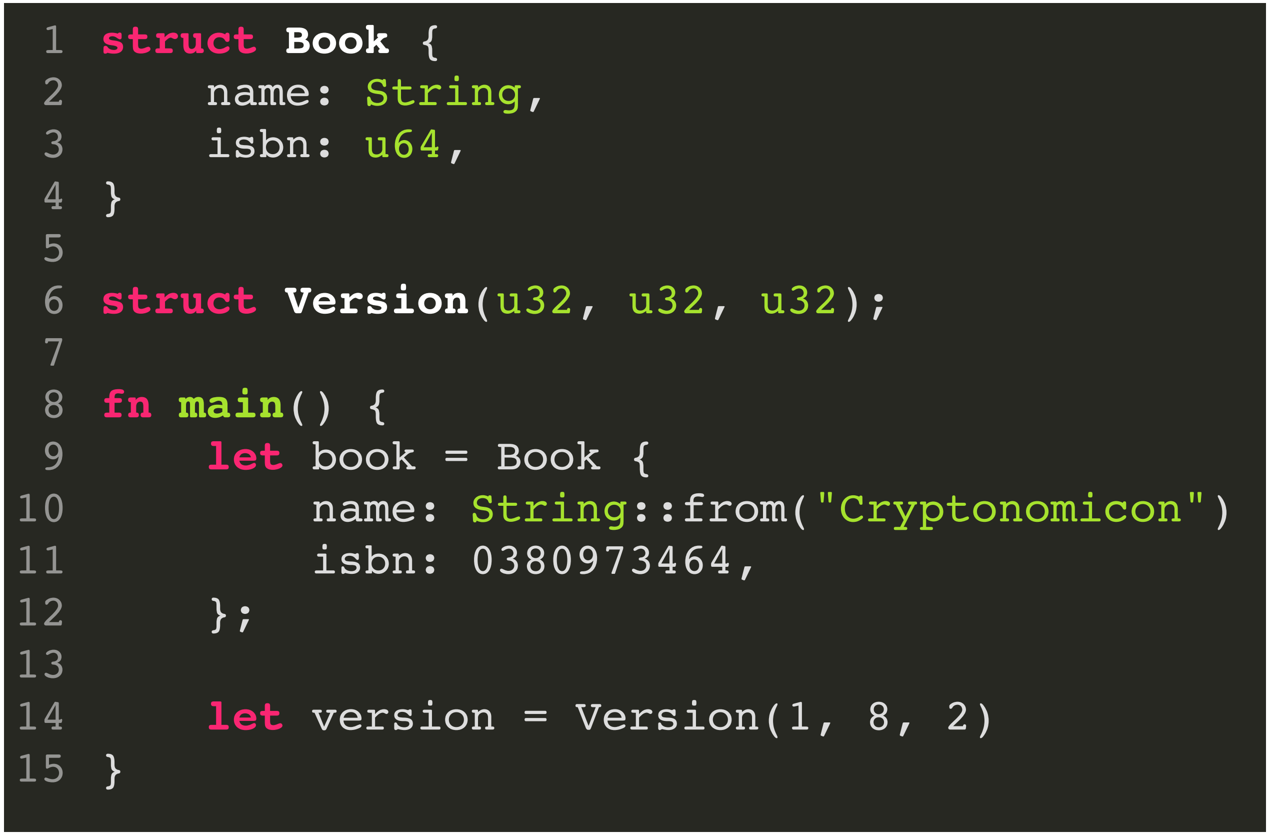
Composite Data Structures

Struct

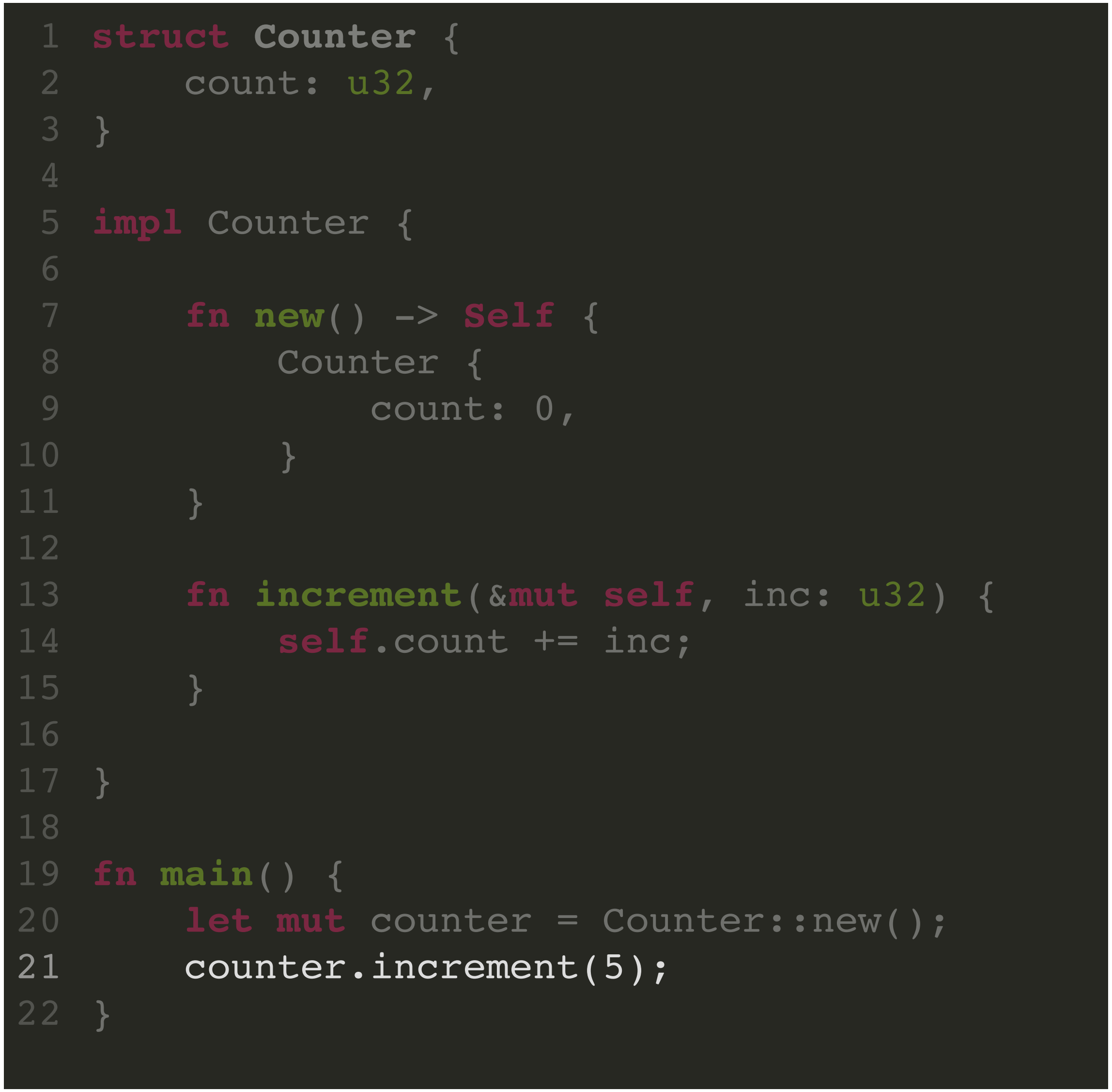
```rust
/// Struct which represents a book in our
/// program.
struct Book {
    name: String,
    /// The ISBN number of the book, which
    /// should be unique for each book.
    isbn: u64,
}
```



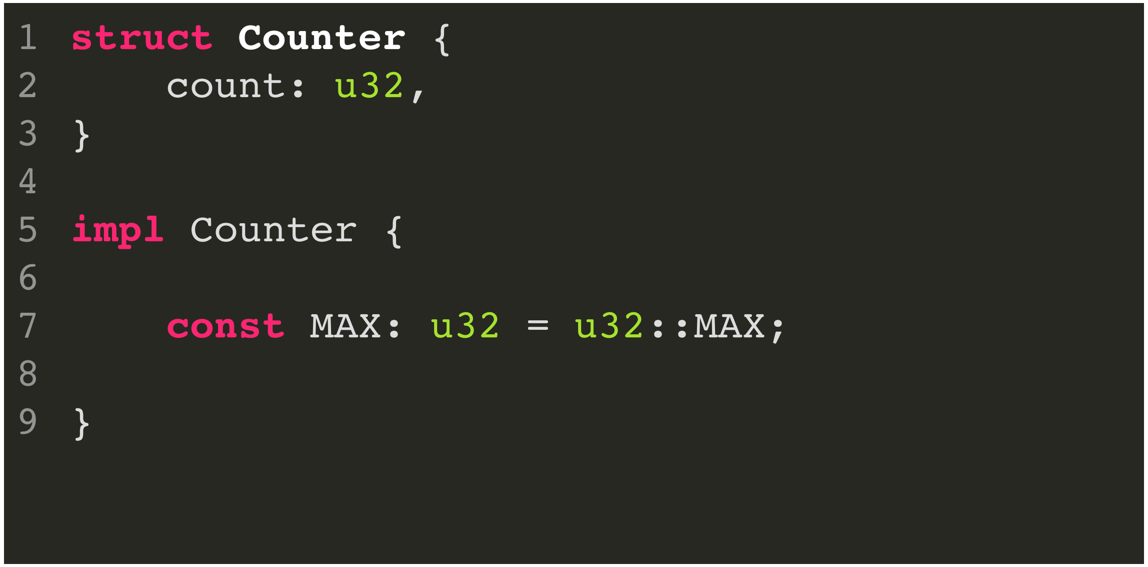
```rust
/// Struct which represents a book in our
/// program.
struct Book {
    name: String,
    /// The ISBN number of the book, which
    /// should be unique for each book.
    isbn: u64,
}

struct Version(u32, u32, u32);

struct Service;
```



```rust
struct Book {
    name: String,
    isbn: u64,
}

struct Version(u32, u32, u32);

fn main() {
    let book = Book {
        name: String::from("Cryptonomicon")
        isbn: 0380973464,
    };

    let version = Version(1, 8, 2)
}
```



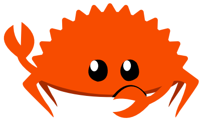
```rust
struct Counter {
    count: u32,
}

impl Counter {

    fn new() -> Self {
        Counter {
            count: 0,
        }
    }

    fn increment(&mut self, inc: u32) {
        self.count += inc;
    }

}

fn main() {
    let mut counter = Counter::new();
    counter.increment(5);
}
```



```
struct Counter {
    count: u32,
}

impl Counter {

    const MAX: u32 = u32::MAX;

}
```

Tuple

```
let a: (u32, u8) = (1, 1);
```



```rust
struct A(u32, u32);
struct B(u32, u32);

fn my_func(a: A) {}

fn main() {
    let val = B(12, 13);
    my_func(val);
}
```

```
error[E0308]: mismatched types
 --> src/main.rs:8:13
  |
8 |     my_func(val);
  |             ^^^ expected struct `A`, found struct `B`
```

```rust
fn my_func(a: (u32, u32)) {}

fn main() {
    let val = (12, 13);
    my_func(val);
}
```

```
()
(u32,)
(u32, u32)
(u32, u32, u32)
(u32, u32, u32, u32)
[...]
```

