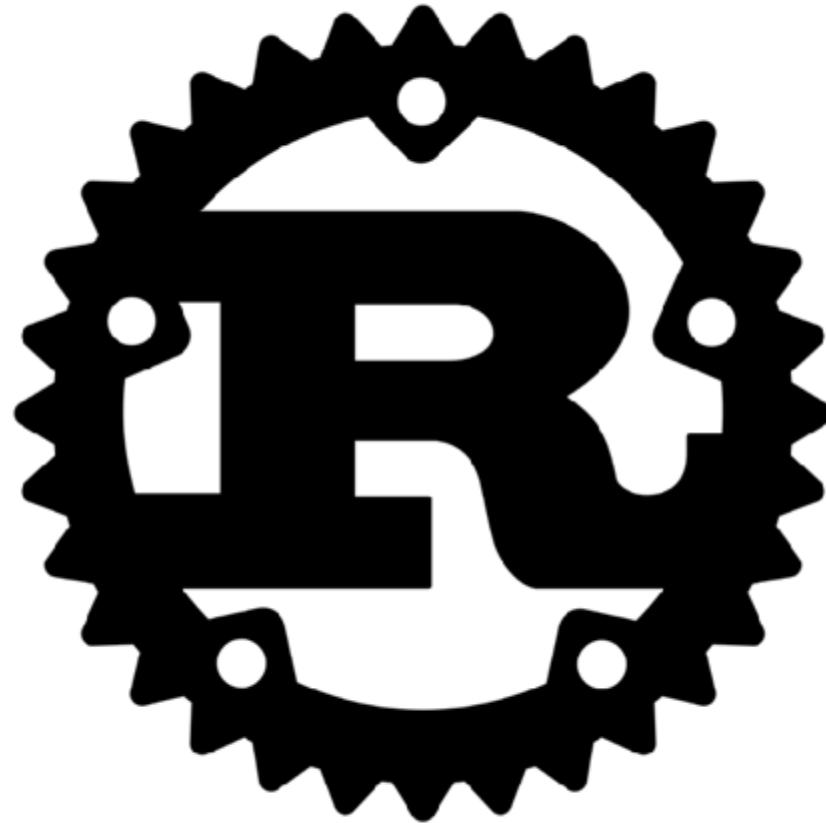


# Rust Programming

## A Crash Course

# GitHub



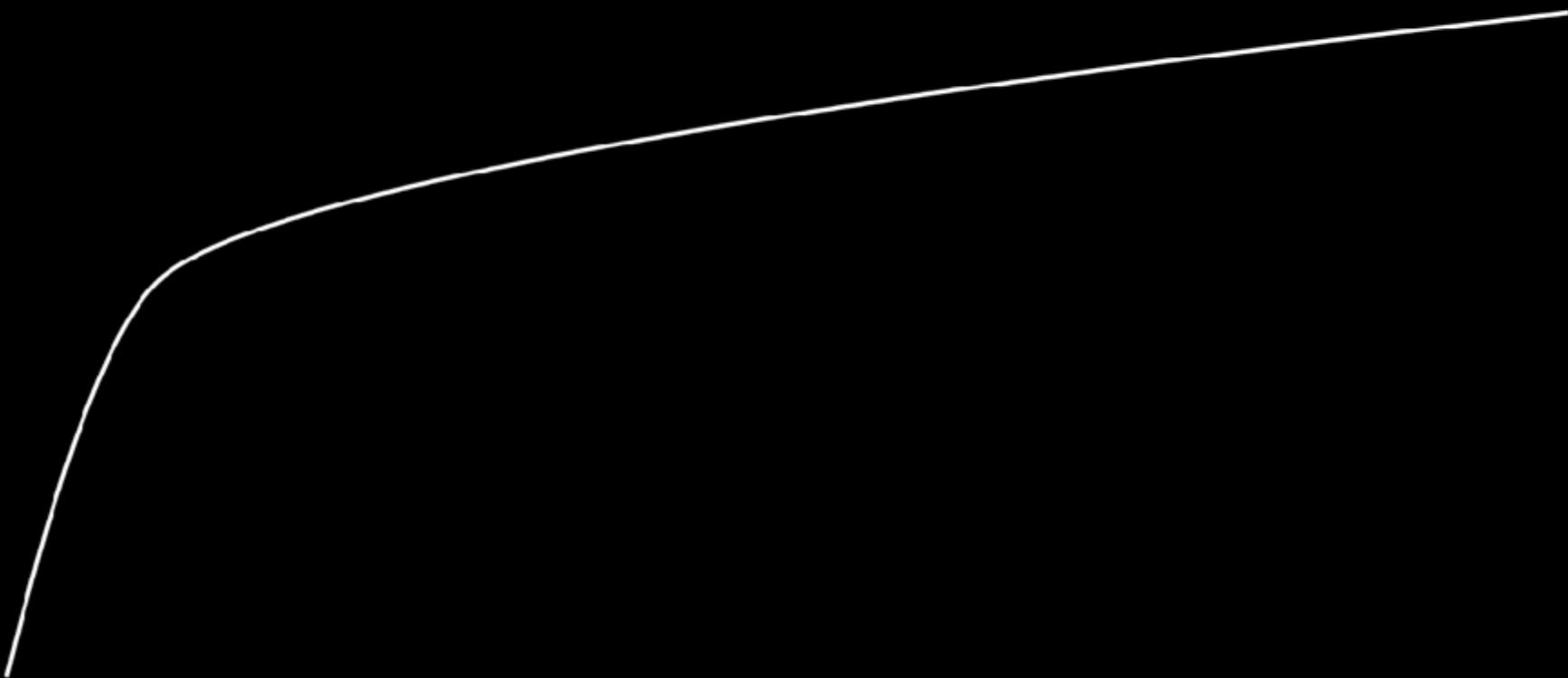


# Kind of Language I Use Most Often

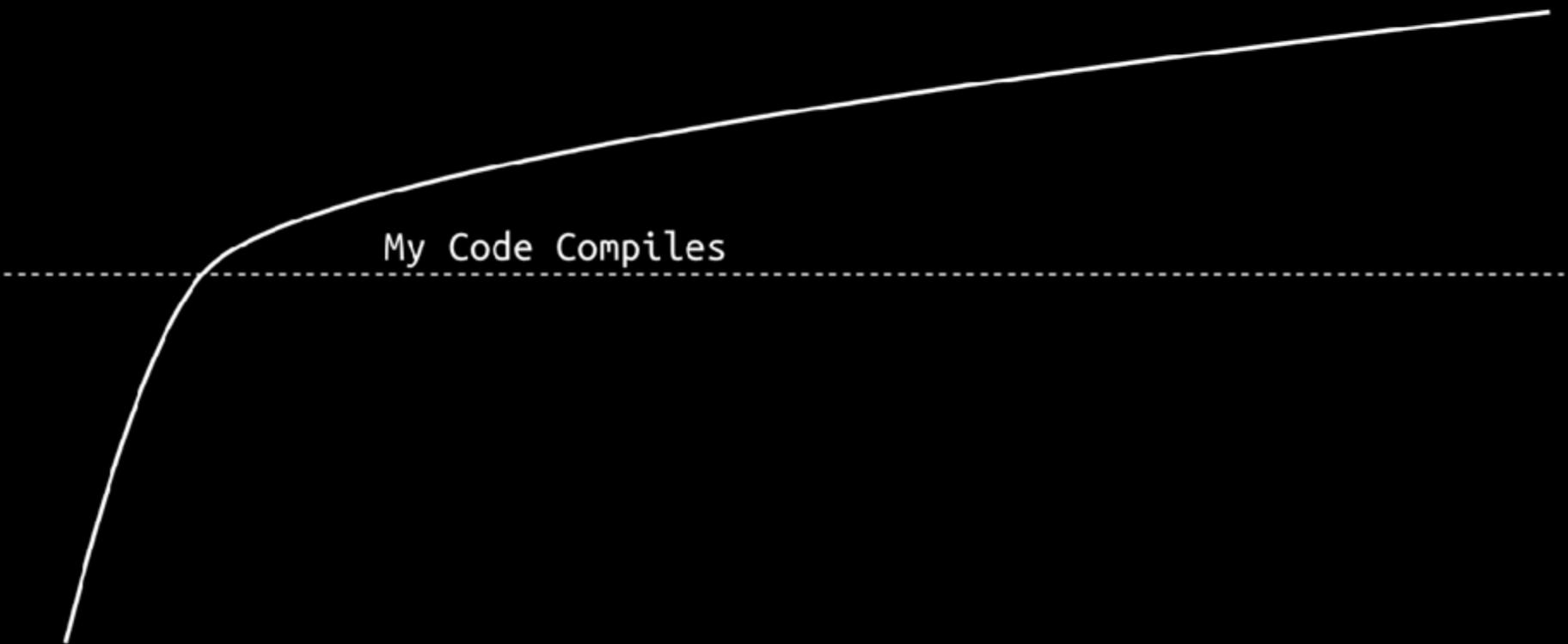
- A) Scripting (Python, Ruby, Javascript, Lua, TCL, ...)
- B) General Purpose (Go, Haskell, Scala, C#, Java, ...)
- C) Systems Programming (C, C++, Rust, D, ...)
- D) Math or Data Science (R, Julia, Matlab, Fortran, ...)
- E) I often use languages across different categories
- F) Other — Explain in a chat comment!