by Structure	by Identity

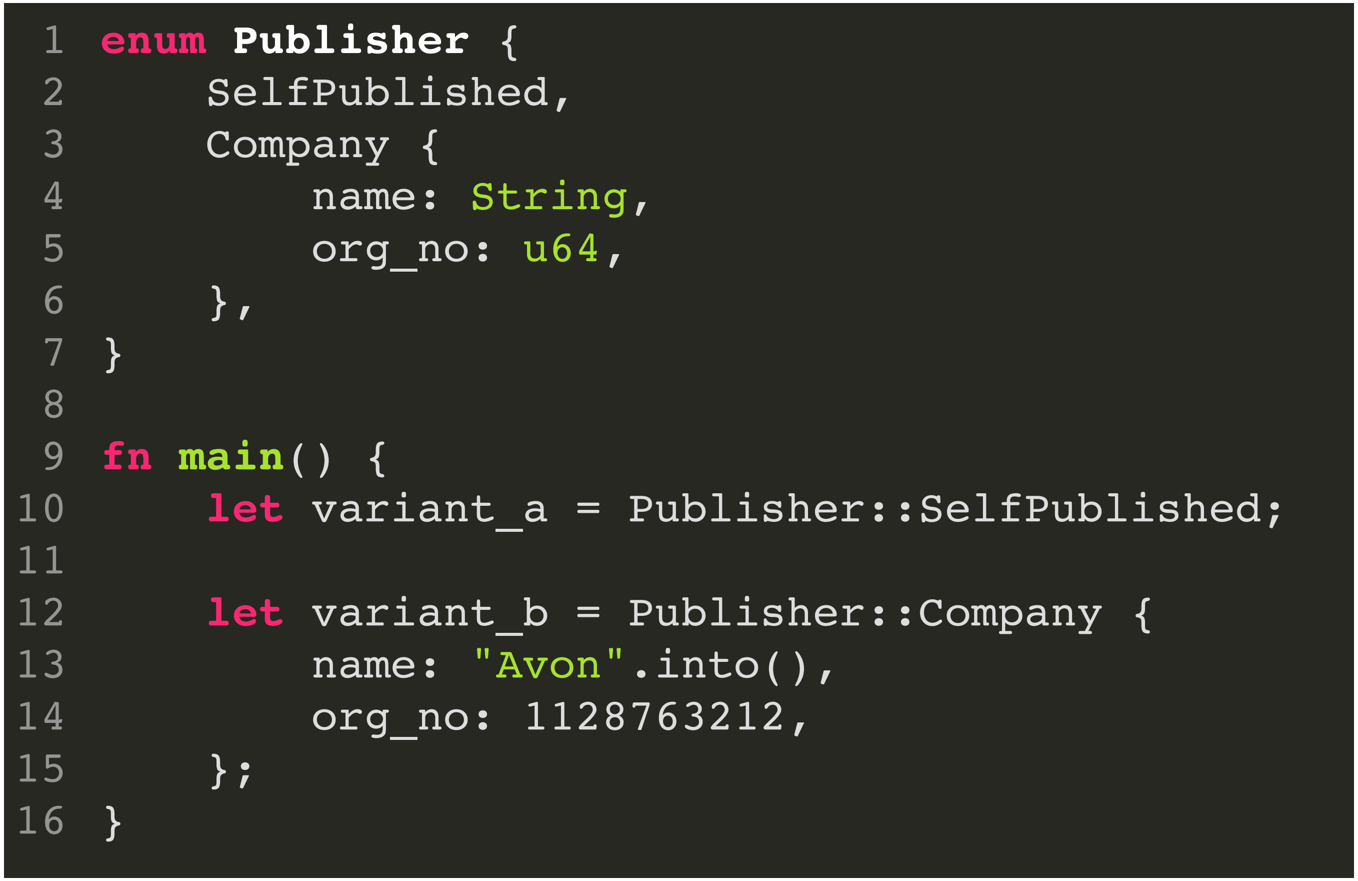
`()`	`struct A;`
	`struct A();`

`(u32, u32)`	`struct A(u32, u32);`

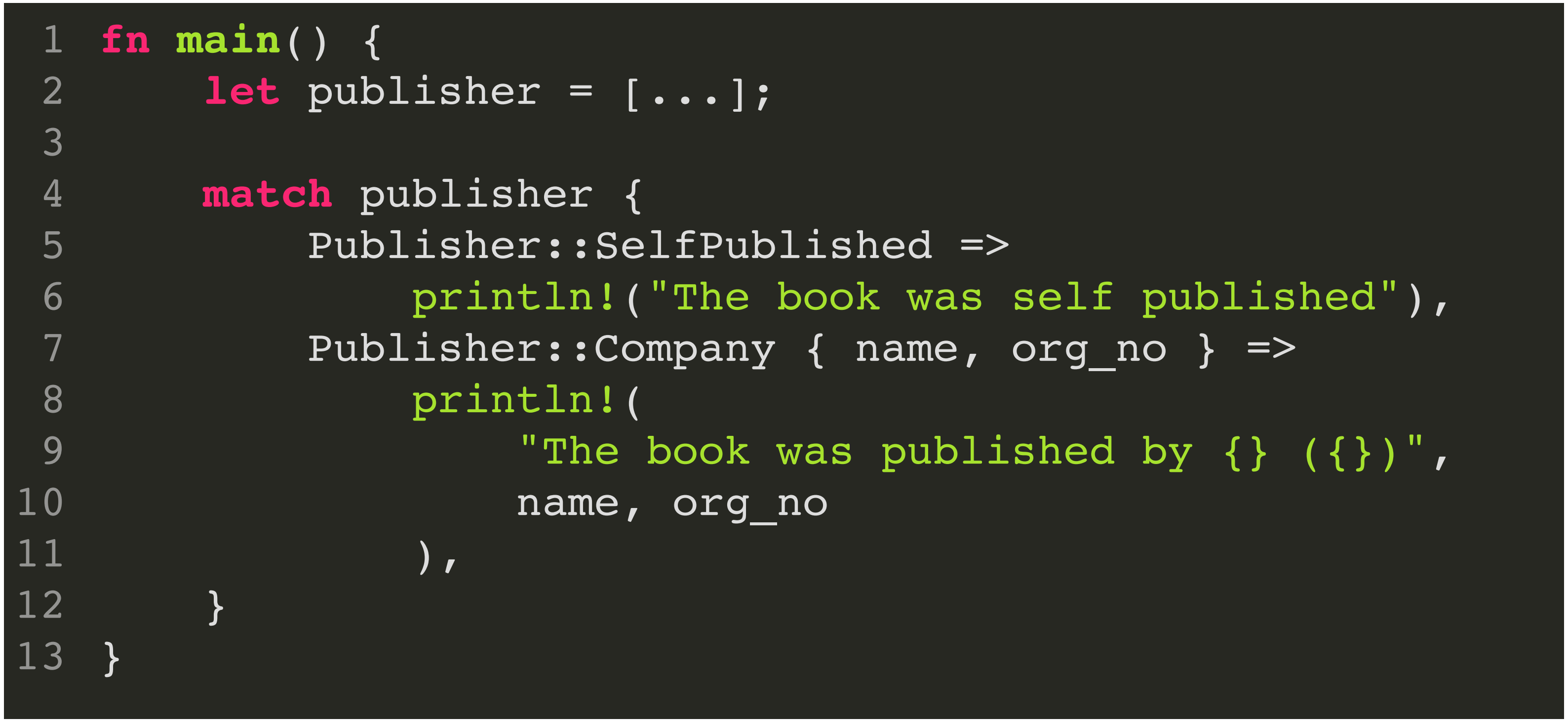
Enum

```
enum BookKind {
    Fantasy,
    SciFi,
    Action,
    Mystery,
}
```

```rust
enum Publisher {
    SelfPublished,
    Company {
        name: String,
        org_no: u64,
    },
}
```



```rust
enum Publisher {
    SelfPublished,
    Company {
        name: String,
        org_no: u64,
    },
}

fn main() {
    let variant_a = Publisher::SelfPublished;

    let variant_b = Publisher::Company {
        name: "Avon".into(),
        org_no: 1128763212,
    };
}
```



```rust
fn main() {
    let publisher = [...];

    match publisher {
        Publisher::SelfPublished =>
            println!("The book was self published"),
        Publisher::Company { name, org_no } =>
            println!(
                "The book was published by {} ({})",
                name, org_no
            ),
    }
}
```

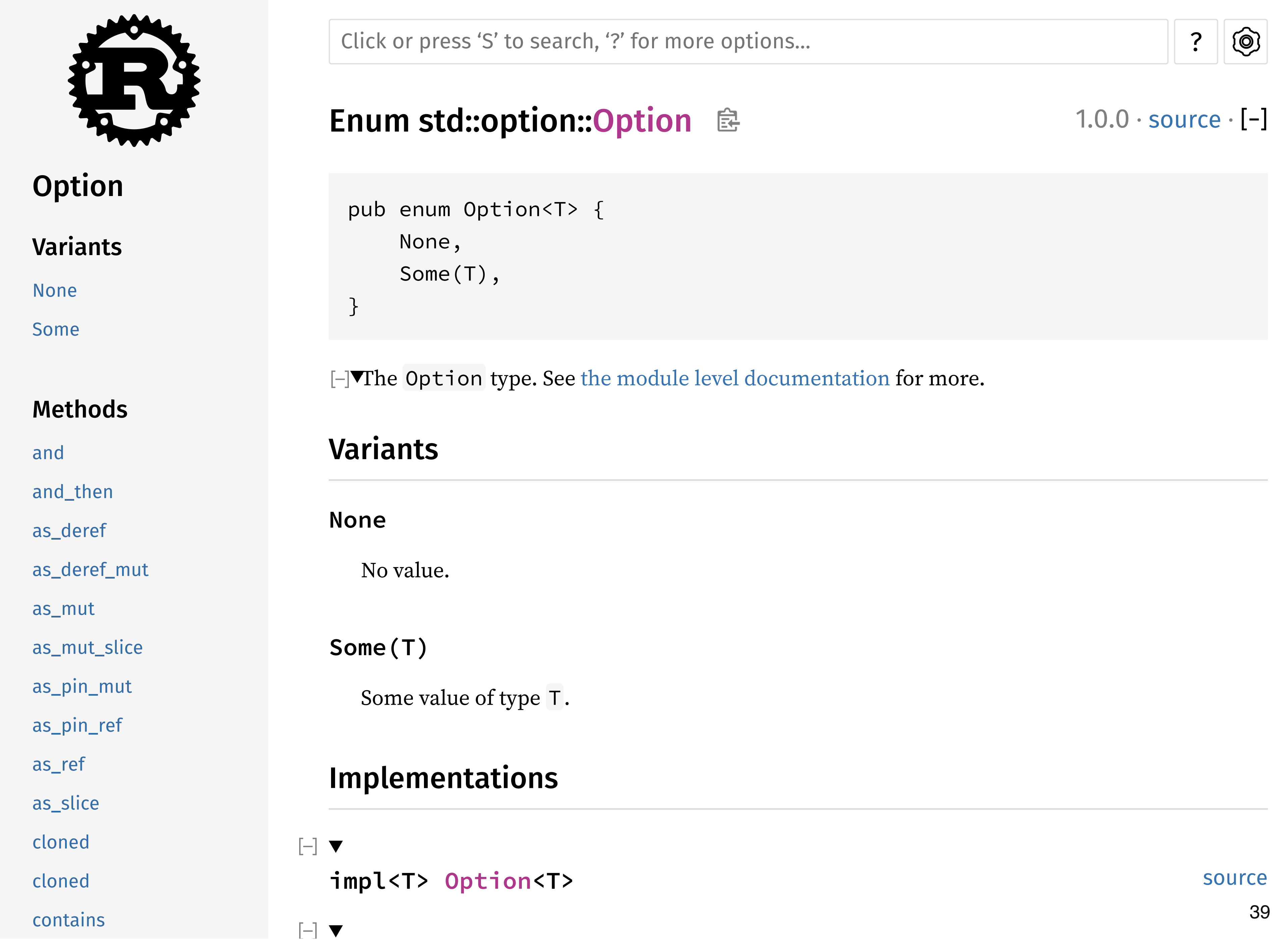
Java:

Class Optional<T>

A container object which may or may not contain a non-`null` value.

If a value is present, `isPresent()` returns `true`. If no value is present, the object is considered *empty* and `isPresent()` returns `false`.

Given we had a way to do type parameters, could something like this be represented well with an `enum`?



Option

Variants

None

Some

Methods

and

and_then

as_deref

as_deref_mut

as_mut

as_mut_slice

as_pin_mut

as_pin_ref

as_ref

as_slice

cloned

cloned

contains

Enum std::option::Option

1.0.0 · source · [−]

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

[−]▼The `Option` type. See the module level documentation for more.

Variants

None

No value.

Some(T)

Some value of type `T`.

Implementations

[−] ▼

`impl<T> Option<T>` source

[−] ▼

```
struct MyStruct<A, B> {
    var_a: A,
    var_b: A,
    var_c: B,
}
```

Derives

```rust
#[derive(Copy, Clone)]
struct Counter {
    num: u32,
    step: u32,
}
```

Derives

```
#[derive(Debug)]
struct Counter {
    num: u32,
    step: u32,
}
```

```
let counter = Counter {
    num: 31,
    step: 1,
};

println!("my struct: {:?}");
```

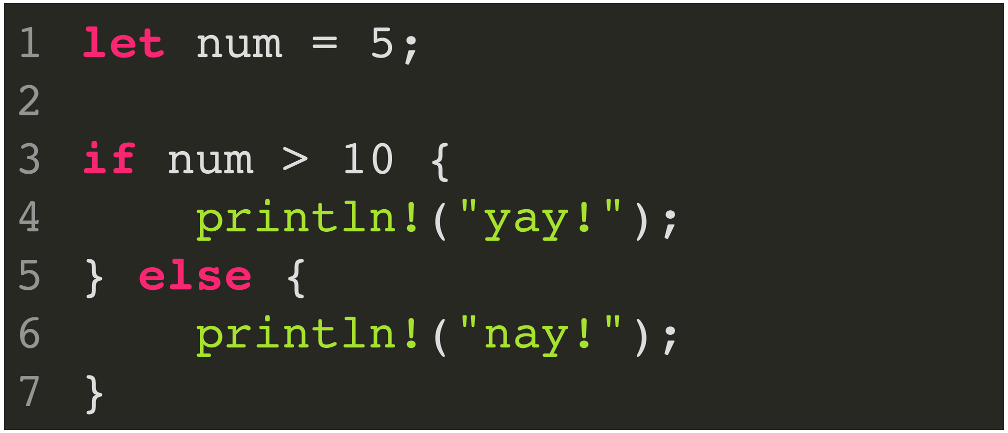
```
my struct: Counter { num: 31, step: 1 }
```

- Ord/PartialOrd
- Eq/PartialEq
- Hash
- Default
- Serialize/Deserialize

Honorary Mention:

Unions

Control Flow

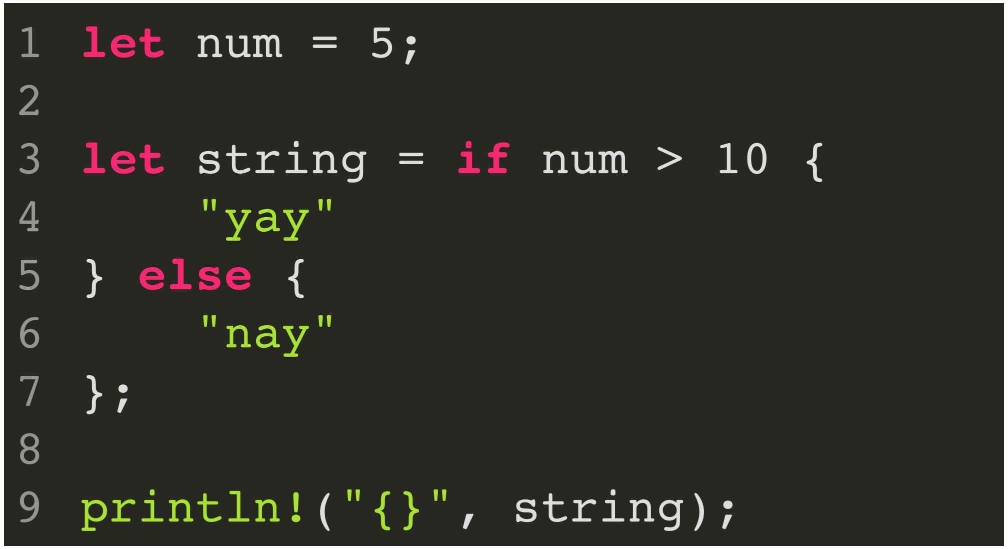


```
let num = 5;

if num > 10 {
    println!("yay!");
} else {
    println!("nay!");
}
```

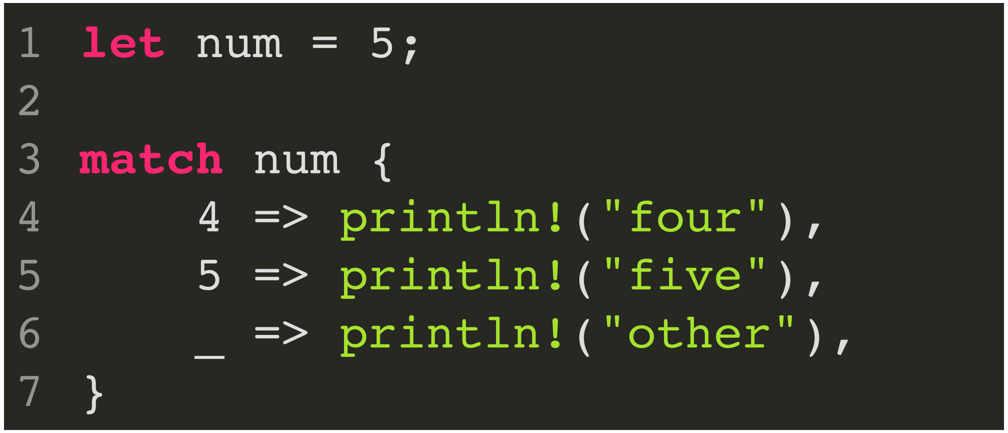
```
let num = 5;

if num {
    println!("yay!");
} else {
    println!("nay!");
}
```

cargo run

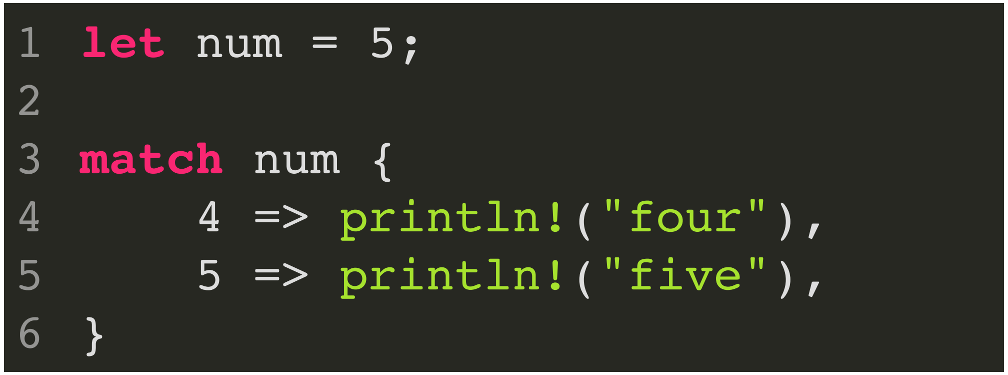
```
error[E0308]: mismatched types
 --> src/main.rs:4:8
  |
4 |     if num {
  |        ^^^ expected `bool`, found integer
```



```rust
let num = 5;

let string = if num > 10 {
    "yay"
} else {
    "nay"
};

println!("{}", string);
```

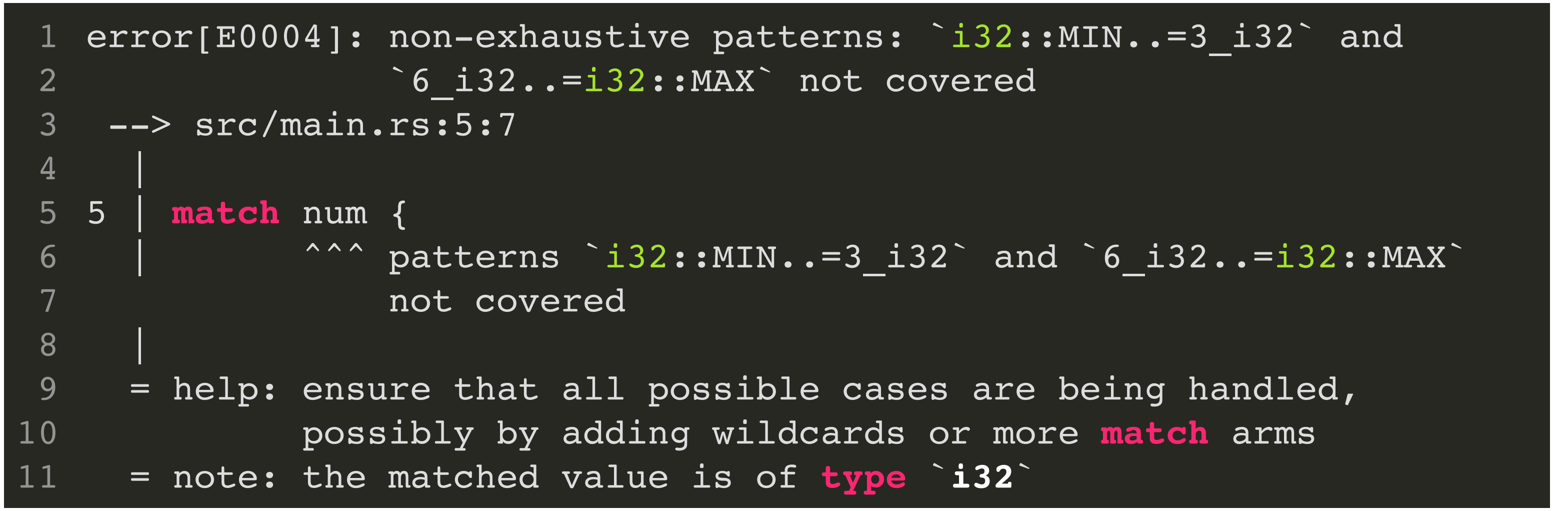


```rust
let num = 5;

match num {
      4 => println!("four"),
      5 => println!("five"),
      _ => println!("other"),
}
```

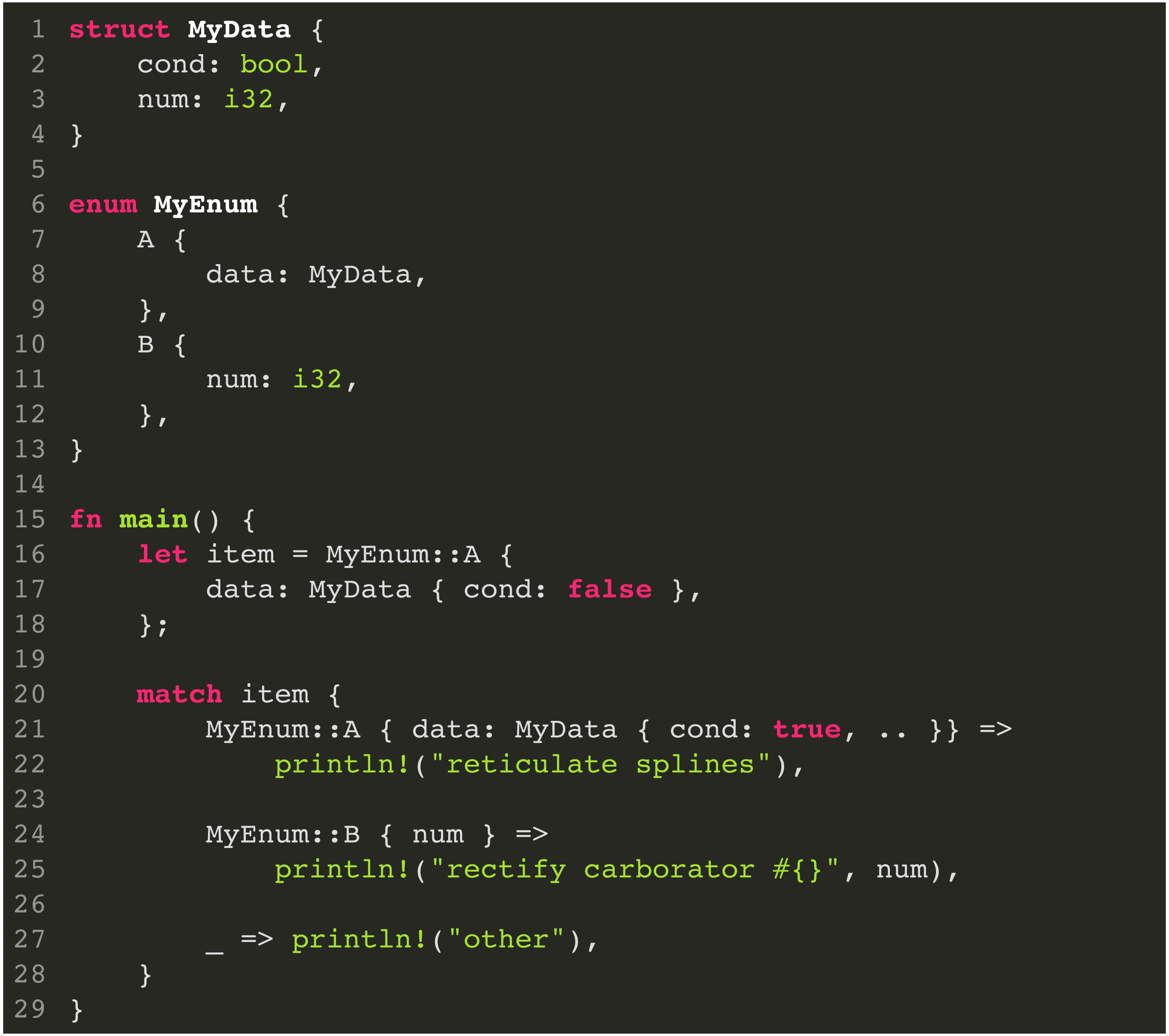


```rust
let num = 5;

match num {
      4 => println!("four"),
      5 => println!("five"),
}
```

cargo run

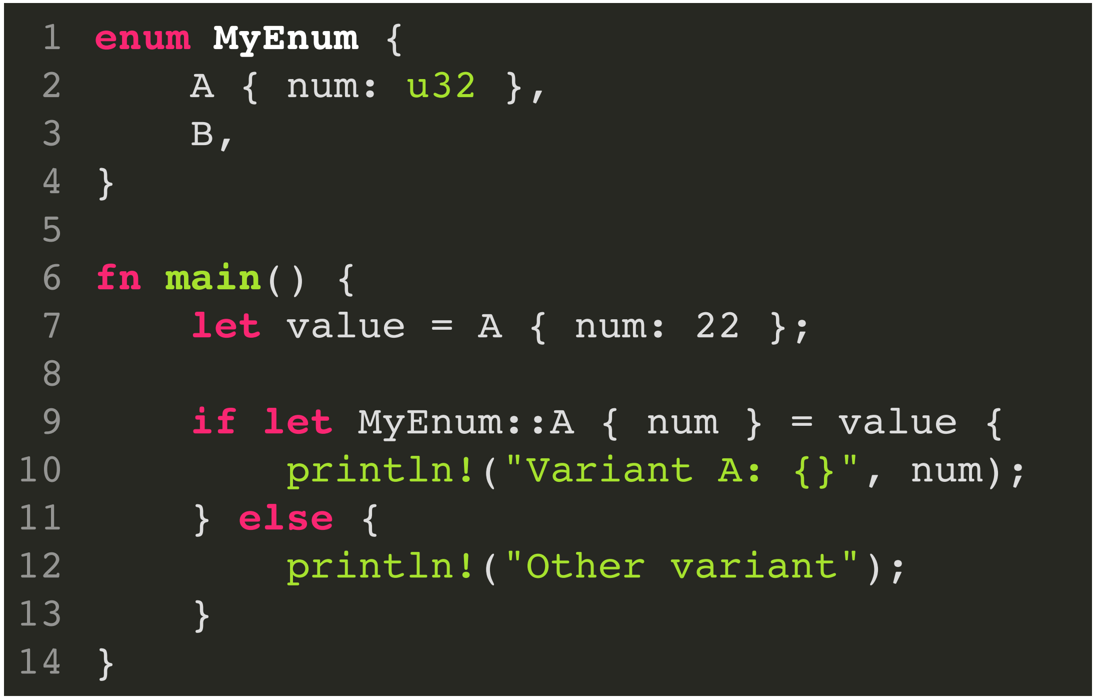


```
error[E0004]: non-exhaustive patterns: `i32::MIN..=3_i32` and
              `6_i32..=i32::MAX` not covered
 --> src/main.rs:5:7
  |
5 | match num {
  |       ^^^ patterns `i32::MIN..=3_i32` and `6_i32..=i32::MAX`
              not covered
  |
  = help: ensure that all possible cases are being handled,
          possibly by adding wildcards or more match arms
  = note: the matched value is of type `i32`
```



```
struct MyData {
    cond: bool,
    num: i32,
}

enum MyEnum {
    A {
        data: MyData,
    },
    B {
        num: i32,
    },
}

fn main() {
    let item = MyEnum::A {
        data: MyData { cond: false },
    };

    match item {
        MyEnum::A { data: MyData { cond: true, .. }} =>
            println!("reticulate splines"),

        MyEnum::B { num } =>
            println!("rectify carborator #{}", num),

        _ => println!("other"),
    }
}
```

Boolean Conditions	Pattern Matching
	`match`
`if`	`if let`
`while`	`while let`



```rust
enum MyEnum {
    A { num: u32 },
    B,
}

fn main() {
    let value = A { num: 22 };

    if let MyEnum::A { num } = value {
        println!("Variant A: {}", num);
    } else {
        println!("Other variant");
    }
}
```

```rust
while source.should_continue() {
    println!("continuing");
}
```

```rust
while let SourceState::Item(item) = source.get_item() {
    println!("has item");
}
```

```
let break_value = loop {
    if let Some(val) = get_item() {
        break val;
    }
};
```

- Loops forever (until break'd)
- Same as `while true`

```
for <item> in <iterator> {
    <body>
}
```

- Works with Iterators
- For now, know it works with
  - lists (vectors)
  - ranges
  - more

Option and Panics

Option enum

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

- std::option::Option - included in global namespace
    - Option, Some(T), None
- ignoring Option gives compiler warning

1. Match on Option

```
fn maybe_get_number() -> Option<u32> { [...] }

match maybe_get_number() {
    Some(num) => println!("Got number: {}", num),
    None => println!("Didn't get anything!"),
}
```

2. Propagate Option using ? operator

```rust
fn maybe_get_number() -> Option<u32> { [...] }

fn my_fun() -> Option<String> {
    let num = maybe_get_number()?;
    Ok(format!("Got number: {}", num))
}
```

3. Panic on None using unwrap

```rust
fn maybe_get_number() -> Option<u32> { [...] }

let num = maybe_get_number().unwrap();
println!("Got number: {}", num);
```

Panic

```rust
fn main() {
    panic!("something bad happened");
}
```

```
thread 'main' panicked at 'what is even going on', src/main.rs:2:5
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```

Modules and Visibility

src/main.rs

```rust
mod a_module {

  pub fn a_function() {}

}

use a_module::a_function;

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

src/a_module.rs

```rust
pub fn a_function() {}
```

src/main.rs

```rust
mod a_module;

use a_module::a_function;

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

src/a_module/mod.rs

```rust
pub fn a_function() {}
```

src/main.rs

```rust
mod a_module;

use a_module::a_function;

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

src/a_module/nested.rs

```rust
pub fn a_function() {}
```

src/a_module/mod.rs

```rust
pub mod nested;
```

src/lib.rs

```rust
mod a_module;

use a_module::nested::a_function;
```

src/lib.rs

```rust
pub struct Counter {
    [...]
}
```

Cargo.toml

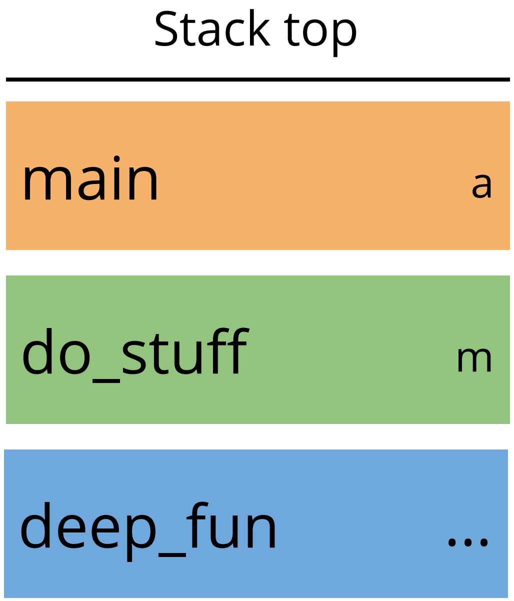
```toml
[dependencies]

# Depend on a crate on crates.io by version
rand = "0.8.4"

# Depend on a crate in a git repository
rand = { git = "https://github.com/rust-random/rand.git" }

# Depend on a local crate, by path
my_crate = { path = "../my_crate" }
```