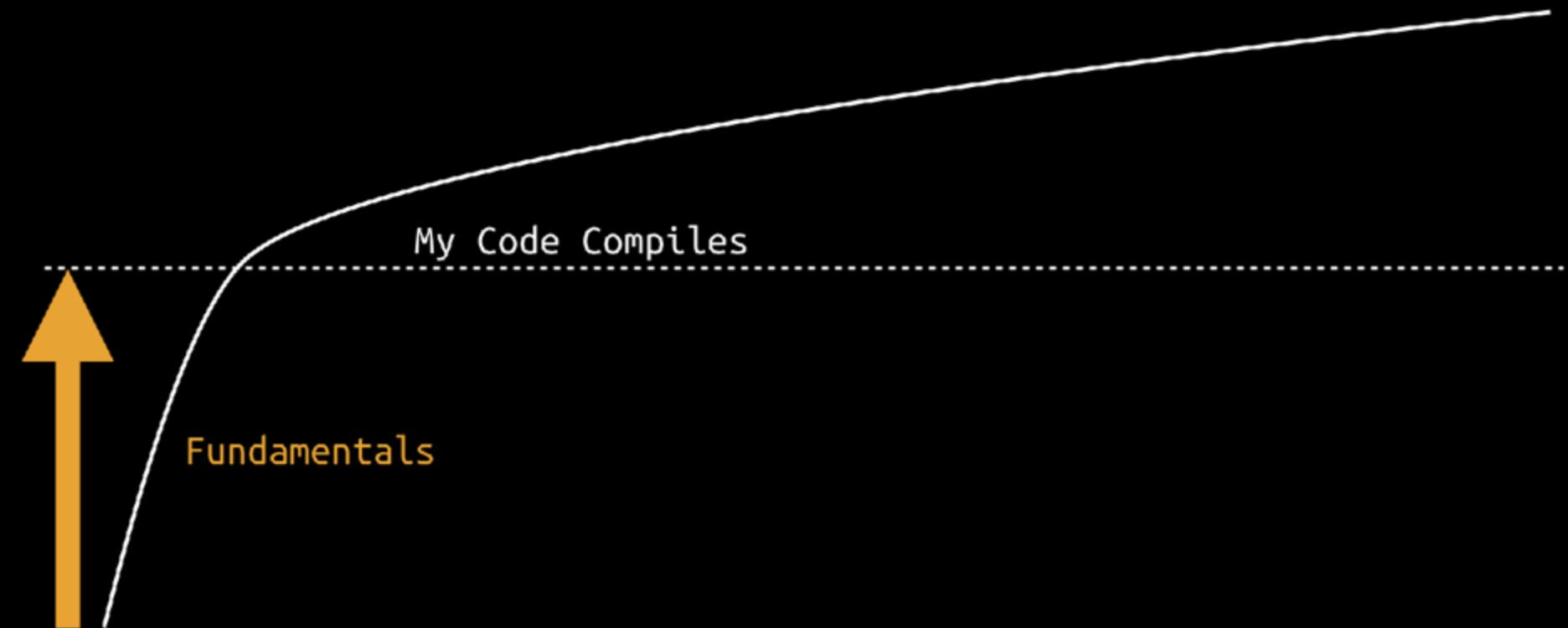
# Rust's Steep Learning Curve



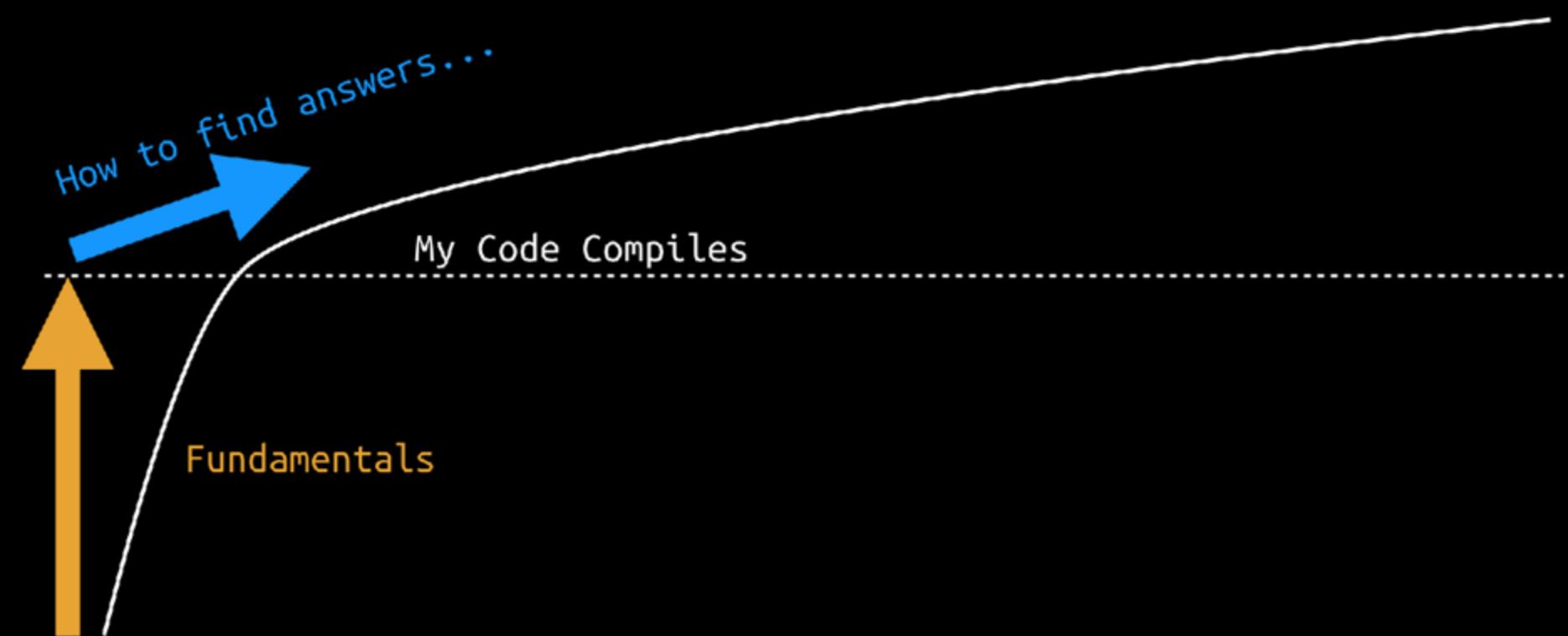
# Rust's Steep Learning Curve



# Rust's Steep Learning Curve



# Rust's Steep Learning Curve



Rust is Awesome

# Systems Programming Language

Safety, Concurrency, Speed

Ruby ↘

Safety

↑ Python

 C, C++ 

Concurrency, Speed

2006 – Graydon Hoare

moz://a



The new **Firefox**  
Fast for Good.

## Introducing the New Firefox: Firefox Quantum

Mark Mayo | November 14, 2017

**I**t's fast. Really fast. Firefox Quantum is over twice as fast as Firefox from 6 months ago, built on a completely overhauled core engine with brand new technology stolen from our advanced research group,

# Performance

Safe & Reliable

# Documentation

```
///! This is a nicely documented "Hello World" project.  
///!  
///! The runnable code in this project looks like this:  
///!  
///! ...  
///! fn main() {  
///!     println!("Hello, World");  
///! }  
///! ...  
  
///! The `main()` function outputs "Hello, World!" This function is called  
///! automatically when the `hello` binary is run. The recommended way to  
///! run the `hello` binary is to run:  
///!  
///! ...  
///! cargo run  
///! ...  
pub fn main() {  
    println!("Hello, World!");  
}  
}
```

A screenshot of a web browser window titled "hello\_world - Rust". The URL in the address bar is "file:///Users/nathan/rust/hello/target/doc/hello\_world/index.html". The page content is the documentation for a "Hello World" project:

Click or press 'S' to search, '?' for more options...

**Crate hello\_world**

**Crates** [-] [src]

**hello\_world**

This is a nicely documented "Hello World" project.

The runnable code in this project looks like this:

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

**Functions**

**main** The `main()` function outputs "Hello, World!" This function is called automatically when the `hello` binary is run. The recommended way to run the `hello` binary is to run:

# Tests

# Platform Support

# Community

Friendly, Safe & Welcoming

# Planning

Rust 2018 Edition

# Useful Error Messages

# Editor Support

# Package Manager

What are you most excited about?

# Ownership

# Systems Programming

time

# Ownership

1. Each value has an owner

2. Only one owner

3. Value gets dropped if its owner goes out of scope

*Questions?*

# Section 2: Cargo & Variables



# Cargo

cargo = package manager

cargo = build system

cargo = test runner

cargo = docs generator

```
$ cargo new hello
```

```
$ cargo new hello  
Created binary (application) `hello` package
```

```
$
```

```
$ cargo new hello  
Created binary (application) 'hello' package
```

```
$ tree --noreport hello
```

```
$ cargo new hello  
    Created binary (application) 'hello' package
```

```
$ tree --noreport hello
```

```
hello  
└── Cargo.toml  
    └── src  
        └── main.rs
```

```
$
```

```
$ cargo new hello  
  Created binary (application) 'hello' package
```

```
$ tree --noreport hello
```

```
hello  
└── Cargo.toml  
    └── src  
        └── main.rs
```

```
$
```

```
$ vim Cargo.toml
```

```
[package]
name = "hello"
version = "0.1.0"
authors = ["Nathan Stocks <nathan.stocks@gmail.com>"]
edition = "2018"

[dependencies]
```

```
[package]
name = "hello"
version = "0.1.0"
authors = ["Nathan Stocks <nathan.stocks@gmail.com>"]
edition = "2018"

[dependencies]
```

```
[package]
name = "hello"
version = "0.1.0" 
See semver.org for more details!
authors = ["Nathan Stocks <nathan.stocks@gmail.com>"]
edition = "2018"
```

```
[dependencies]
```

```
[package]
name = "hello"      See semver.org for more details!
version = "0.1.0"
authors = ["Nathan Stocks <nathan.stocks@gmail.com>"]
edition = "2018"
```

```
[dependencies]
```

```
[package]
name = "hello"      See semver.org for more details!
version = "0.1.0"
authors = ["Nathan Stocks <nathan.stocks@gmail.com>"]
edition = "2018"
```

```
[dependencies]
```

```
$ vim src/main.rs
```