Memory

Stack

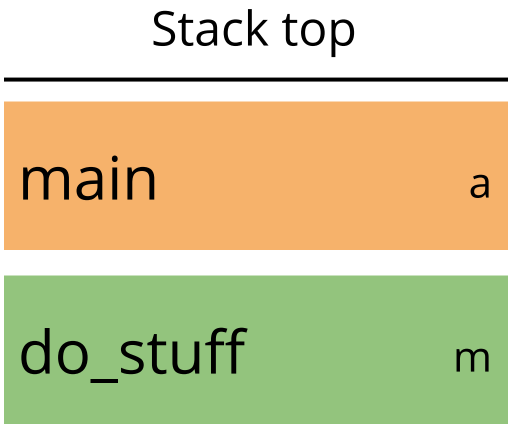
Heap

Stack



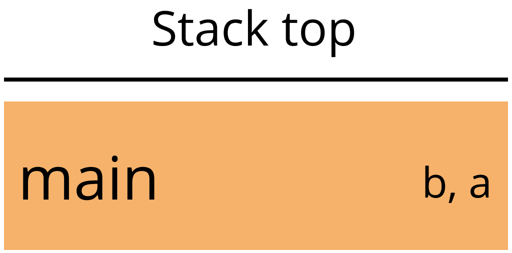
```
fn main() {
    let a = 1;
    let b = do_stuff(a);
}

fn do_stuff(m: u32) {
    [...]
}
```

Stack



```
fn main() {
    let a = 1;
    let b = do_stuff(a);
}

fn do_stuff(m: u32) {
    [...]
}
```

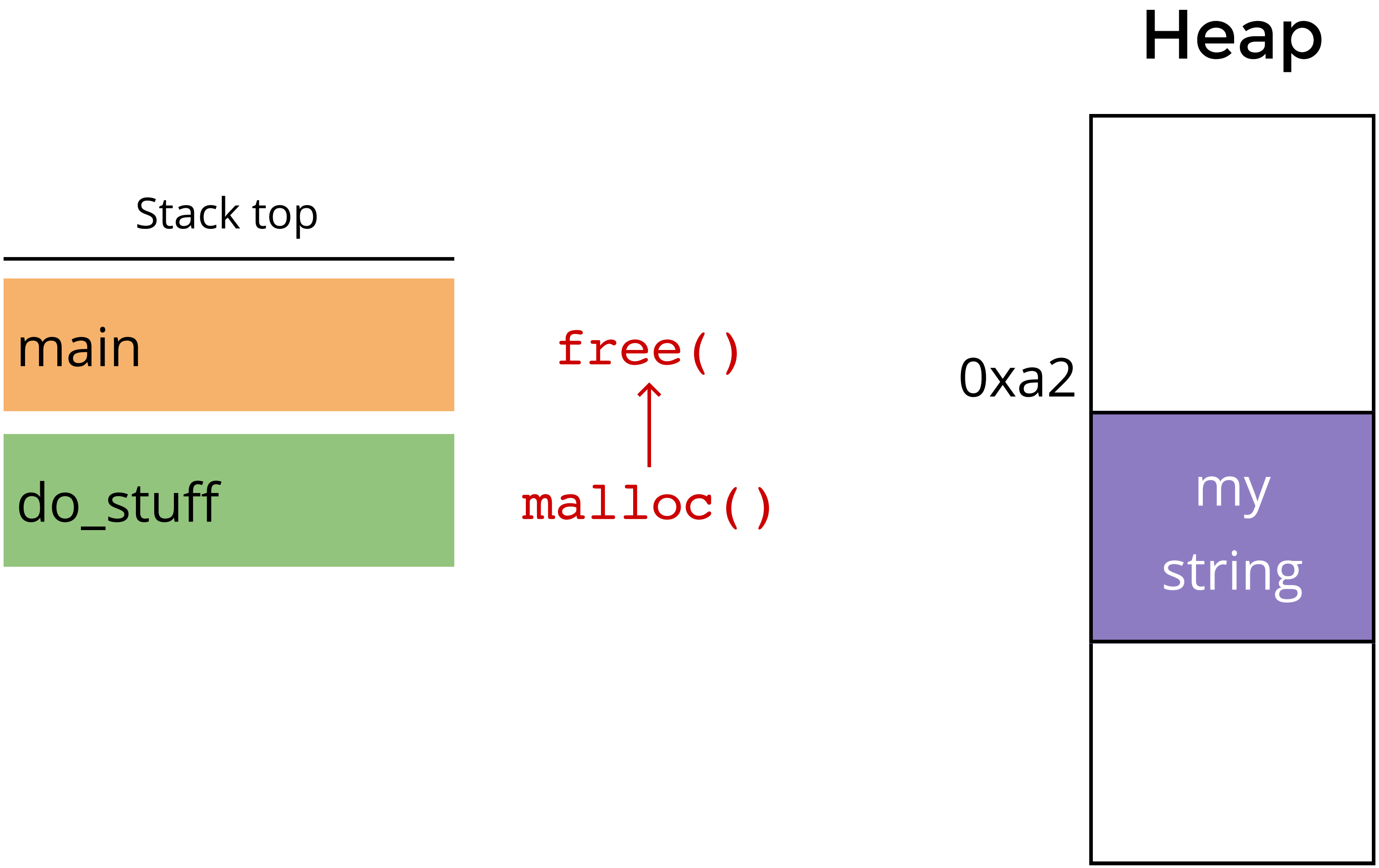
Stack



```rust
fn main() {
    let a = 1;
    let b = do_stuff(a);
}

fn do_stuff(m: u32) {
    [...]
}
```

Stack

Heap



Heap

Stack top

main

do_stuff

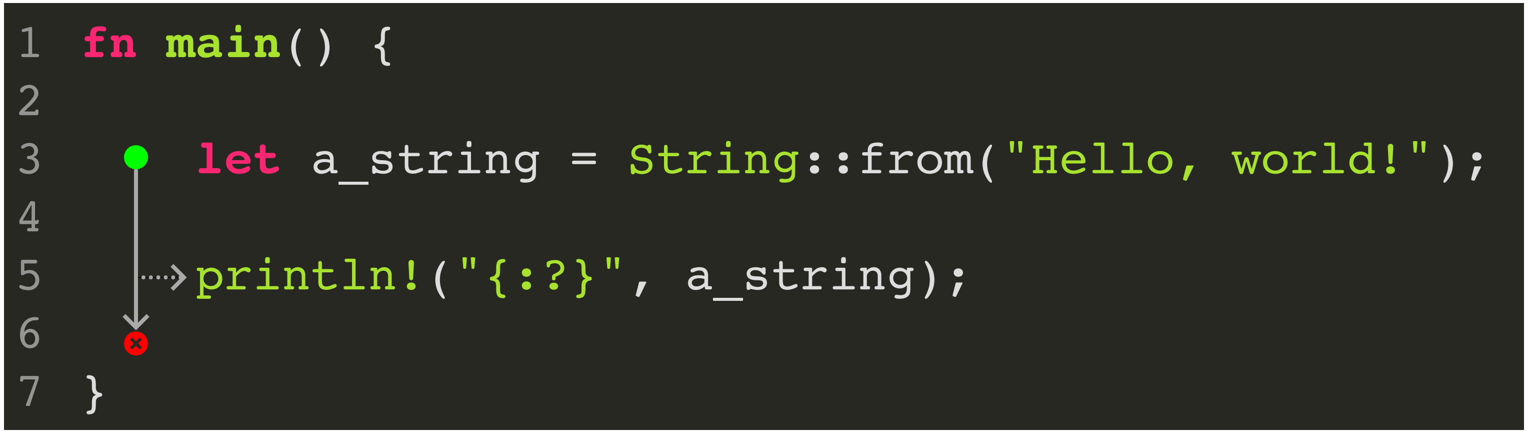
free()

malloc()