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

```
$ cargo run
```

```
$ cargo run
Compiling hello v0.1.0 (/Users/nathan/hello)
Finished dev [unoptimized + debuginfo] target(s) in 0.87 secs
Running `target/debug/hello'
Hello, world!
$
```

```
$ cargo run  
Compiling hello v0.1.0 (/Users/nathan/hello)  
  Finished dev [unoptimized + debuginfo] target(s) in 0.87 secs  
    Running `target/debug/hello`  
Hello, world!
```

```
$ cargo run  
  Finished dev [unoptimized + debuginfo] target(s) in 0.63 secs  
    Running `target/debug/hello`  
Hello, world!
```

```
$
```

```
$ cargo run
Compiling hello v0.1.0 (/Users/nathan/hello)
Finished dev [unoptimized + debuginfo] target(s) in 0.87 secs
Running `target/debug/hello`
Hello, world!

$ cargo run
Finished dev [unoptimized + debuginfo] target(s) in 0.63 secs
Running `target/debug/hello`
Hello, world!

$ ls target/debug/hello
target/debug/hello

$
```

```
$ cargo run
  Compiling hello v0.1.0 (/Users/nathan/hello)
    Finished dev [unoptimized + debuginfo] target(s) in 0.87 secs
      Running `target/debug/hello`
Hello, world!

$ cargo run
  Finished dev [unoptimized + debuginfo] target(s) in 0.63 secs
      Running `target/debug/hello`
Hello, world!

$ ls target/debug/hello
target/debug/hello

$ target/debug/hello
Hello, world!
```

```
$ cargo run --release
```

```
$ cargo run --release
Compiling hello v0.1.0 (/Users/nathan/hello)
Finished dev [optimized] target(s) in 0.84 secs
Running `target/release/hello'
Hello, world!
$
```

# Variables

```
fn main() {  
    let bunnies = 2;  
}
```



```
fn main() {  
    let bunnies = 2;  
}
```

```
fn main() {  
    let bunnies = 2;  
}
```

```
fn main() {  
    let bunnies: i32 = 4;  
}
```

```
fn main() {  
    let bunnies: i32 = 4;  
}
```



```
fn main() {  
    let (bunnies, carrots) = (8, 50);  
}
```

```
fn main() {  
    let (bunnies, carrots) = (8, 50);  
}
```

```
fn main() {  
    let (bunnies, carrots) = (8, 50);  
}
```

```
fn main() {  
    let bunnies = 16;  
}
```

Why are variables immutable by default in  
Rust? Make your guess!

- A) Safety
- B) Concurrency
- C) Speed
- D) Safety, Concurrency, AND Speed
- E) It's just the way things have always been done

```
fn main() {  
    let bunnies = 16;  
    bunnies = 2; // Error!  
}
```

```
error[E0384]: cannot assign twice to immutable variable `bunnies`
--> src/main.rs:3:5
```

```
2 |     let bunnies = 16;
|     -----
|     |
|     first assignment to `bunnies`
|     help: make this binding mutable: `mut bunnies`
3 |     bunnies = 2; // Error!
|     ^^^^^^^^^^^^ cannot assign twice to immutable variable
```

```
error: aborting due to previous error
```

```
For more information about this error, try `rustc --explain E0384`.
error: Could not compile `hoppity`.
```

```
error[E0384]: cannot assign twice to immutable variable `bunnies`
--> src/main.rs:3:5
```

```
| 2 |     let bunnies = 16;
|     |
|     |     -----
|     |     first assignment to `bunnies`
|     |     help: make this binding mutable: `mut bunnies`
| 3 |     bunnies = 2; // Error!
|     |     ^^^^^^^^^^ cannot assign twice to immutable variable
```

```
error: aborting due to previous error
```

```
For more information about this error, try `rustc --explain E0384`.
error: Could not compile `hoppity`.
```

```
error[E0384]: cannot assign twice to immutable variable `bunnies`
--> src/main.rs:3:5
```

```
2 |     let bunnies = 16;
|     -----
|     |
|     first assignment to `bunnies`
|     help: make this binding mutable: `mut bunnies`
3 |     bunnies = 2; // Error!
|     ^^^^^^^^^^^^ cannot assign twice to immutable variable
```

```
error: aborting due to previous error
```

```
For more information about this error, try `rustc --explain E0384`.
error: Could not compile `hoppity`.
```

```
error[E0384]: cannot assign twice to immutable variable `bunnies`
--> src/main.rs:3:5
```

```
2 |     let bunnies = 16;
|     -----
|     |
|     first assignment to `bunnies`
|     help: make this binding mutable: `mut bunnies`
3 |     bunnies = 2; // Error!
|     ^^^^^^^^^^^^ cannot assign twice to immutable variable
```

```
error: aborting due to previous error
```

```
For more information about this error, try `rustc --explain E0384`.
error: Could not compile `hoppity`.
```

```
error[E0384]: cannot assign twice to immutable variable `bunnies`
--> src/main.rs:3:5
```

```
2 |     let bunnies = 16;
|     -----
|     |
|     first assignment to `bunnies`
|     help: make this binding mutable: `mut bunnies`
3 |     bunnies = 2; // Error!
|     ^^^^^^^^^^^^ cannot assign twice to immutable variable
```

```
error: aborting due to previous error
```

```
For more information about this error, try `rustc --explain E0384`.
error: Could not compile `hoppity`.
```

```
error[E0384]: cannot assign twice to immutable variable `bunnies`
--> src/main.rs:3:5
```

```
2 |     let bunnies = 16;
|     -----
|     |
|     first assignment to `bunnies`
|     help: make this binding mutable: `mut bunnies`
3 |     bunnies = 2; // Error!
|     ^^^^^^^^^^^^ cannot assign twice to immutable variable
```

```
error: aborting due to previous error
```

```
For more information about this error, try `rustc --explain E0384`.
error: Could not compile `hoppity`.
```

```
error[E0384]: cannot assign twice to immutable variable `bunnies`
--> src/main.rs:3:5
```

```
2 |     let bunnies = 16;
|     -----
|     |
|     first assignment to `bunnies`
|     help: make this binding mutable: `mut bunnies`
3 |     bunnies = 2; // Error!
|     ^^^^^^^^^^^^ cannot assign twice to immutable variable
```

```
error: aborting due to previous error
```

```
For more information about this error, try `rustc --explain E0384`.
error: Could not compile `hoppity`.
```

```
error[E0384]: cannot assign twice to immutable variable `bunnies`
--> src/main.rs:3:5
```

```
2 |     let bunnies = 16;
|     -----
|     |
|     first assignment to `bunnies`
|     help: make this binding mutable: `mut bunnies`
3 |     bunnies = 2; // Error!
|     ^^^^^^^^^^^^ cannot assign twice to immutable variable
```

```
error: aborting due to previous error
```

```
For more information about this error, try `rustc --explain E0384`.
error: Could not compile `hoppity`.
```

What is the correct syntax for initializing a mutable variable?

See: "rustc --explain E0384"

- A) mutable let bunnies = 16;
- B) let bunnies = 16 mut;
- C) let mut bunnies = 16;
- D) let \*bunnies = &16;

```
fn main() {  
    let mut bunnies = 32;  
    bunnies = 2;  
}
```

```
fn main() {  
    let mut bunnies = 32;  
    bunnies = 2;  
}
```

const

```
let warp_factor = ask_scotty();
```

```
const warp_factor = ask_scotty();
```



```
const WARP_FACTOR = ask_scotty();
```



```
const WARP_FACTOR: f64 = ask_scotty();
```



```
const WARP_FACTOR: f64 = 9.9;
```





# *Exercise A*

# *Exercise A*

[https://github.com/CleanCut/rust\\_programming](https://github.com/CleanCut/rust_programming)

*Questions?*

# Section 3: Functions

# Scope

```
fn main() {
    let x = 5;
    {
        let y = 99;
        println!("{} , {}" , x , y);
    }
    println!("{} , {}" , x , y); // Error!
}
```

```
fn main() {
    let x = 5;
    {
        let y = 99;
        println!("{} , {}", x, y);
    }
    println!("{} , {}", x, y); // Error!
}
```

```
fn main() {
    let x = 5;
    {
        let y = 99;
        println!("{} , {}", x, y);
    }
    println!("{} , {}", x, y); // Error!
}
```

```
fn main() {
    let x = 5;
    {
        let y = 99;
        println!("{} , {}", x, y);
    }
    println!("{} , {}", x, y); // Error!
}
```

```
fn main() {
    let x = 5;
    {
        let y = 99;
        println!("{} , {}" , x , y);
    }
    println!("{} , {}" , x , y); // Error!
}
```

```
fn main() {  
    let x = 5;  
    {  
        let y = 99;  
        println!("{} , {}" , x , y);  
    } ←  
    println!("{} , {}" , x , y); // Error!  
}
```

```
fn main() {
    let x = 5;
    {
        let y = 99;
        println!("{} , {}" , x , y);
    }
    println!("{} , {}" , x , y); // Error!
}
```

```
error[E0425]: cannot find value `y` in this scope
--> src/main.rs:7:27
7 |     println!("{} , {}" , x , y); // Error!
   |           ^ did you mean `x`?
```

```
error: aborting due to previous error
```

```
For more information about this error, try `rustc --explain E0425`.
error: Could not compile `hello`.
```

```
error[E0425]: cannot find value `y` in this scope
--> src/main.rs:7:27
7 |     println!("{} , {}" , x , y); // Error!
      |           ^ did you mean `x`?
```

```
error: aborting due to previous error
```

```
For more information about this error, try `rustc --explain E0425`.
error: Could not compile `hello`.
```

```
fn main() {
    let x = 5;
    {
        let y = 99;
        println!("{} , {}" , x , y);
    }
    println!("{} , {}" , x , y); // Error!
}
```

```
fn main() {
    let x = 5;
    {
        let x = 99;
        println!("{}", x); // Prints "99"
    }
    println!("{}", x); // Prints "5"
}
```

```
fn main() {
    let x = 5;
    {
        let x = 99;
        println!("{}", x); // Prints "99"
    }
    println!("{}", x); // Prints "5"
}
```

```
fn main() {
    let x = 5;
    {
        let x = 99;
        println!("{}", x); // Prints "99"
    }
    println!("{}", x); // Prints "5"
}
```

```
fn main() {
    let x = 5;
    {
        let x = 99;
        println!("{}", x); // Prints "99"
    }
    println!("{}", x); // Prints "5"
}
```

```
fn main() {  
    let x = 5;  
    {  
        let x = 99;  
        println!("{}", x); // Prints "99"  
    } ←  
    println!("{}", x); // Prints "5"  
}
```

```
fn main() {
    let x = 5;
    {
        let x = 99;
        println!("{}", x); // Prints "99"
    }
    println!("{}", x); // Prints "5"
}
```

```
fn main() {  
    let mut x = 5;    // x is mutable  
    let x = x;        // x is now immutable  
}
```

```
fn main() {  
    let mut x = 5;    // x is mutable  
    let x = x;        // x is now immutable  
}
```

```
fn main() {  
    let mut x = 5;    // x is mutable  
    let x = x;        // x is now immutable  
}
```

```
fn main() {  
    let meme = "More cowbell!";  
    let meme = make_image(meme);  
}
```

# Memory Safety

```
fn main() {  
    let enigma: i32;  
    println!("{}", enigma); // Error!  
}
```

```
error[E0381]: use of possibly uninitialized variable: `enigma`
--> src/main.rs:3:20
3 |     println!("{}", enigma);
   |          ^^^^^^ use of possibly uninitialized `enigma`  
error: aborting due to previous error  
error: Could not compile `hello`.
```

```
error[E0381]: use of possibly uninitialized variable: `enigma`
--> src/main.rs:3:20
3 |     println!("{}", enigma);
|          ^^^^^^ use of possibly uninitialized `enigma`  
error: aborting due to previous error
error: Could not compile `hello`.
```

```
fn main() {
    let enigma: i32;
    if true {
        enigma = 42;
    }
    println!("enigma is {}", enigma); // Error!
}
```

```
fn main() {
    let enigma: i32;
    if true {
        enigma = 42;
    }
    println!("enigma is {}", enigma); // Error!
}
```

```
fn main() {
    let enigma: i32;
    if true {
        enigma = 42;
    } else {
        enigma = 7;
    }
    println!("enigma is {}", enigma);
}
```

```
fn main() {
    let enigma: i32;
    if true {
        enigma = 42;
    } else {
        enigma = 7;
    }
    println!("enigma is {}", enigma);
}
```

```
#include <stdio.h>

int main() {
    int enigma;
    printf("%d\n", enigma);
}
```

```
$ gcc hello.c -o hello
```

```
$ ./hello
```

```
1
```

```
$
```

# Functions

```
fn main() {  
}
```

```
fn do_stuff() {  
}
```



```
fn do_stuff() {  
}
```

```
fn main() {  
    do_stuff(); // It works!  
}
```

```
fn do_stuff() {  
}
```



```
fn do_stuff(qty: f64
```

```
fn do_stuff(qty: f64, oz: f64)
```

```
fn do_stuff(qty: f64, oz: f64) -> f64
```

```
fn do_stuff(qty: f64, oz: f64) -> f64 {  
}
```

```
fn do_stuff(qty: f64, oz: f64) -> f64 {  
    return qty * oz;  
}
```

```
fn do_stuff(qty: f64, oz: f64) -> f64 {  
    qty * oz  
}
```

```
{ return true; }
```

{ true }

```
fn main() {  
    let x = do_stuff(2.0, 12.5);  
}  
  
fn do_stuff(qty: f64, oz: f64) -> f64 {  
    qty * oz  
}
```

```
fn main() {
    let x = do_stuff(2.0, 12.5);
}

fn do_stuff(qty: f64, oz: f64) -> f64 {
    println!("{} {}-oz sarsaparilla(s)!", qty, oz);
    qty * oz
}
```

*Exercise B*

*Questions?*

*5-min Break*

**PLACEHOLDER**  
**Timer: 5 Minutes**

# Section 4: Module System

```
$ tree --noreport hello
```

```
hello
```

```
└── Cargo.toml
```

```
src
```

```
└── lib.rs ←
```

```
    └── main.rs
```

```
$ tree --noreport hello
hello
└── Cargo.toml
    └── src
        ├── lib.rs
        └── main.rs      // the 'hello' binary
```

```
$ tree --noreport hello
hello
└── Cargo.toml
    └── src
        ├── lib.rs          // the `hello` library
        └── main.rs          // the `hello` binary
```

```
$ tree --noreport hello
hello
└── Cargo.toml
    └── src
        ├── lib.rs          // the `hello` library
        └── main.rs          // the `hello` binary
```

```
$ vim src/lib.rs
```

```
fn greet() {  
    println!("Hi!");  
}
```

src/lib.rs

```
fn greet() {  
    println!("Hi!");  
}
```

src/main.rs

```
fn main() {  
    hello::greet(); // won't work, yet  
}
```

src/lib.rs

```
fn greet() {  
    println!("Hi!");  
}
```

src/main.rs

```
fn main() {  
    hello::greet(); // won't work, yet  
}
```

src/lib.rs

```
fn greet() {  
    println!("Hi!");  
}
```

src/main.rs

```
fn main() {  
    hello::greet(); // won't work, yet  
}
```

src/lib.rs

```
pub fn greet() {  
    println!("Hi!");  
}
```

src/main.rs

```
fn main() {  
    hello::greet(); // works!  
}
```

src/lib.rs

```
pub fn greet() {  
    println!("Hi!");  
}
```

src/main.rs

```
fn main() {  
    hello::greet(); // works!  
}
```

use

src/lib.rs

```
pub fn greet() {  
    println!("Hi!!!");  
}
```

src/main.rs

```
fn main() {  
    hello::greet();  
}
```

src/lib.rs

```
pub fn greet() {  
    println!("Hi!!!");  
}
```

src/main.rs

```
use hello::greet;  
  
fn main() {  
    hello::greet();  
}
```

src/lib.rs

```
pub fn greet() {  
    println!("Hi!!!");  
}
```

src/main.rs

```
use hello::greet;  
  
fn main() {  
    greet();  
}
```

```
use std::collections::HashMap;
```



rust std |



Google Search

I'm Feeling Lucky



rust std vector



All

Shopping

News

Images

Videos

More

Settings

Tools

About 564,000 results (0.68 seconds)

## std::vec::Vec - Rust

<https://doc.rust-lang.org/std/vec/struct.Vec.html> ▾

API documentation for the Rust `Vec` struct in crate `std`. ... Pushing 10 or fewer elements onto the vector will not change its capacity or cause reallocation to ...

### Vec<T>

The vec! macro is provided to make initialization more ...

### Struct std::vec::Vec

Most fundamentally, Vec is and always will be a (pointer ...

### std::vec::Vec

API documentation for the Rust `Vec` struct in crate `std`.

[More results from rust-lang.org »](#)



# The Rust community's crate registry

Install Cargo

Getting Started

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.

**1,034,918,825** Downloads  
 **25,783** Crates in stock

## New Crates

tilecoding (0.1.1)



## Most Downloaded

libc (0.2.54)

## Just Updated

nummap (0.3.0)



lang-tester (0.1.0)



rand (0.6.5)

sodiumoxide (0.2.2)



```
[package]
name = "hello"
version = "0.1.0"
authors = ["Nathan Stocks <nathan.stocks@gmail.com>"]
edition = "2018"

[dependencies]
```

```
[package]
name = "hello"
version = "0.1.0"
authors = ["Nathan Stocks <nathan.stocks@gmail.com>"]
edition = "2018"

[dependencies]
```

```
[package]
name = "hello"
version = "0.1.0"
authors = ["Nathan Stocks <nathan.stocks@gmail.com>"]
edition = "2018"

[dependencies]
rand = "0.6.5"
```

```
fn main() {  
    let x = rand::thread_rng().gen_range(0, 100);  
}
```

```
use rand::thread_rng;

fn main() {
    let x = thread_rng().gen_range(0, 100);
}
```

# Section 5: Simple Types

# Scalar Types

# Integer Types

Unsigned	Signed
u8	i8
u16	i16
u32	i32
u64	i64
u128	i128
usize	isize

**Unsigned**      **Signed**

**u8**

**i8**

**u16**

**i16**

**u32**

**i32**

**u64**

**i64**

**u128**

**i128**

**usize**

**isize**

**Unsigned**

**Signed**

**u8**

**i8**

**u16**

**i16**

**u32**

**i32**

**u64**

**i64**

**u128**

**i128**

**usize**

**isize**

Unsigned

**u8**

**u16**

**u32**

**u64**

**u128**

**usize**

Signed

**i8**

**i16**

**i32**

**i64**

**i128**

**isize**

**Unsigned**

**u8**

**u16**

**u32**

**u64**

**u128**

**usize**

**Signed**

**i8**

**i16**

**i32**

**i64**

**i128**

**isize**

Unsigned	Signed
u8	i8
u16	i16
u32	i32
u64	i64
u128	i128
usize	isize

Unsigned	Signed
u8	i8
u16	i16
u32	i32
u64	i64
u128	i128
usize	isize

Unsigned

u8

u16

u32

u64

u128

usize

Signed

i8

i16

i32

i64

i128

isize

Unsigned	Signed
u8	i8
u16	i16
u32	i32
u64	i64
u128	i128
usize	isize

Unsigned

u8

u16

usize

Signed

i8

i16

isize

# Integer Literals

Decimal	1000000
Hex	0xdeadbeef
Octal	0o77543211
Binary	0b11110011
Byte (u8 only)	b'A'

Decimal	1000000
Hex	0xdeadbeef
Octal	0o77543211
Binary	0b11110011
Byte (u8 only)	b'A'

Decimal	1000000
Hex	0xdeadbeef
Octal	0o77543211
Binary	0b11110011
Byte (u8 only)	b'A'

Decimal	1000000
Hex	0xdeadbeef
Octal	0o77543211
Binary	0b11110011
Byte (u8 only)	b'A'