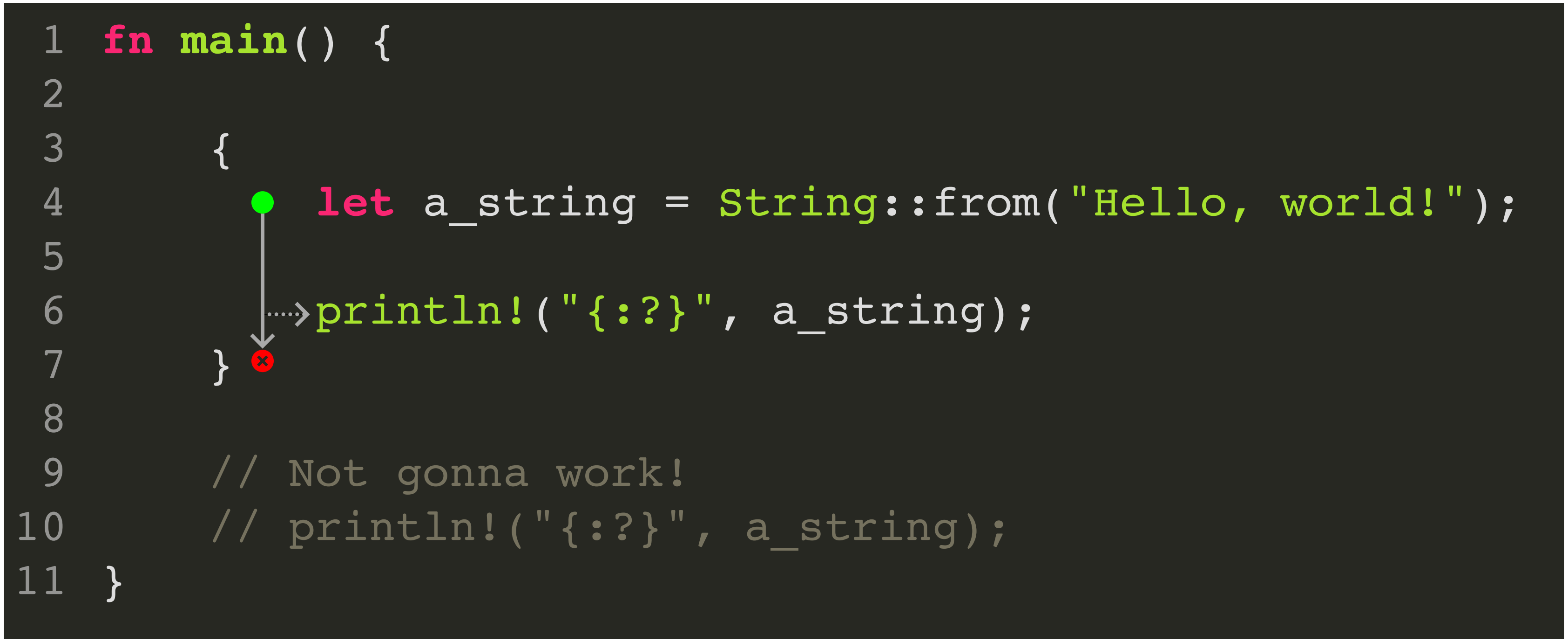
0xa2

my string

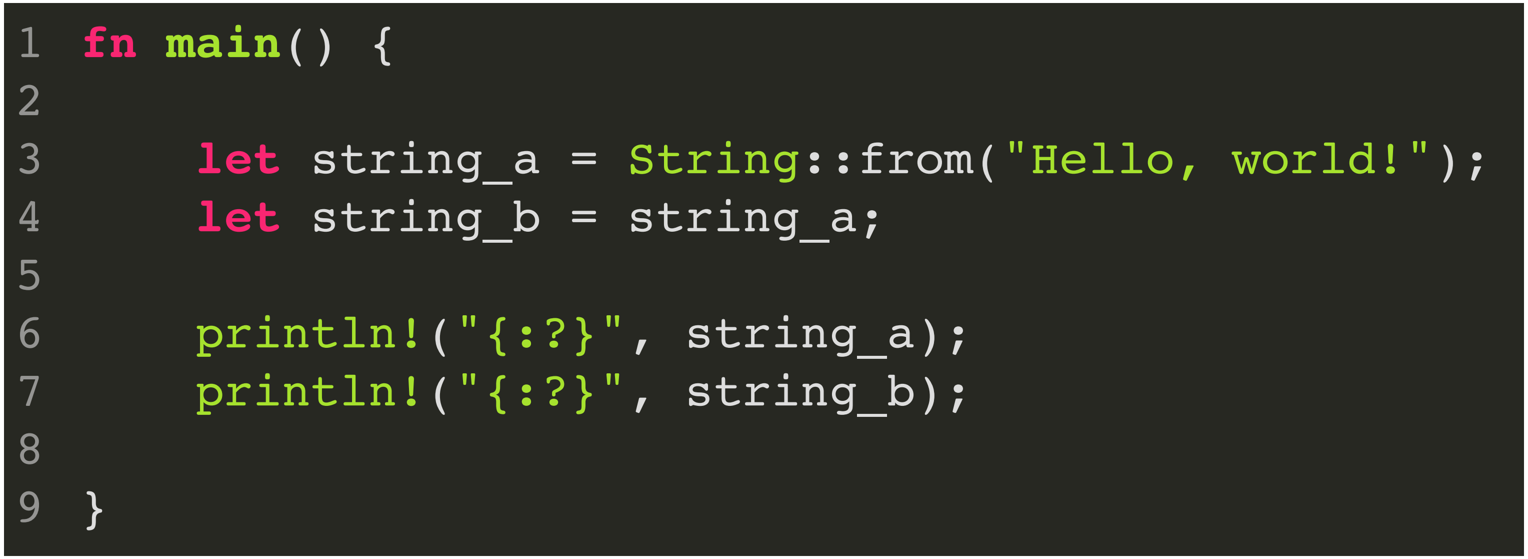
Ownership



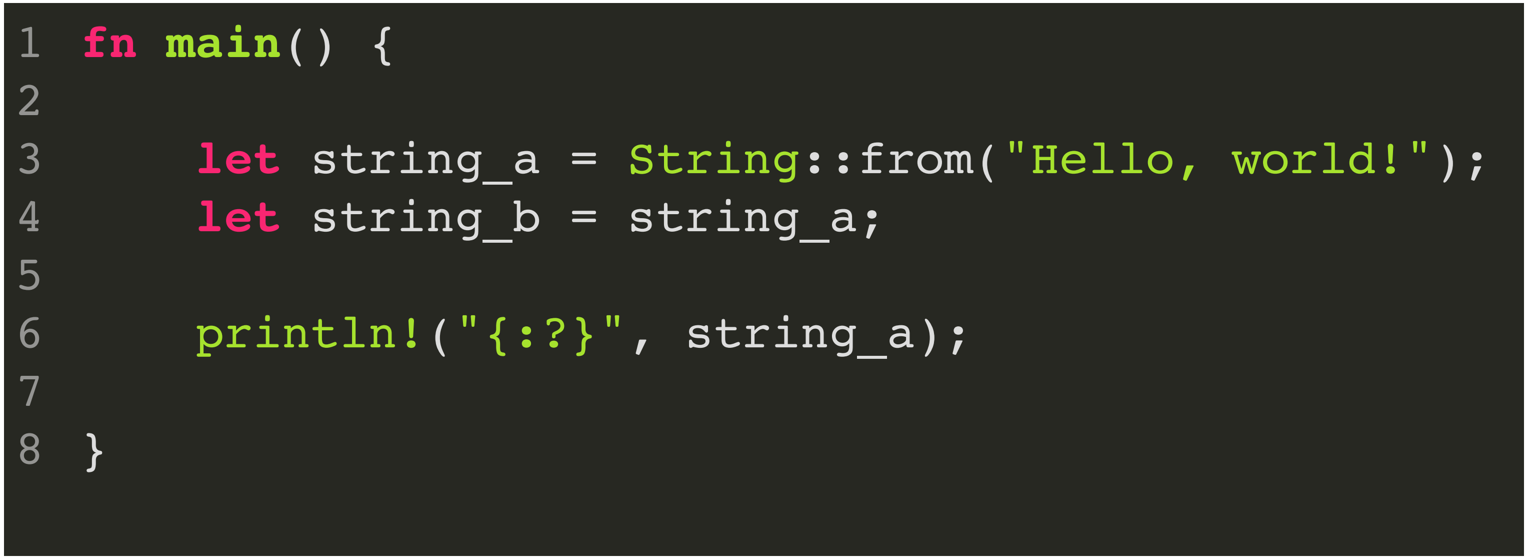
```rust
fn main() {

    let a_string = String::from("Hello, world!");

    println!("{:?}", a_string);

}
```



```rust
fn main() {

    {
        let a_string = String::from("Hello, world!");

        println!("{:?}", a_string);
    }

    // Not gonna work!
    // println!("{:?}", a_string);
}
```

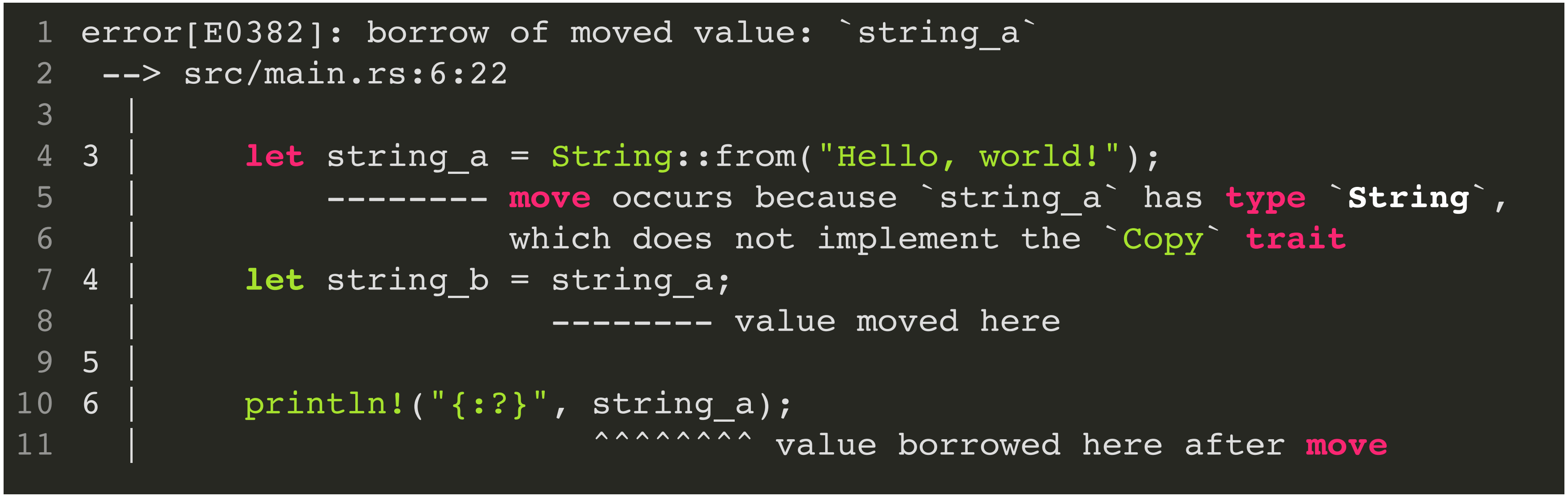


```rust
fn main() {

    let string_a = String::from("Hello, world!");
    let string_b = string_a;

    println!("{:?}", string_a);
    println!("{:?}", string_b);

}
```



```rust
fn main() {

    let string_a = String::from("Hello, world!");
    let string_b = string_a;

    println!("{:?}", string_a);

}
```

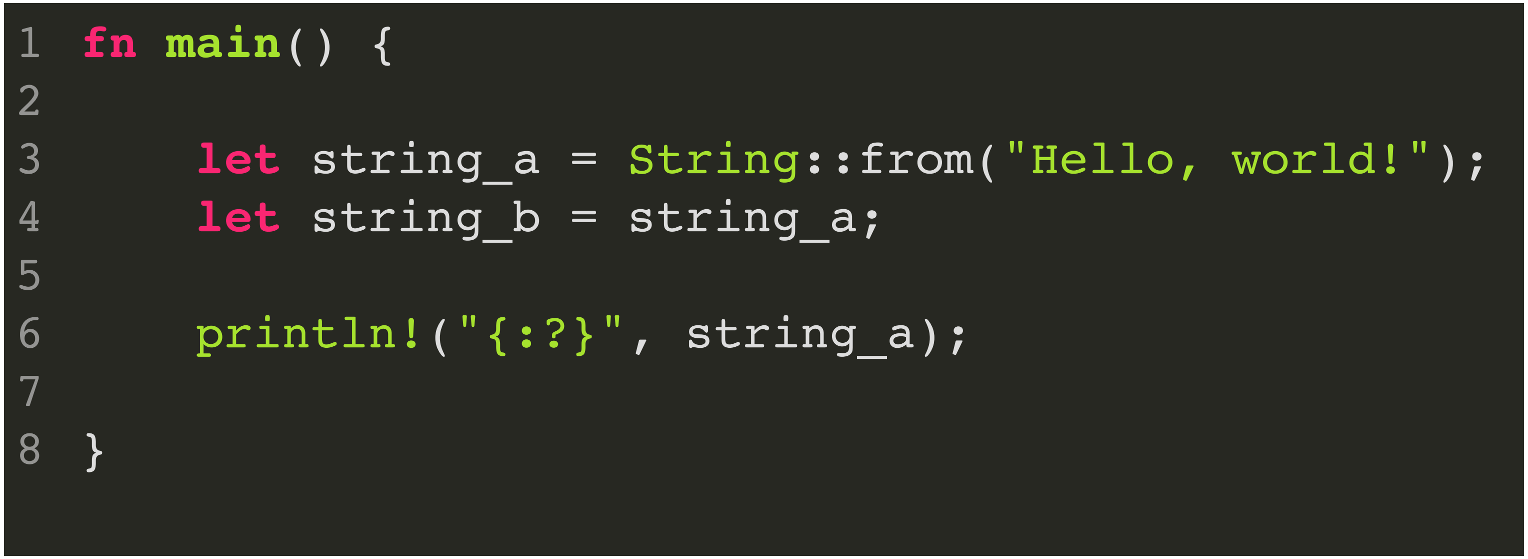
cargo run



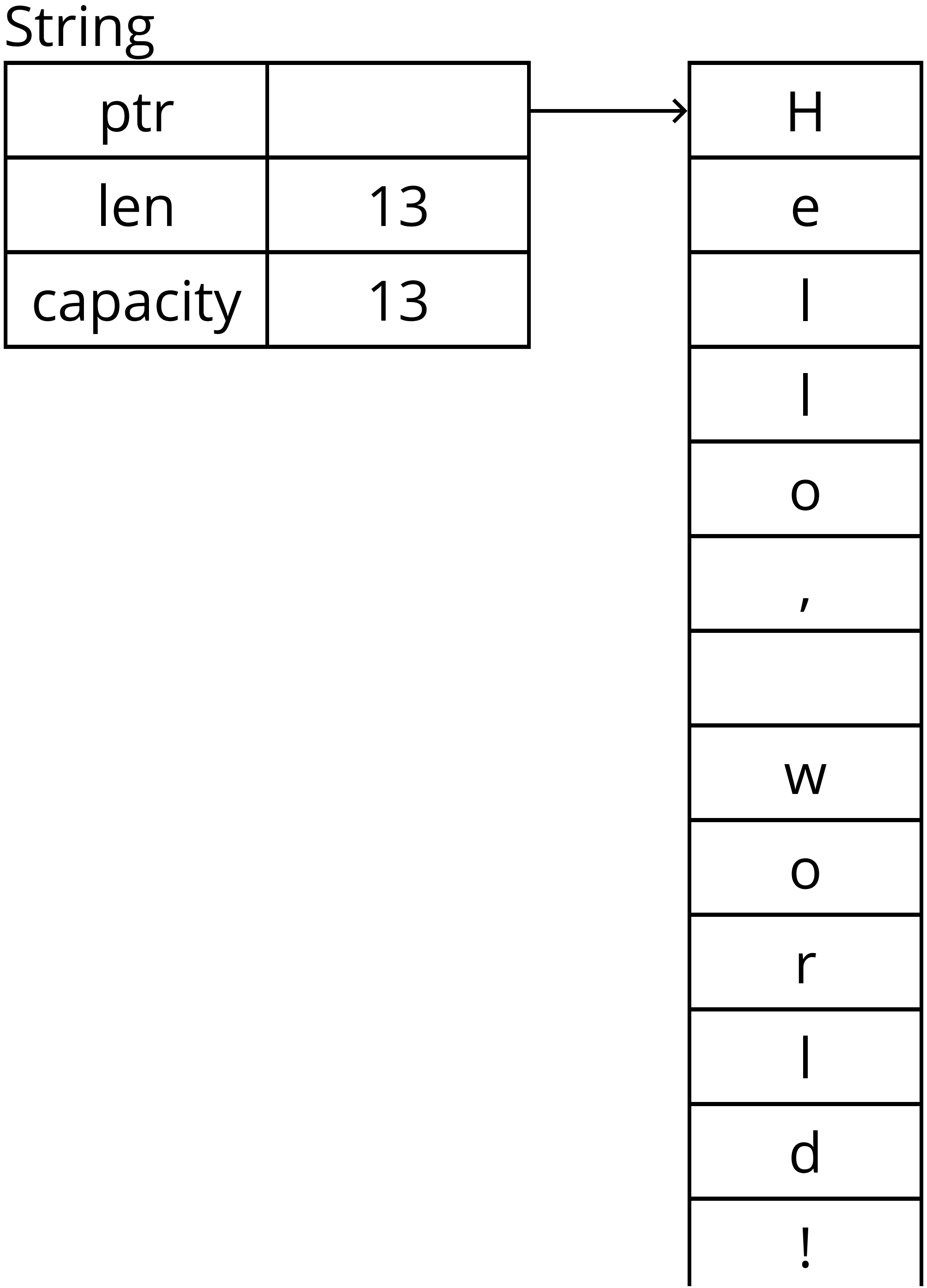
```
error[E0382]: borrow of moved value: `string_a`
 --> src/main.rs:6:22
  |
3 |     let string_a = String::from("Hello, world!");
  |         -------- move occurs because `string_a` has type `String`,
  |                  which does not implement the `Copy` trait
4 |     let string_b = string_a;
  |                    -------- value moved here
5 |
6 |     println!("{:?}", string_a);
  |                      ^^^^^^^^ value borrowed here after move
```

The rules of the game

- Each value in Rust has an *owner*
- There can only be a single owner at a time
- When the owner goes out of scope, the value will be *dropped*



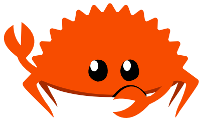
```rust
fn main() {

    let string_a = String::from("Hello, world!");
    let string_b = string_a;

    println!("{:?}", string_a);

}
```



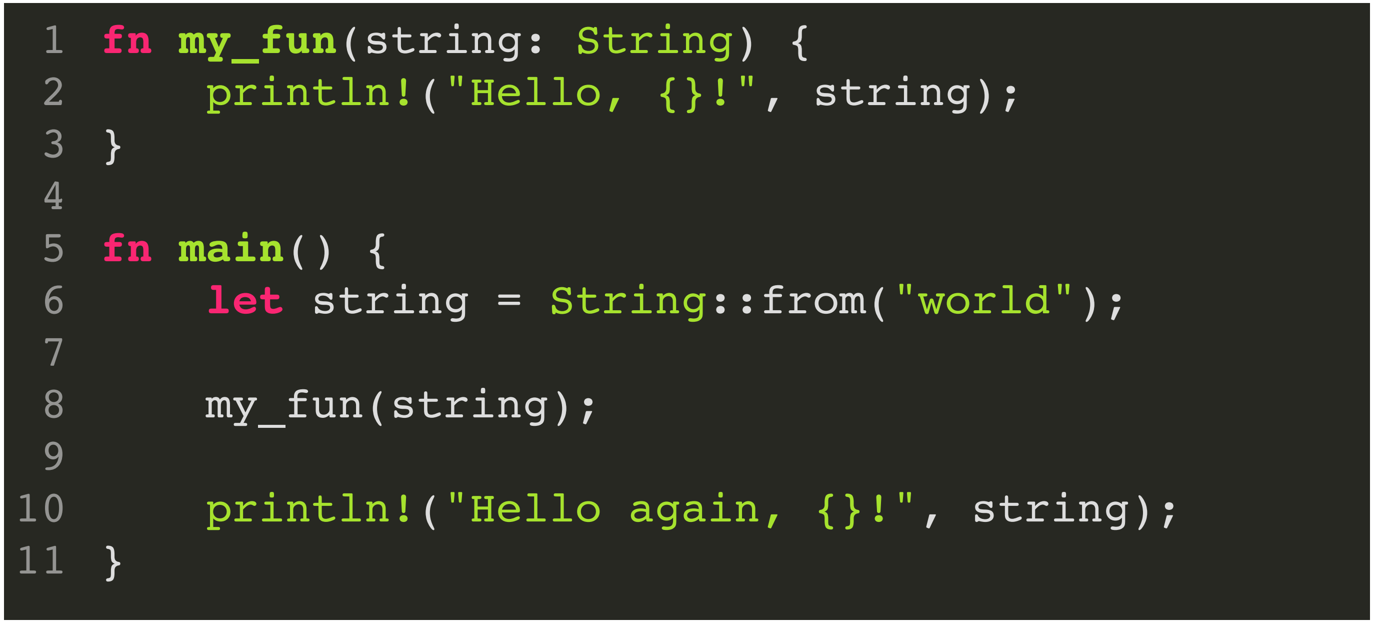
String

ptr	
len	13
capacity	13

H
e
l
l
o
,
w
o
r
l
d
!