Decimal	1000000
Hex	0xdeadbeef
Octal	0o77543211
Binary	0b11110011
Byte (u8 only)	b'A'

Decimal	1000000
Hex	0xdeadbeef
Octal	0o77543211
Binary	0b11110011
Byte (u8 only)	b'A'

Decimal	1000000
Hex	0xdeadbeef
Octal	0o77543211
Binary	0b11110011
Byte (u8 only)	b'A'

Decimal	1 <u>000</u> _000
Hex	0xdead_beef
Octal	0o7754 <u>3211</u>
Binary	0b1111 <u>0011</u>
Byte (u8 only)	b 'A'

Decimal	1_0_0_0_0_0_0
Hex	0xde_ad_be_ef
Octal	0o77_54_32_11
Binary	0b11_11_00_11
Byte (u8 only)	b 'A'

# Floating Point types

f32

f64

f32

f64

# Floating Point Literals

IEEE-754

3.14159

X

.1



θ . 1

# Suffixes

```
let x: u16 = 5;  
let y: f32 = 3.14;
```

```
let x = 5u16;  
let y = 3.14f32;
```

```
let x = 5_u16;  
let y = 3.14_f32;
```

# Boolean Type

bool

true

false

true as u8

false as u8

# Character Type

A large brown horse is grazing in a lush green field. The horse is positioned on the left side of the frame, facing towards the right. It has a thick, dark brown coat and a long, flowing mane. In the background, there's a wooden stable or barn on the left and rolling green hills under a bright blue sky with scattered white clouds.

# Dragon

char

b

й

你

۹



nothing to see here

4 bytes (32 bits)

[char]

UCS-4 / UTF-32

```
let my_letter = 'a';
let i_kratkoa = 'й';
let ideograph = '你';
let diacritic = 'ő';
let my_rocket = '🚀';
```

# Compound Types

# Tuple

```
let info = (1, 3.3, 999);
```

```
let info: (u8, f64, i32) = (1, 3.3, 999);
```

```
let info = (1, 3.3, 999);
```

```
let info = (1, 3.3, 999);
let jets = info.0;
let fuel = info.1;
let ammo = info.2;
```

```
let info = (1, 3.3, 999);
// let jets = info.0;
// let fuel = info.1;
// let ammo = info.2;
let (jets, fuel, ammo) = info;
```

(u8, u8, i32, u64)





# Array

```
let buf = [1, 2, 3];
```

How Many



```
let buf = [0; 3];
```



Value

```
let buf: [u8; 3] = [1, 2, 3];
```



buf[0]

32

Vec

# *Exercise C*

*Questions?*

# Section 6: Control Flow & Strings

# Control Flow

```
if num == 5 {  
    msg = "five";  
}
```

```
if num == 5 {  
    msg = "five";  
}
```

```
if num == 5 {  
    msg = "five";  
}  
else if
```

```
if num == 5 {  
    msg = "five";  
}
```

```
if num == 5 {  
    msg = "five";  
} else if num == 4 {  
    msg = "four";  
}
```

```
if num == 5 {  
    msg = "five";  
} else if num == 4 {  
    msg = "four";  
}
```

```
if num == 5 {  
    msg = "five";  
} else if num == 4 {  
    msg = "four";  
} else {  
    msg = "other";  
}
```

```
if num == 5 {  
    msg = "five";  
} else if num == 4 {  
    msg = "four";  
} else {  
    msg = "other";  
}
```

```
msg = if num == 5 {  
    "five"  
} else if num == 4 {  
    "four"  
} else {  
    "other"  
};
```

```
msg = if num == 5 {  
    "five"  
} else if num == 4 {  
    "four"  
} else {  
    "other"  
};
```

```
msg = if num == 5 {  
    "five"  
} else if num == 4 {  
    "four"  
} else {  
    "other"  
};
```

```
msg = if num == 5 {  
    "five"  
} else if num == 4 {  
    "four"  
} else {  
    "other"  
};
```

```
msg = if num == 5 {  
    "five"  
} else if num == 4 {  
    "four"  
} else {  
    "other"  
};
```

```
if (being_attacked)  
    klaxon_alert();
```

C

```
if (being_attacked)
    klaxon_alert(); Isn't that just the worst? Another pain point we avoid is
    fire_missiles(); // Oops.
```

C



```
num = a ? b : c;
```

C

```
num = a ? x ? y : z : c;
```

C

```
num = a  
? x ? y : z  
: c;
```

C

```
num = if a { b } else { c };
```

```
num = if a {  
    if x { y } else { y }  
} else {  
    c  
};
```

loop { }

```
loop {  
    break;  
}
```

```
loop {  
    loop {  
        loop {  
            break;  
        }  
    }  
}
```

```
'bob: loop {  
    loop {  
        loop {  
            break;  
        }  
    }  
}
```

```
'bob: loop {  
    loop {  
        loop {  
            break;  
        }  
    }  
}
```

```
'bob: loop {  
    loop {  
        loop {  
            break;  
        }  
    }  
}
```

```
'bob: loop {  
    loop {  
        loop {  
            break 'bob;  
        }  
    }  
}
```

```
'bob: loop {  
    loop {  
        continue;  
    }  
}
```

```
'bob: loop {  
    loop {  
        continue;  
    }  
}
```

```
'bob': loop {  
    loop {  
        continue 'bob';  
    }  
}
```

```
'bob: loop {  
    loop {  
        continue 'bob;  
    }  
}
```

```
while dizzy() {  
    // do stuff  
}
```

```
loop {  
    if !dizzy() { break }  
    // do stuff  
}
```

```
loop {
    // do stuff
    if !dizzy() { break }
}
```

```
for num in [7, 8, 9].iter() {  
    // do stuff with num  
}
```

```
for (x, y) in [(1,2), (3,4)] {  
    // do stuff with x and y  
}
```

```
for num in 0..50 {  
    // do stuff with num  
}
```

```
for num in 0..50 {  
    // do stuff with num  
}
```

```
for num in 0..=50 {  
    // do stuff with num  
}
```

# Strings



6 types

2 types

str

&str

```
let msg = "Hello 🌎";
```



&str

String

```
let msg = "ab".to_string();
```



```
let msg = String::from("ab");
```



`&str`

`ptr →`



`len = 4`

# String

ptr → 

a	b						
---	---	--	---	--	--	--	--

len = 4

capacity = 8

# String

ptr →



len = 4

&str

capacity = 8

Valid UTF-8



my\_string[3]



English

6900+ languages

```
let word = "สวัสดี";
```

```
let word = "สวัสดี";
```



```
word[3]
```

# bytes

224	184	170	224	184	167	224	184	177	224	184	170	224	184	148	224	184	181
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

# bytes

224	184	170	224	184	167	224	184	177	224	184	170	224	184	148	224	184	181
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----



word[3]

# bytes

224	184	170	224	184	167	224	184	177	224	184	170	224	184	148	224	184	181
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

# unicode scalars

'ଶ'	'ବ'	'ଓ'	'ଶ'	'ଡ'	'ଓ'
-----	-----	-----	-----	-----	-----

# bytes

224	184	170	224	184	167	224	184	177	224	184	170	224	184	148	224	184	181
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

## unicode scalars

```
[ 'ଶ' , 'ବ' , 'ୟ' , 'ଶ' , 'ଡ' , 'ୟ' ]
```



word[3]

# bytes

224	184	170	224	184	167	224	184	177	224	184	170	224	184	148	224	184	181
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

## unicode scalars

```
[ 'ଶ', 'ବ', 'ଓ', 'ଶ', 'ଦ', 'ଓ' ]
```





# bytes

224	184	170	224	184	167	224	184	177	224	184	170	224	184	148	224	184	181
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

## unicode scalars

```
[ 'ສ', 'ວ', 'ໝ', 'ສ', 'ດ', 'ຝ' ]
```

## unicode graphemes

```
[ "ສ", "ວ", "ສ", "ດ" ]
```

# bytes

224	184	170	224	184	167	224	184	177	224	184	170	224	184	148	224	184	181
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

## unicode scalars

```
[ 'ສ', 'ວ', 'ໝ', 'ສ', 'ດ', 'ໝ' ]
```

## unicode graphemes

```
[ "ສ", "ວ", "ສ", "ດ" ]
```

# bytes

224	184	170	224	184	167	224	184	177	224	184	170	224	184	148	224	184	181
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

## unicode scalars

```
[ 'ສ', 'ວ', 'ໝ', 'ສ', 'ດ', 'ຝ' ]
```

## unicode graphemes

```
[ "ສ", "ວ", "ສ", "ດ" ]
```



# bytes

224	184	170	224	184	167	224	184	177	224	184	170	224	184	148	224	184	181
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

# bytes

224	184	170	224	184	167	224	184	177	224	184	170	224	184	148	224	184	181
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

```
word.bytes();
```

```
word.chars();
```

unicode-segmentation

```
graphemes(my_string, true)
```

.push()	.contains()	.trim_left()
.truncate()	.starts_with()	.trim_right()
.pop()	.ends_with()	.trim_matches()
.remove()	.find()	.trim_left_matches()
.insert()	.rfind()	.trim_right_matches()
.insert_str()	.split()	.parse()
.len()	.rsplit()	.replace()
.is_empty()	.split_terminator()	.replacen()
.split_off()	.rsplit_terminator()	.to_lowercase()
.clear()	.splitn()	.to_uppercase()
.drain()	.rsplitn()	.repeat()
.is_char_boundary()	.matches()	
.split_at()	.rmatches()	
.split_at_mut()	.match_indices()	
.split_whitespace()	.rmatch_indices()	
.lines()	.trim()	

.nth(3)

*Exercise D*

*Questions?*

*5-min Break*

**PLACEHOLDER**  
**Timer: 5 Minutes**

# Section 7: A Different Mindset

# Ownership

# 3 Rules

1. Each value has an owner

2. Only one owner

3. Value gets dropped if its owner goes out of scope

```
let s1 = String::from("abc");
```

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

```
let s1 = String::from("abc");
let s2 = s1;
println!("{}", s1); // Error!
```

```
error[E0382]: borrow of moved value: `s1`
```

```
--> src/main.rs:4:20
```

```
|  
|     let s2 = s1;  
|             -- value moved here  
4 |     println!("{}", s1); // Error!  
|                         ^^ value borrowed here after move
```

```
= note: move occurs because `s1` has type  
`std::string::String`, which does not implement the `Copy` trait
```

```
error[E0382]: borrow of moved value: `s1`
```

```
--> src/main.rs:4:20
```

```
|  
|     let s2 = s1;  
|             -- value moved here  
|     println!("{}", s1); // Error!  
|                         ^^ value borrowed here after move
```

```
= note: move occurs because `s1` has type  
'std::string::String', which does not implement the `Copy` trait
```

Stack

Heap

Stack

Heap

In order

Stack

Heap

In order

Fixed-size

Stack

Heap

In order

Fixed-size

LIFO

Stack

Heap

In order

Fixed-size

LIFO

Fast

Stack

Heap

In order

Unordered

Fixed-size

LIFO

Fast

Stack

Heap

In order

Unordered

Fixed-size

Variable-size

LIFO

Fast

Stack

Heap

In order

Unordered

Fixed-size

Variable-size

LIFO

Unordered

Fast

Stack

Heap

In order

Unordered

Fixed-size

Variable-size

LIFO

Unordered

Fast

Slow

```
let s1 = String::from("abc");
```

# Stack

s1

ptr	
len	
capacity	

# Heap

# Stack

s1

ptr	
len	
capacity	3

# Heap

# Stack

s1

ptr	
len	3
capacity	3

# Heap

# Stack

s1

ptr	
len	3
capacity	3

# Heap

a
b
c



# Stack

s1

ptr	
len	3
capacity	3

# Heap

a
b
c



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

# Stack

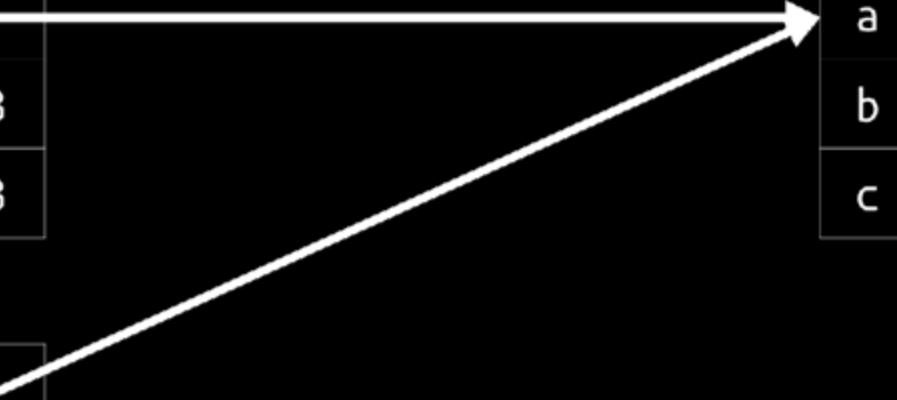
s1

ptr	
len	3
capacity	3

# Heap

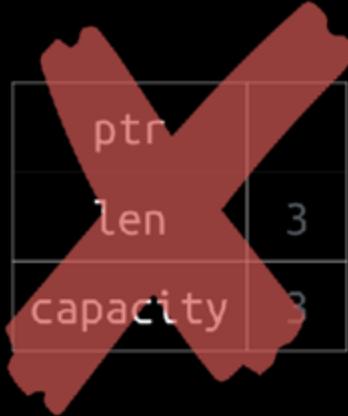
s2

ptr	
len	3
capacity	3



# Stack

s1



ptr	
len	3
capacity	3

s2

ptr	
len	3
capacity	3

# Heap

a
b
c



```
let s1 = String::from("abc");
let s2 = s1;
println!("{}", s1); // Error!
```

```
let s1 = String::from("abc");
let s2 = s1.clone();
println!("{}", s1);
```

```
let s1 = String::from("abc");
let s2 = s1.clone();
println!("{}", s1);
```

# Stack

s1

ptr	
len	3
capacity	3

# Heap

a
b
c



# Stack

s1

ptr	
len	3
capacity	3

# Heap

a
b
c



s2

ptr	
len	3
capacity	3

# Stack

# Heap

s1

ptr	
len	3
capacity	3



a
b
c

s2

ptr	
len	3
capacity	3



a
b
c

# Stack

# Heap

s1

ptr	
len	3
capacity	3



a
b
c

s2

ptr	
len	3
capacity	3



a
b
c

# Stack

s1

ptr	
len	3
capacity	3

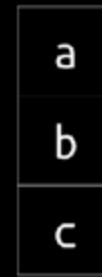
# Heap

a
b
c

- Drop {
1. Destructor
  2. Free heap
  3. Pop stack

# Stack

# Heap



# Stack

# Heap

s1

ptr	
len	3
capacity	3





```
let s1 = String::from("abc");
```

```
let s1 = String::from("abc");

fn do_stuff(s: String) {
    // do stuff
}
```

```
let s1 = String::from("abc");
do_stuff(s1);
```

```
fn do_stuff(s: String) {
    // do stuff
}
```

```
let s1 = String::from("abc");
do_stuff(s1);
println!("{}", s1); // Error, moved!

fn do_stuff(s: String) {
    // do stuff
}
```



```
let mut s1 = String::from("abc");
do_stuff(s1);
println!("{}", s1); // Error!

fn do_stuff(s: String) -> String {
}
```

```
let mut s1 = String::from("abc");
do_stuff(s1);
println!("{}", s1); // Error!

fn do_stuff(s: String) -> String {
    s
}
```

```
let mut s1 = String::from("abc");
s1 = do_stuff(s1);
println!("{}", s1); // Error!

fn do_stuff(s: String) -> String {
    s
}
```

# References & Borrowing

```
let s1 = String::from("abc");
```

```
let s1 = String::from("abc");

fn do_stuff(s: &String) {
    // do stuff
}
```

```
let s1 = String::from("abc");

fn do_stuff(s: &String) {
    // do stuff
}
```

```
let s1 = String::from("abc");
do_stuff(&s1);
```

```
fn do_stuff(s: &String) {
    // do stuff
}
```

```
let s1 = String::from("abc");
do_stuff(&s1);
```

```
fn do_stuff(s: &String) {
    // do stuff
}
```

```
let s1 = String::from("abc");
do_stuff(&s1);
```

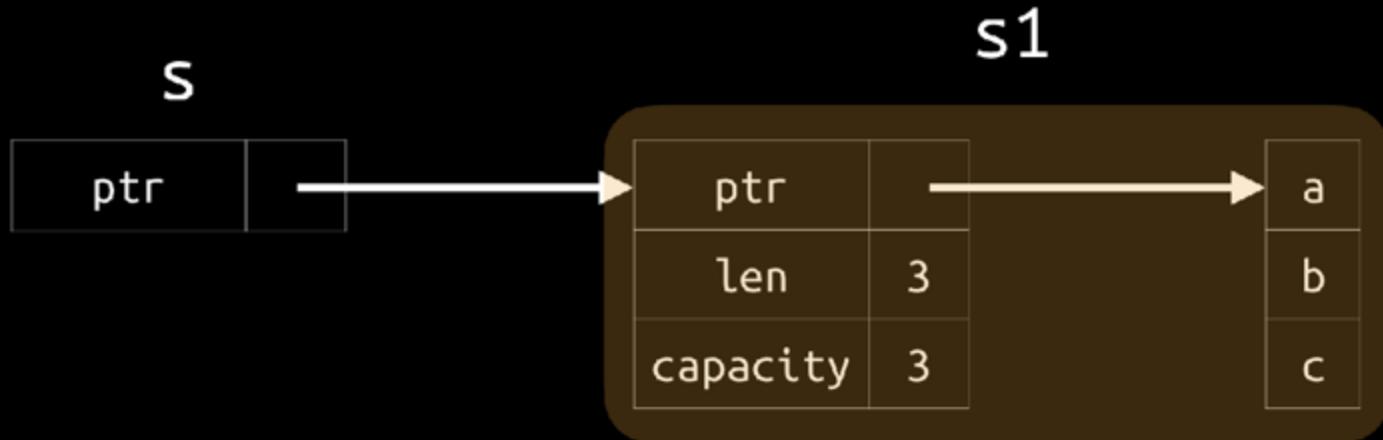
```
fn do_stuff(s: &String) {
    // do stuff
}
```

```
let s1 = String::from("abc");
do_stuff(&s1);
println!("{}", s1);
```

```
fn do_stuff(s: &String) {
    // do stuff
}
```

s1





# Lifetimes

References must always be valid

```
let mut s1 = String::from("abc");
do_stuff(&s1);

fn do_stuff(s: &String) {
    s.insert_str(0, "Hi, ");
}

}
```

```
let mut s1 = String::from("abc");
do_stuff(&mut s1);

fn do_stuff(s: &mut String) {
    s.insert_str(0, "Hi, ");
}
}
```

```
let mut s1 = String::from("abc");
do_stuff(&mut s1);

fn do_stuff(s: &mut String) {
    s.insert_str(0, "Hi, ");
}
}
```

X

&X

&mut x

i32

&i32

&mut i32

You can have **either**

Exactly one mutable reference

Any number of immutable  
references

Thread A



Thread B



Thread C



# Compiler

```
error[E0384]: cannot assign twice to immutable variable `bunnies`
--> src/main.rs:3:5
```

```
2 |     let bunnies = 16;
|     -----
|     |
|     first assignment to `bunnies`
|     help: make this binding mutable: `mut bunnies`
3 |     bunnies = 2; // Error!
|     ^^^^^^^^^^^^ cannot assign twice to immutable variable
```

```
error: aborting due to previous error
```

```
For more information about this error, try `rustc --explain E0384`.
error: Could not compile `hoppity`.
```



```
$ ./my_c_program  
Segmentation fault (core dumped)
```

```
println!("{}", variab);  
          ^^^^^^ did you mean `variable`?
```

```
println!("{}", variab);  
          ^^^^^^ did you mean `variable`?  
Why yes, I did! 💕💕😎
```