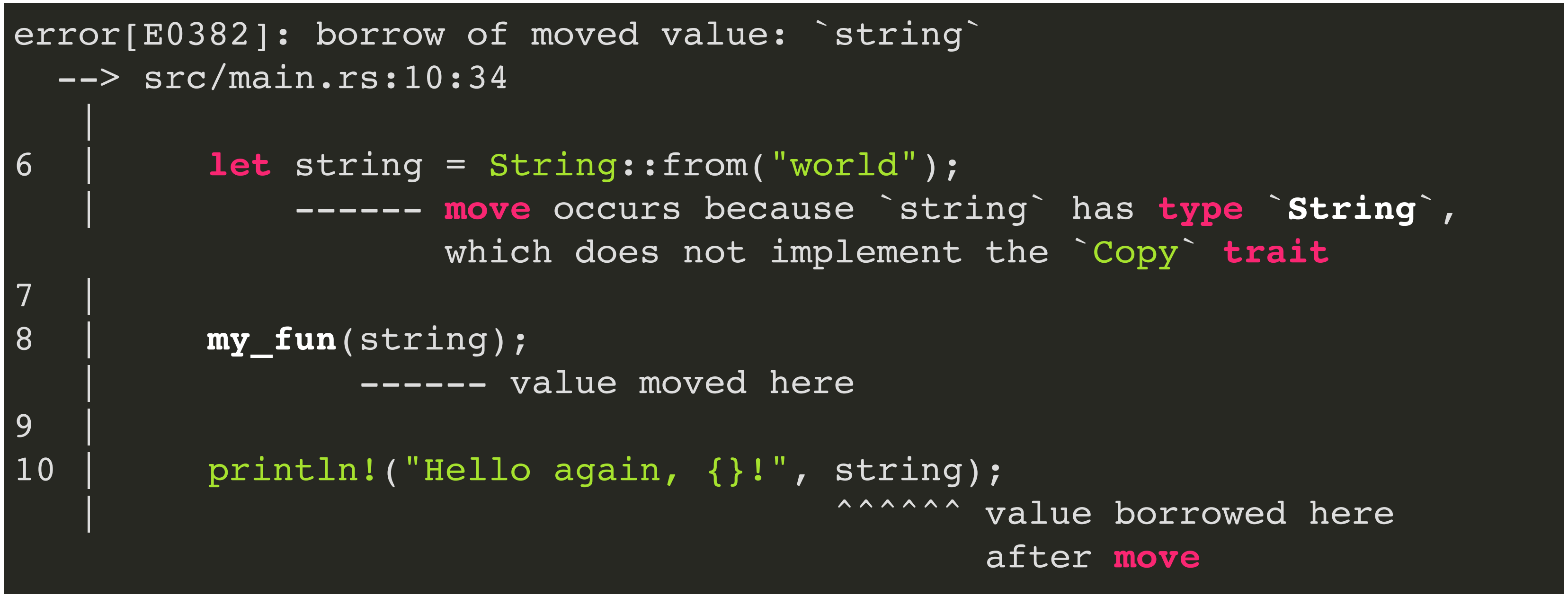


```rust
fn my_fun(string: String) {
    println!("Hello, {}!", string);
}

fn main() {
    let string = String::from("world");

    my_fun(string);

    println!("Hello again, {}!", string);
}
```



```
fn my_fun(string: String) {
    println!("Hello, {}!", string);
}

fn main() {
    let string = String::from("world");

    my_fun(string);

    println!("Hello again, {}!", string);
}
```

cargo run



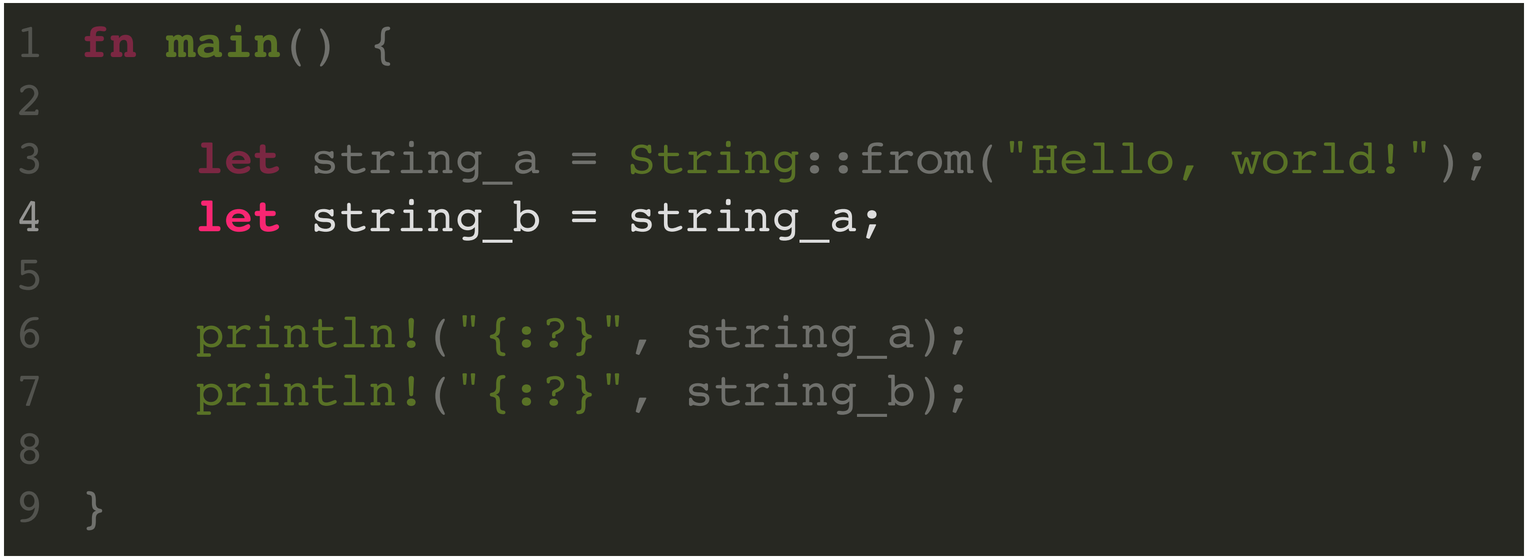
```
error[E0382]: borrow of moved value: `string`
  --> src/main.rs:10:34
   |
6  |     let string = String::from("world");
   |         ------ move occurs because `string` has type `String`,
                    which does not implement the `Copy` trait
7  |
8  |     my_fun(string);
   |            ------ value moved here
9  |
10 |     println!("Hello again, {}!", string);
   |                                  ^^^^^^ value borrowed here
                                             after move
```

Copy

- Performed implicitly by the compiler
- Performed by straight memcpy

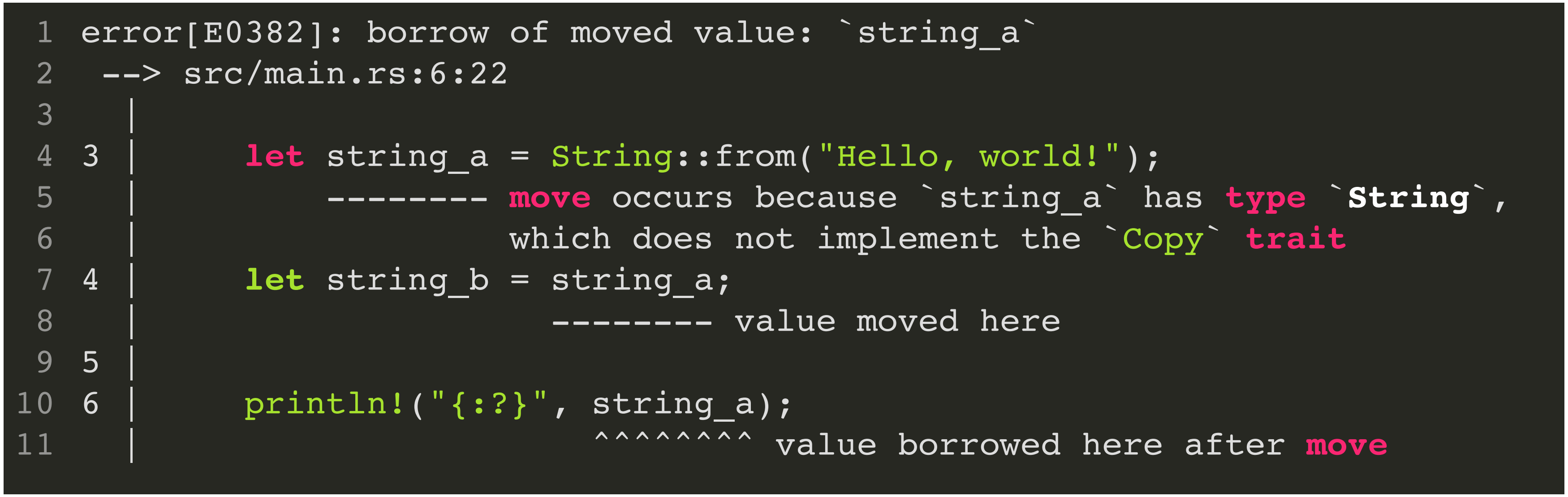
Clone

- Performed explicitly by calling `.clone()`
- Can have custom implementation



```
fn main() {

    let string_a = String::from("Hello, world!");
    let string_b = string_a;

    println!("{:?}", string_a);
    println!("{:?}", string_b);

}
```

cargo run

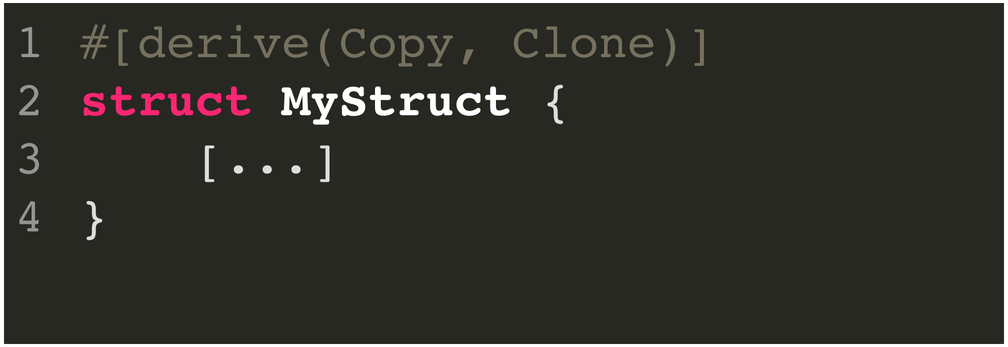


```
error[E0382]: borrow of moved value: `string_a`
 --> src/main.rs:6:22
  |
3 |     let string_a = String::from("Hello, world!");
  |         -------- move occurs because `string_a` has type `String`,
  |                  which does not implement the `Copy` trait
4 |     let string_b = string_a;
  |                    -------- value moved here
5 |
6 |     println!("{:?}", string_a);
  |                      ^^^^^^^^ value borrowed here after move
```

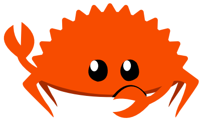
```rust
fn main() {

    let string_a = String::from("Hello, world!");
    let string_b = string_a.clone();

    println!("{:?}", string_a);
    println!("{:?}", string_b);

}
```

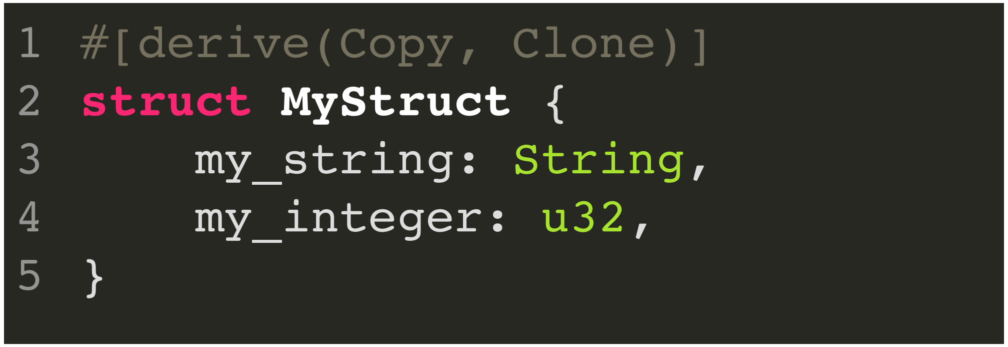
cargo run

```
$ cargo run
"Hello, world!"
"Hello, world!"
```



```
#[derive(Copy, Clone)]
struct MyStruct {
    [...]
}
```



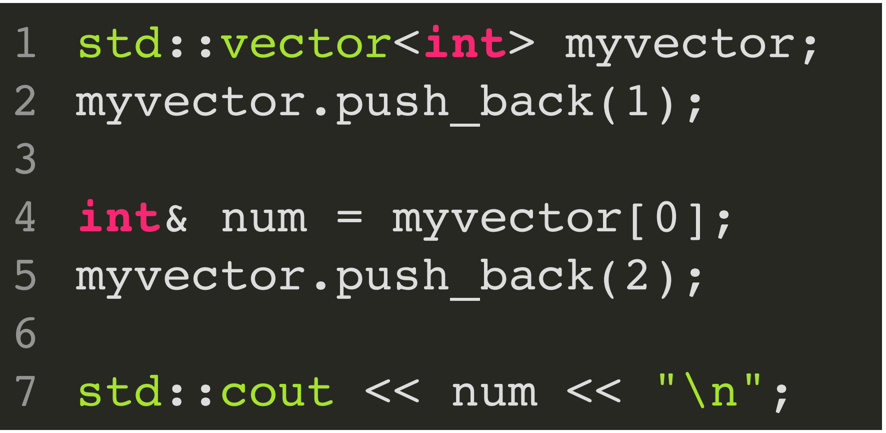


```
#[derive(Copy, Clone)]
struct MyStruct {
    my_string: String,
    my_integer: u32,
}
```

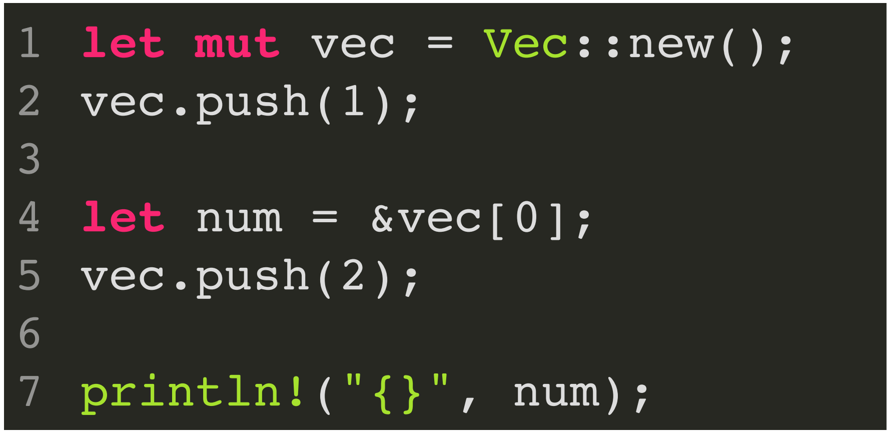
C#	R
class	struct without Copy
struct	struct with Copy
implementing IClonable	struct with Clone

The sequel of Ownership:

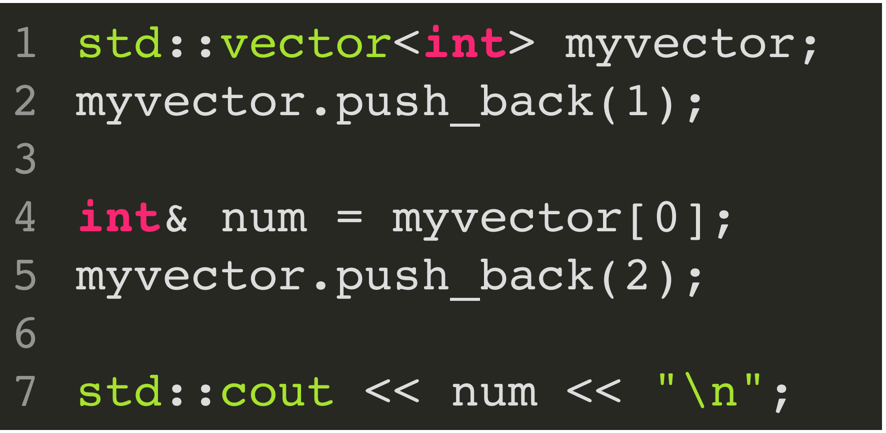
Borrowing & The Borrow Checker



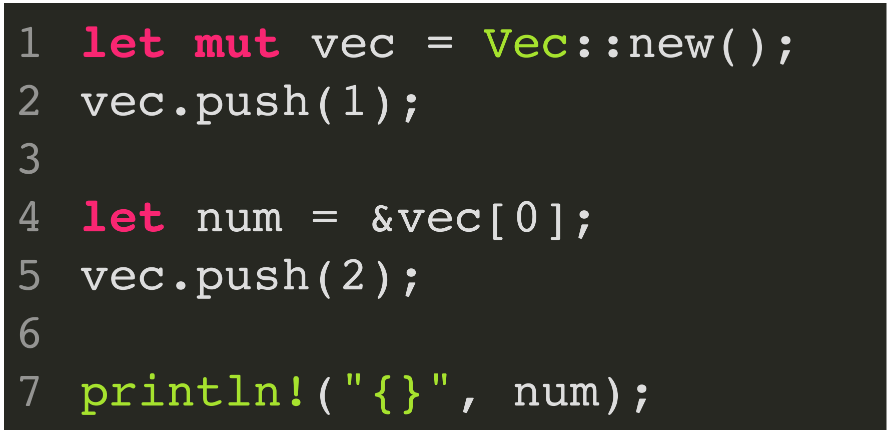
```cpp
std::vector<int> myvector;
myvector.push_back(1);

int& num = myvector[0];
myvector.push_back(2);

std::cout << num << "\n";
```



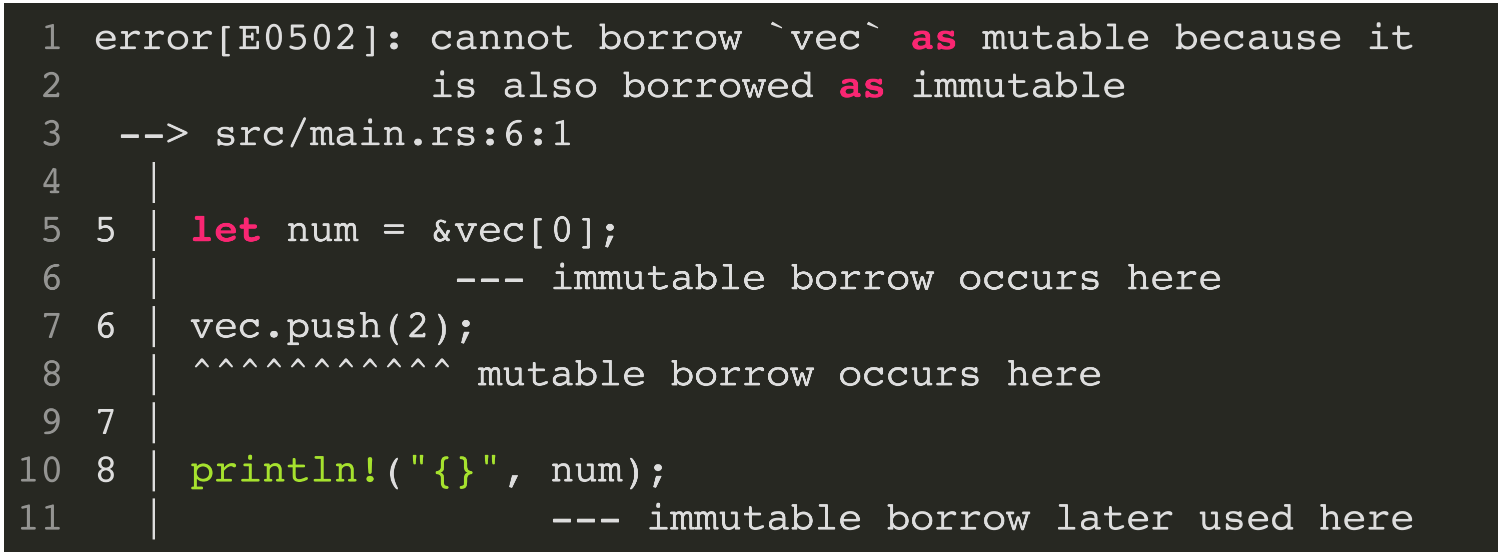
```rust
let mut vec = Vec::new();
vec.push(1);

let num = &vec[0];
vec.push(2);

println!("{}", num);
```



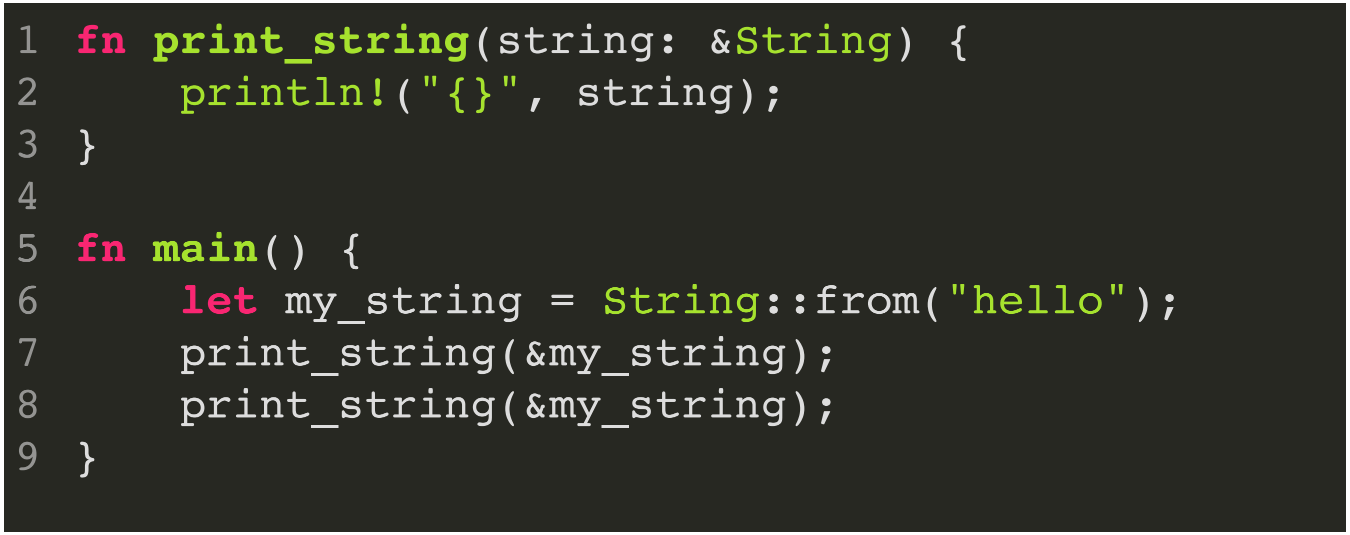
```cpp
std::vector<int> myvector;
myvector.push_back(1);

int& num = myvector[0];
myvector.push_back(2);

std::cout << num << "\n";
```



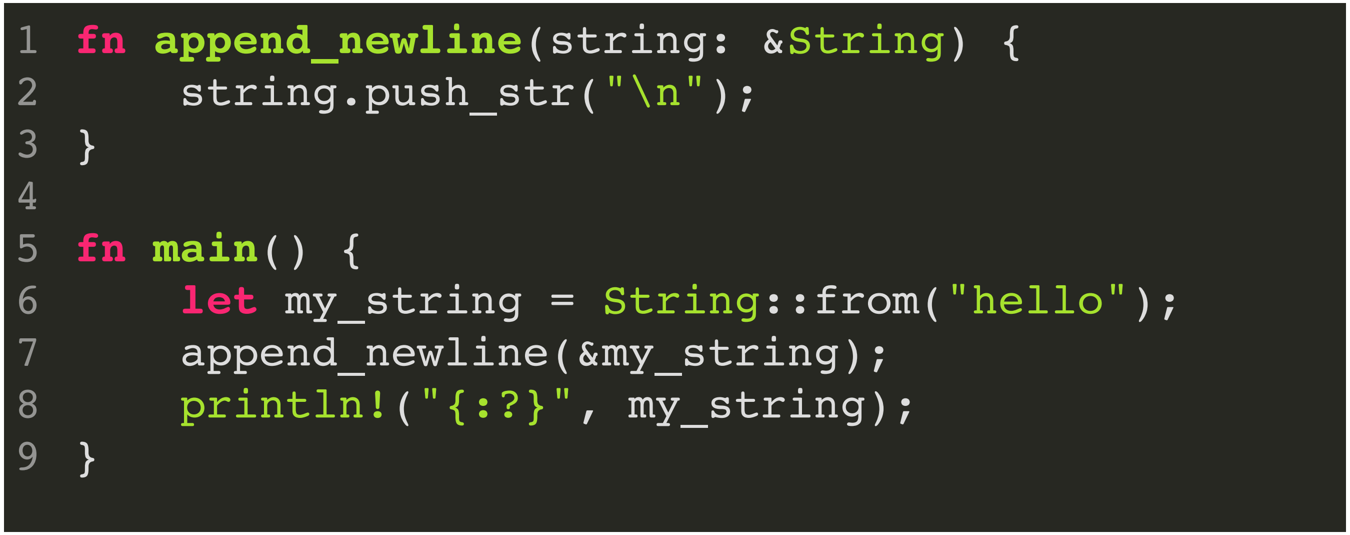
```rust
let mut vec = Vec::new();
vec.push(1);

let num = &vec[0];
vec.push(2);

println!("{}", num);
```



```
error[E0502]: cannot borrow `vec` as mutable because it
              is also borrowed as immutable
 --> src/main.rs:6:1
  |
5 | let num = &vec[0];
  |            --- immutable borrow occurs here
6 | vec.push(2);
  | ^^^^^^^^^^^ mutable borrow occurs here
7 |
8 | println!("{}", num);
  |                --- immutable borrow later used here
```



```rust
fn print_string(string: &String) {
    println!("{}", string);
}

fn main() {
    let my_string = String::from("hello");
    print_string(&my_string);
    print_string(&my_string);
}
```



```rust
fn append_newline(string: &String) {
    string.push_str("\n");
}

fn main() {
    let my_string = String::from("hello");
    append_newline(&my_string);
    println!("{:?}", my_string);
}
```

```rust
fn append_newline(string: &String) {
    string.push_str("\n");
}

fn main() {
    let my_string = String::from("hello");
    append_newline(&my_string);
    println!("{:?}", my_string);
}
```

⬇ cargo run

```
error[E0596]: cannot borrow `*string` as mutable, as it is behind a `&`
              reference
 --> src/main.rs:2:5
  |
1 | fn append_newline(string: &String) {
  |                           -------- help: consider changing this to be
                                       a mutable reference: `&mut String`
2 |     string.push_str("\n");
  |     ^^^^^^ `string` is a `&` reference, so the data it refers to
               cannot be borrowed as mutable
```

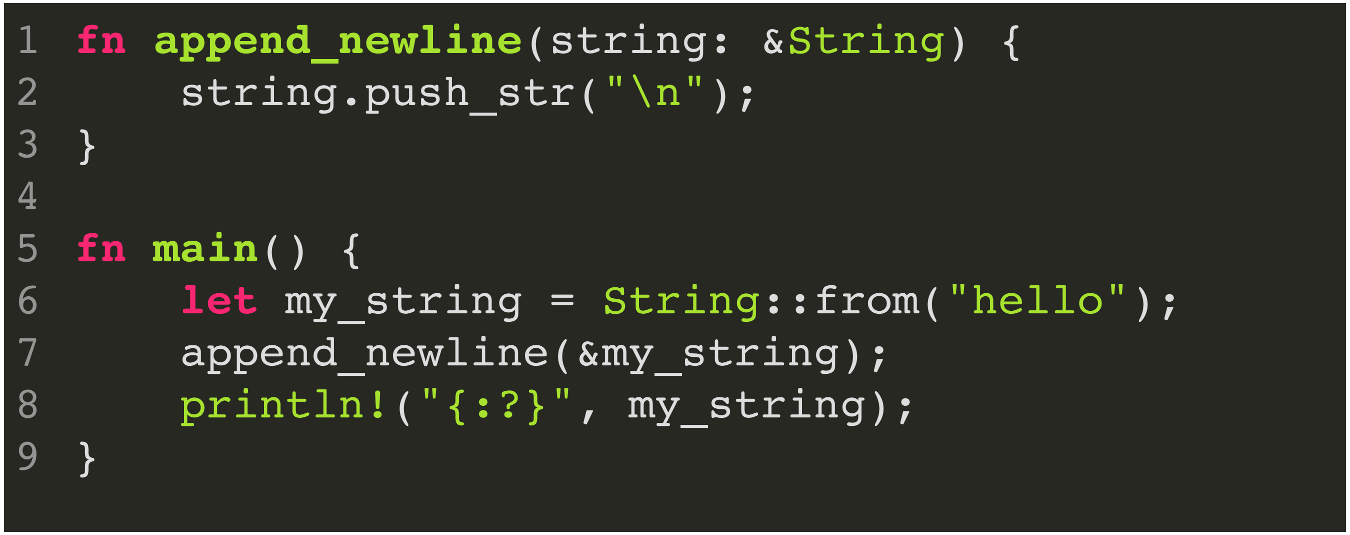
`&_`

- several may exist for a single piece of data
- may be copied
- may only be read