*Exercise E*

*Questions?*

# Section 8: Power Tools Part 1

# Structs

```
struct RedFox {  
    enemy: bool,  
    life: u8,  
}
```

```
struct RedFox {  
    enemy: bool,  
    life: u8,  
}
```

```
struct RedFox {  
    enemy: bool,  
    life: u8,  
}
```

```
struct RedFox {  
    enemy: bool,  
    life: u8,  
}
```

```
struct RedFox {  
    enemy: bool,  
    life: u8,  
}
```

```
struct RedFox {  
    enemy: bool,  
    life: u8,  
}
```

```
let fox = RedFox {  
    enemy: true,  
    life: 70,  
};
```

```
impl RedFox {  
    fn new() -> Self {  
        Self {  
            enemy: true,  
            life: 70,  
        }  
    }  
}
```

```
impl RedFox {  
    fn new() -> Self {  
        Self {  
            enemy: true,  
            life: 70,  
        }  
    }  
}
```

```
impl RedFox {  
    fn new() -> Self {  
        Self {  
            enemy: true,  
            life: 70,  
        }  
    }  
}
```

```
impl RedFox {  
    fn new() -> Self {  
        Self {  
            enemy: true,  
            life: 70,  
        }  
    }  
}
```

```
impl RedFox {  
    fn new() -> RedFox {  
        RedFox {  
            enemy: true,  
            life: 70,  
        }  
    }  
}
```

```
let fox = RedFox::new();
```

```
let fox = RedFox::new();
```

```
let fox = RedFox::new();
let life_left = fox.life;
fox.enemy = false;
fox.some_method();
```

```
let fox = RedFox::new();
let life_left = fox.life;
fox.enemy = false;
fox.some_method();
```

```
impl RedFox {  
    // associated function  
    fn function() ...  
    // methods  
    fn move(self) ...  
    fn borrow(&self) ...  
    fn mut_borrow(&mut self) ...  
}
```

```
impl RedFox {  
    // associated function  
    fn function() ...  
    // methods  
    fn move(self) ...  
    fn borrow(&self) ...  
    fn mut_borrow(&mut self) ...  
}
```

# Class Inheritance

# Struct Inheritance



Struct Inheritance

# Object-Oriented?

# Religious War



Religious War

Why?

# Traits

```
struct RedFox {  
    enemy: bool,  
    life: u32,  
}
```

```
struct RedFox {  
    enemy: bool,  
    life: u32,  
}
```

```
trait Noisy {  
    fn get_noise(&self) -> &str;  
}
```

```
struct RedFox {  
    enemy: bool,  
    life: u32,  
}
```

```
trait Noisy {  
    fn get_noise(&self) -> &str;  
}
```

```
struct RedFox {  
    enemy: bool,  
    life: u32,  
}
```

```
trait Noisy {  
    fn get_noise(&self) -> &str;  
}
```

```
struct RedFox {  
    enemy: bool,  
    life: u32,  
}
```

```
trait Noisy {  
    fn get_noise(&self) -> &str;  
}
```

```
struct RedFox {  
    enemy: bool,  
    life: u32,  
}
```

```
trait Noisy {  
    fn get_noise(&self) -> &str;  
}
```

```
struct RedFox {  
    enemy: bool,  
    life: u32,  
}
```

```
trait Noisy {  
    fn get_noise(&self) -> &str;  
}
```

```
impl Noisy for RedFox {  
    fn get_noise(&self) -> &str { "Meow?" }  
}
```

```
struct RedFox {  
    enemy: bool,  
    life: u32,  
}
```

```
trait Noisy {  
    fn get_noise(&self) -> &str;  
}
```

```
impl Noisy for RedFox {  
    fn get_noise(&self) -> &str { "Meow?" }  
}
```

```
struct RedFox {  
    enemy: bool,  
    life: u32,  
}
```

```
trait Noisy {  
    fn get_noise(&self) -> &str;  
}
```

```
impl Noisy for RedFox {  
    fn get_noise(&self) -> &str { "Meow?" }  
}
```

```
struct RedFox {  
    enemy: bool,  
    life: u32,  
}
```

```
trait Noisy {  
    fn get_noise(&self) -> &str;  
}
```

```
impl Noisy for RedFox {  
    fn get_noise(&self) -> &str { "Meow?" }  
}
```

```
struct RedFox {  
    enemy: bool,  
    life: u32,  
}
```

```
trait Noisy {  
    fn get_noise(&self) -> &str;  
}
```

```
impl Noisy for RedFox {  
    fn get_noise(&self) -> &str { "Meow?" }  
}
```

```
struct RedFox {  
    enemy: bool,  
    life: u32,  
}
```

```
trait Noisy {  
    fn get_noise(&self) -> &str;  
}
```

```
impl Noisy for RedFox {  
    fn get_noise(&self) -> &str { "Meow?" }  
}
```

```
struct RedFox {  
    enemy: bool,  
    life: u32,  
}  
  
impl RedFox {  
    fn get_noise(&self) -> &str { "Meow?" }  
}
```

```
fn print_noise<T: Noisy>(item: T) {  
    println!("{}{}", item.get_noise());  
}
```

```
fn print_noise<T: Noisy>(item: T) {  
    println!("{}{}", item.get_noise());  
}
```

```
fn print_noise<T: Noisy>(item: T) {  
    println!("{}{}", item.get_noise());  
}
```

```
fn print_noise<T: Noisy>(item: T) {  
    println!("{}{}", item.get_noise());  
}
```

```
fn print_noise<T: Noisy>(item: T) {  
    println!("{}{}", item.get_noise());  
}
```

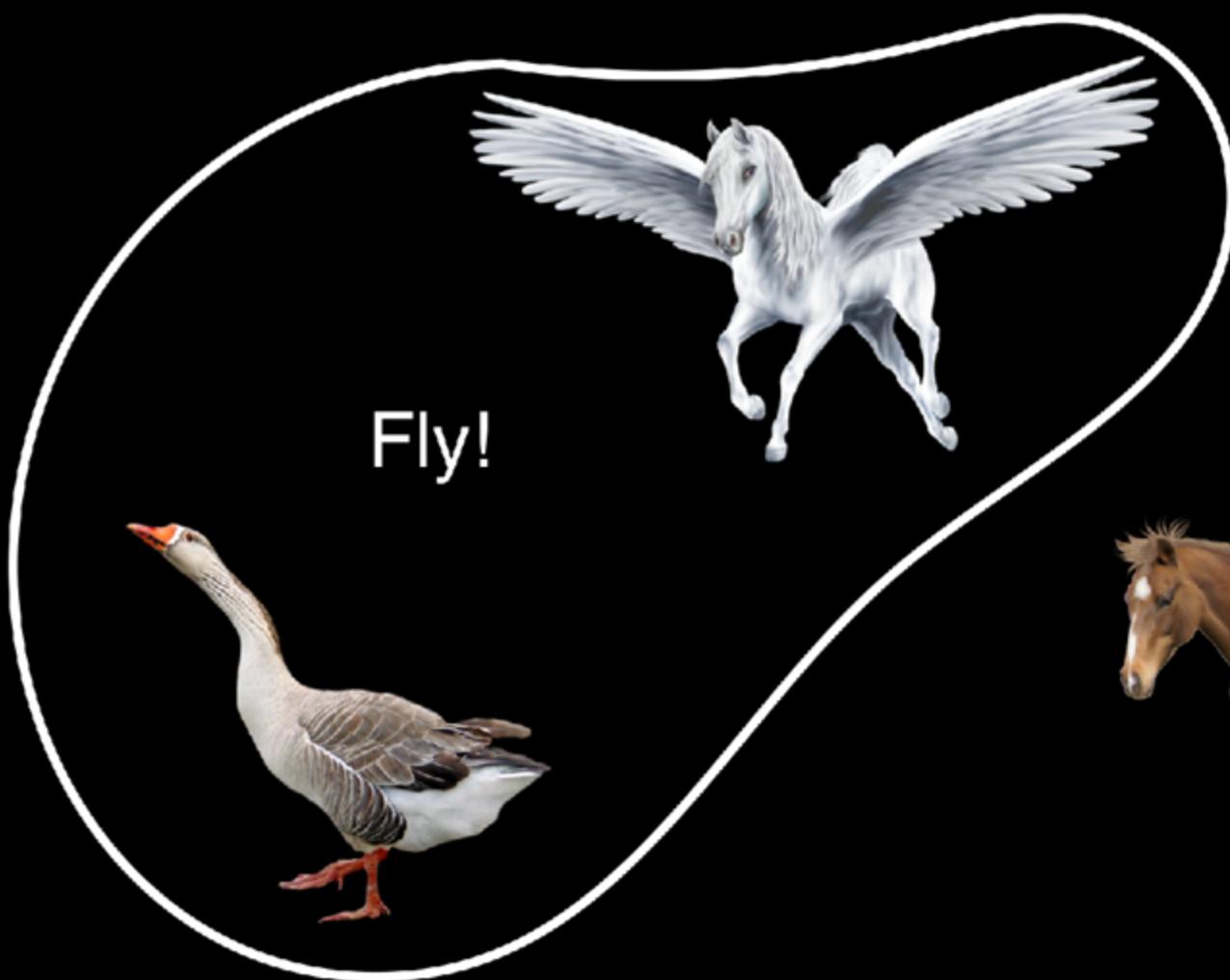
```
fn print_noise<T: Noisy>(item: T) {  
    println!("{}{}", item.get_noise());  
}  
  
impl Noisy for u8 {  
    fn get_noise(&self) -> &str { "BYTE!" }  
}  
  
fn main() {  
    print_noise(5_u8); // prints "BYTE!"  
}
```

```
fn print_noise<T: Noisy>(item: T) {  
    println!("{}: {}", item, item.get_noise());  
}  
  
impl Noisy for u8 {  
    fn get_noise(&self) -> &str { "BYTE!" }  
}  
  
fn main() {  
    print_noise(5_u8); // prints "BYTE!"  
}
```

```
fn print_noise<T: Noisy>(item: T) {  
    println!("{}: {}", item, item.get_noise());  
}  
  
impl Noisy for u8 {  
    fn get_noise(&self) -> &str { "BYTE!" }  
}  
  
fn main() {  
    print_noise(5_u8); // prints "BYTE!"  
}
```

*Copy*





Fly!