`&mut _`

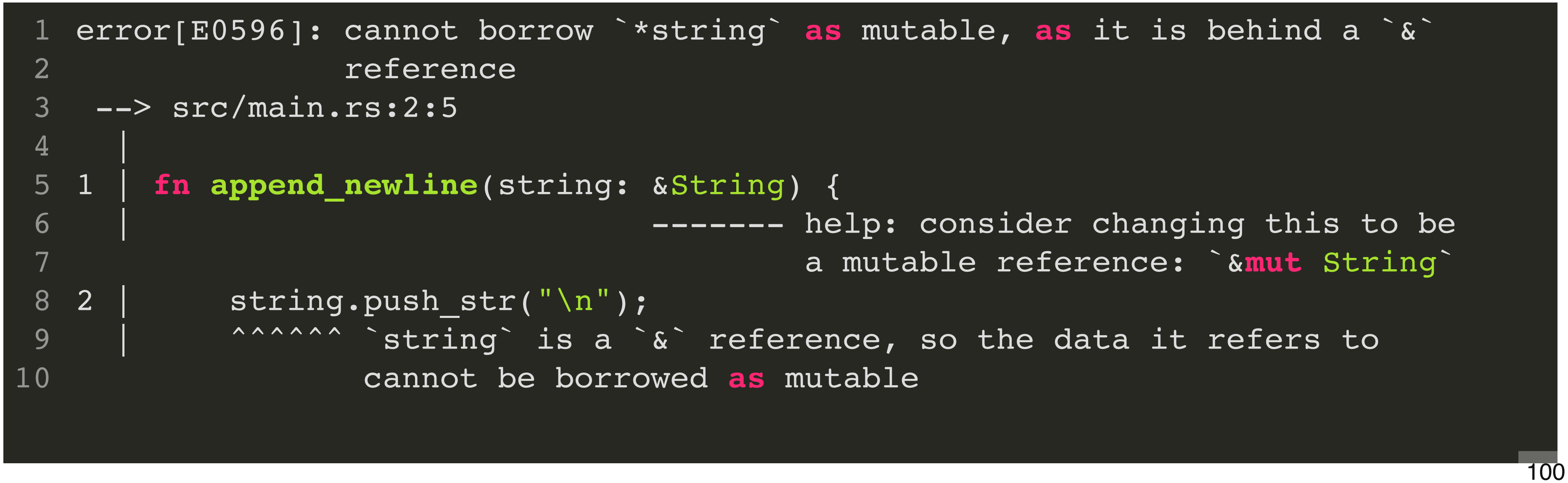
- only ONE may exist for any single piece of data
- may only be moved
- may be written or read

Mutually exclusive!

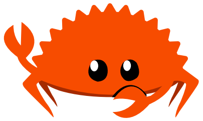


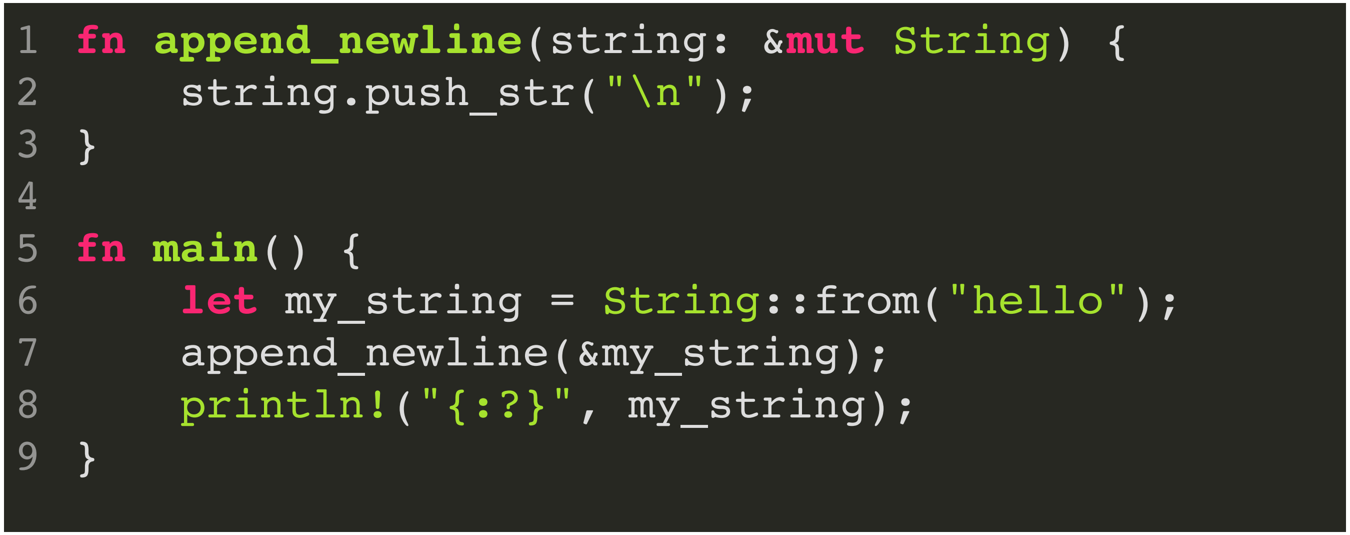
```rust
fn append_newline(string: &String) {
    string.push_str("\n");
}

fn main() {
    let my_string = String::from("hello");
    append_newline(&my_string);
    println!("{:?}", my_string);
}
```

cargo run



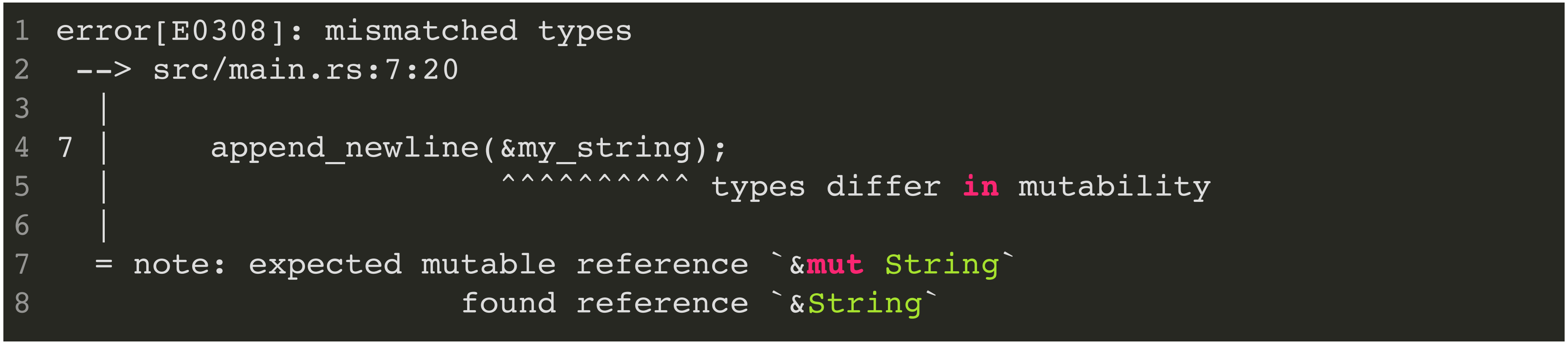
```
error[E0596]: cannot borrow `*string` as mutable, as it is behind a `&`
              reference
 --> src/main.rs:2:5
  |
1 | fn append_newline(string: &String) {
  |                           ------- help: consider changing this to be
                                      a mutable reference: `&mut String`
2 |     string.push_str("\n");
  |     ^^^^^^ `string` is a `&` reference, so the data it refers to
               cannot be borrowed as mutable
```





```rust
fn append_newline(string: &mut String) {
    string.push_str("\n");
}

fn main() {
    let my_string = String::from("hello");
    append_newline(&my_string);
    println!("{:?}", my_string);
}
```

cargo run



```
error[E0308]: mismatched types
 --> src/main.rs:7:20
  |
7 |     append_newline(&my_string);
  |                    ^^^^^^^^^^ types differ in mutability
  |
  = note: expected mutable reference `&mut String`
                     found reference `&String`
```

```
fn append_newline(string: &mut String) {
    string.push_str("\n");
}

fn main() {
    let my_string = String::from("hello");
    append_newline(&mut my_string);
    println!("{:?}", my_string);
}
```

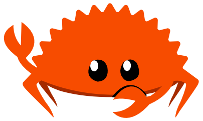
cargo run

```
error[E0596]: cannot borrow `my_string` as mutable, as it is
              not declared as mutable
 --> src/main.rs:7:20
  |
6 |     let my_string = String::from("hello");
  |         --------- help: consider changing this to be
                            mutable: `mut my_string`
7 |     append_newline(&mut my_string);
  |                    ^^^^^^^^^^^^^^ cannot borrow as mutable
```

```rust
fn append_newline(string: &mut String) {
    string.push_str("\n");
}

fn main() {
    let mut my_string = String::from("hello");
    append_newline(&mut my_string);
    println!("{:?}", my_string);
}
```

```
cargo run
```

```
"hello\n"
```

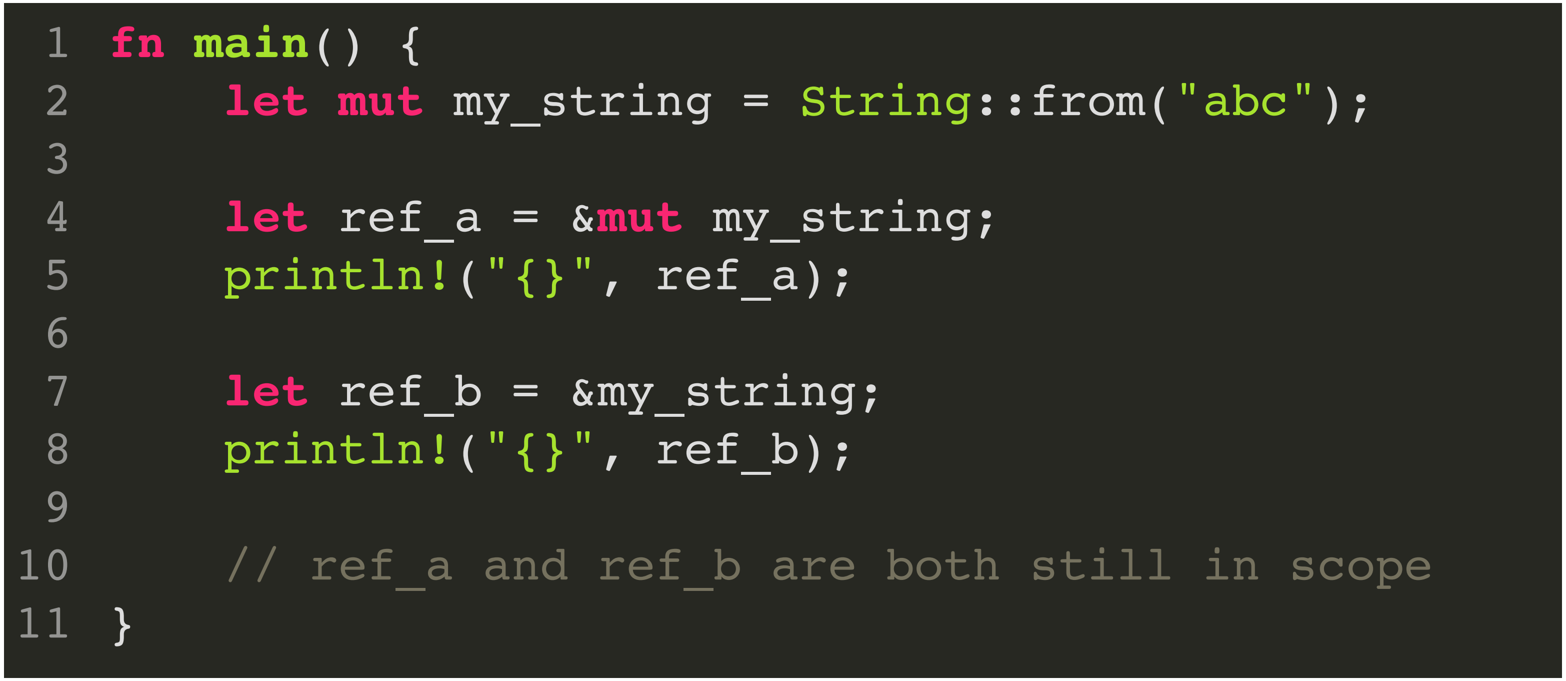


```rust
fn main() {
    let mut my_string = String::from("abc");

    let ref_a = &mut my_string;
    let ref_b = &my_string;

    println!("{} {}", ref_a, ref_b);
}
```

```rust
1 fn main() {
2     let mut my_string = String::from("abc");
3 
4     let ref_a = &mut my_string;
5     let ref_b = &my_string;
6 
7     println!("{} {}", ref_a, ref_b);
8 }
```

⬇ cargo run

```
error[E0502]: cannot borrow `my_string` as immutable because
              it is also borrowed as mutable
 --> src/main.rs:5:17
  |
4 |     let ref_a = &mut my_string;
  |                 -------------- mutable borrow occurs here
5 |     let ref_b = &my_string;
  |                 ^^^^^^^^^^ immutable borrow occurs here
6 |
7 |     println!("{} {}", ref_a, ref_b);
  |                       ----- mutable borrow later used here
```



```rust
fn main() {
    let mut my_string = String::from("abc");

    let ref_a = &mut my_string;
    println!("{}", ref_a);

    let ref_b = &my_string;
    println!("{}", ref_b);

    // ref_a and ref_b are both still in scope
}
```

```
$ cargo run
abc
abc
```

Dereferencing

```
let mut a = 0;

let a_ref = &mut a;
*a_ref = 1;

println!(a);
```

```rust
let a = 12;
let a_ref = &a;

let a2 = *a_ref;
```