Ridden





Explode







Fly  
Explode



Fly  
Ride



Fly  
Explode





Fly  
Explode



Fly  
Ride



Ride  
Explode



Fly  
Explode



Fly  
Ride



Ride  
Explode

# Inheritance

Movement



Run



Ride

Fly

Movement



Run

Damage



Explode



Ride

Fly

Horse

Movement

Damage



Run



Explode



Ride

Fly

Horse  
Ride  
Explode

Movement



Damage



Run

Ride Fly

Horse  
Movement  
Run  
Ride  
Damage  
Explode

Movement



Run



Ride

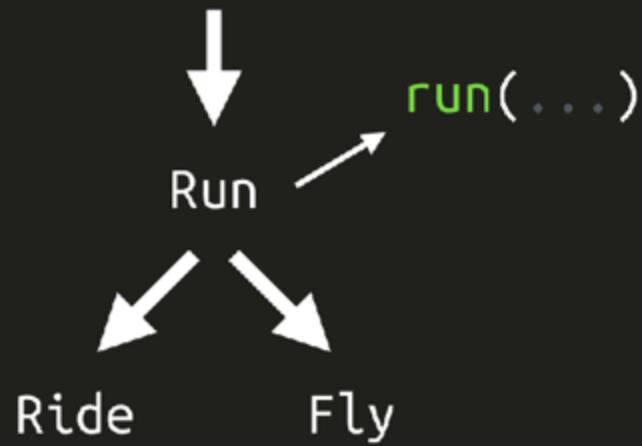
Fly

Damage



Explode

## Movement



Robot

Run

Ostrich

Run

Soldier

Run

Child

Run

Movement



Run



Fly

```
trait Run {  
    fn run(&self);  
}
```

```
trait Run {
    fn run(&self) {
        println!("I'm running!");
    }
}
```

```
trait Run {  
    fn run(&self) {  
        println!("I'm running!");  
    }  
}
```

```
struct Robot {}  
impl Run for Robot {}
```

```
trait Run {  
    fn run(&self) {  
        println!("I'm running!");  
    }  
}
```

```
struct Robot {}  
impl Run for Robot {}
```

```
fn main() {  
    let robot = Robot {};  
    robot.run();  
}
```

# No Fields

# *Exercise F*

*Questions?*

*5-min Break*

# Section 9: Power Tools Part 2

# Collections

Vec<T>

```
let mut v: Vec<i32> = Vec::new();
```

```
let mut v: Vec<i32> = Vec::new();
```

```
let mut v: Vec<i32> = Vec::new();
```

```
let mut v: Vec<i32> = Vec::new();
v.push(2);
v.push(4);
v.push(6);
```

```
let mut v: Vec<i32> = Vec::new();
v.push(2);
v.push(4);
v.push(6);
let x = v.pop();           // x is 6
```

```
let mut v: Vec<i32> = Vec::new();
v.push(2);
v.push(4);
v.push(6);
let x = v.pop();           // x is 6
println!("{}", v[1]); // prints "4"
```

```
let mut v = vec![2, 4, 6];
```

HashMap<K, V>

VecDeque  
LinkedList

HashSet  
BinaryHeap

BTreeMap  
BTreeSet

**VecDeque**  
**LinkedList**

**HashSet**  
**BinaryHeap**

**BTreeMap**  
**BTreeSet**

VecDeque  
LinkedList

HashSet  
BinaryHeap

BTreeMap  
BTreeSet

# Enums

# Algebraic Data Types

```
enum Color {  
    Red,  
    Green,  
    Blue,  
}
```



```
enum Color {  
    Red,  
    Green,  
    Blue,  
}
```

```
enum Color {  
    Red,  
    Green,  
    Blue,  
}
```

```
enum Color {  
    Red,  
    Green,  
    Blue,  
}
```

```
let color = Color::Red;
```

```
enum DispenserItem {  
    Zero,  
}
```

```
enum DispenserItem {  
    Zero,  
    One(u8),  
}
```

```
enum DispenserItem {  
    Zero,  
    One(u8),  
    Two(String, i32),  
}
```

```
enum DispenserItem {
    Zero,
    One(u8),
    Two(String, i32),
    Stuff {x: i32, y: i32},
}
```

```
enum DispenserItem {
    Zero,
    One(u8),
    Two(String, i32),
    Stuff {x: i32, y: i32},
}
```

```
use DispenserItem::*;

let item = Zero;
```

```
enum DispenserItem {
    Zero,
    One(u8),
    Two(String, i32),
    Stuff {x: i32, y: i32},
}
```

```
use DispenserItem::*;

let item = One(69);
```

```
enum DispenserItem {
    Zero,
    One(u8),
    Two(String, i32),
    Stuff {x: i32, y: i32},
}

use DispenserItem::*;

let item = Two("q".to_string(), 7);
```

```
enum DispenserItem {
    Zero,
    One(u8),
    Two(String, i32),
    Stuff {x: i32, y: i32},
}

use DispenserItem::*;

let item = Stuff { x: 24, y: 48 };
```

```
enum DispenserItem {
    Zero,
    One(u8),
    Two(String, i32),
    Stuff {x: i32, y: i32},
}
```

```
impl DispenserItem {
    fn display(&self) { }
}
```

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

```
if let Some(x) = my_variable {  
    println!("value is {}", x);  
}
```

```
if let Some(x) = my_variable {  
    println!("value is {}", x);  
}
```

```
if let Some(x) = my_variable {  
    println!("value is {}", x);  
}
```

```
if let Some(x) = my_variable {  
    println!("value is {}", x);  
}
```

```
if let Some(x) = my_variable {  
    println!("value is {}", x);  
}
```

```
match my_variable {
    Some(x) => {
        println!("value is {}", x);
    },
    None => {
        println!("no value");
    },
}
```

```
match my_variable {
    Some(x) => {
        println!("value is {}", x);
    },
    None => {
        println!("no value");
    },
}
```

```
match my_variable {
    Some(x) => {
        println!("value is {}", x);
    },
    None => {
        println!("no value");
    },
}
```

```
match my_variable {
    Some(x) => {
        println!("value is {}", x);
    },
    None => {
        println!("no value");
    },
}
```

```
match my_variable {  
    Some(x) => {  
        println!("value is {}", x);  
    },  
    None => {  
        println!("no value");  
    },  
}
```

```
match my_variable {
    Some(x) => {
        println!("value is {}", x);
    },
    None => {
        println!("no value");
    },
}
```

```
match my_variable {
    Some(x) => {
        println!("value is {}", x);
    },
    None => {
        println!("no value");
    },
}
```

```
match my_variable {  
    _ => {  
        println!("who cares");  
    },  
}
```

```
match my_variable {
    Some(x) => {
        println!("value is {}", x);
    },
    None => {
        println!("no value");
    },
}
```

```
match my_variable {  
    Some(x) => 42,  
    None => 43,  
}
```

```
let x = match my_variable {  
    Some(x) => 42,  
    None => 43,  
};
```

# *Exercise G*

*Questions?*

*5-min Break*

# Section 10: Option & Result

Option

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

```
let mut x: Option<i32> = None;
```

```
let mut x: Option<i32> = None;  
x = Some(5);
```

```
let mut x = None;  
x = Some(5);
```

```
let mut x = None;  
x = Some(5);  
x.is_some(); // true
```

```
let mut x = None;  
x = Some(5);  
x.is_some(); // true  
x.is_none(); // false
```

```
let mut x = None;  
x = Some(5);  
x.is_some(); // true  
x.is_none(); // false  
for i in x {  
    println!("{}", i); // prints 5  
}
```

```
let mut x = None;  
x = Some(5);  
x.is_some(); // true  
x.is_none(); // false  
for i in x {  
    println!("{}", i); // prints 5  
}
```

# Result

```
#[must_use]
enum Result<T, E> {
    Ok(T),
    Err(E),
}
```

```
use std::fs::File;
fn main() {
    File::open("foo");
}
```

```
warning: unused `std::result::Result` that must be used
--> src/main.rs:3:5
3 |     File::open("foo");
   | ^^^^^^^^^^^^^^^^^^
= note: #[warn(unused_must_use)] on by default
= note: this `Result` may be an `Err` variant, which should be handled
```

```
use std::fs::File;

fn main() {
    let res = File::open("foo");
    let f = res.unwrap();
}
```

```
use std::fs::File;

fn main() {
    let res = File::open("foo");
    let f = res.expect("message");
}
```

```
use std::fs::File;

fn main() {
    let res = File::open("foo");
    if res.is_ok() {
        let f = res.unwrap();
    }
}
```

```
use std::fs::File;

fn main() {
    let res = File::open("foo");
    match res {
        Ok(f) => { /* do stuff */ },
        Err(e) => { /* do stuff */ },
    }
}
```

# Section 11: The Final Project

# Final Project! *(Exercise 2)*



Now what?

# Some Things We Didn't Cover

Attributes	Embedded	Panics
Benchmarks	FFI (C ABI)	Publishing crates
Best practices	Generics	Smart Pointers
Channels	Libraries	Testing
Closures	Lifetimes	Tracebacks
Cross-compiling	Macros	Trait objects
Custom builds	Memory allocators	WASM
Debugging	Modules	Workspaces
Dependencies	Multithreading	
Documentation	Mutexes	<i>...and more!</i>

Thank You

