Rust Workshop

Day 1

Why do people use Rust at Work?

according to the Rust Annual Survey 2023

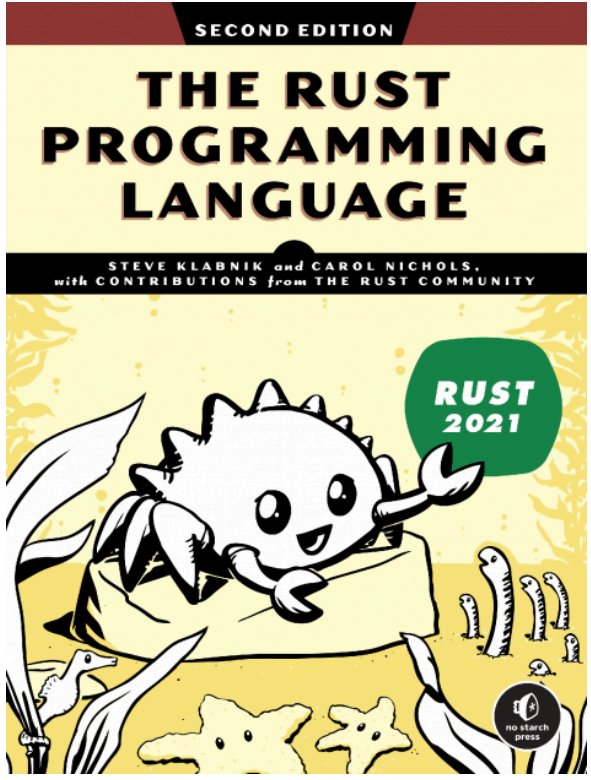
- (85%) Rust allows us to build relatively correct and bug free software
- (83%) For Rust's performance (speed, memory footprint) characteristics
- (70%) Its security and safety properties are important to us
- (70%) We find it enjoyable or fun to program in Rust

Goal

You are confident to start new projects in Rust where it is the best technical fit.

Program

Day 1	Language Basics 1 🤔	common programming concepts & ownership
Day 2	Language Basics 2 🤓	structs, enums, modules, collections, error handling
Day 3	Advanced Features 🤯	generics, traits, lifetimes, closures, iterators
Day 4	The Rust Ecosystem 📦	libraries, documentation, tools, news, CI/CD
Day 5	Shippable Projects 🚀	CLI tools, web APIs, python modules, WASM apps
Day 6	Wrap-Up ✅	finish projects, questions, feedback

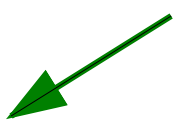

SECOND EDITION
THE RUST PROGRAMMING LANGUAGE
STEVE KLABNIK and CAROL NICHOLS,
with CONTRIBUTIONS from THE RUST COMMUNITY
RUST 2021
no starch press

About these slides

Although they aren't meant as reference material...
Both the interactive version and PDF exports are available at:

senekor.github.io/rust-workshop

Using the interactive slides:
Hover over the bottom-left corner for controls.



Language Basics 1

book chapters 3 & 4

common programming concepts

ownership

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

Variables

book chapter 3.1

Variable Declaration

```
1  let x = 5;
2
3  let x: i32 = 5;
```

Variable Declaration

```
1  let x = 5;
2
3  let x: i32 = 5;
```

Mutability

```
let x = 5;
x = 6; // error: cannot assign twice to immutable variable `x`

let mut x = 5;
x = 6; // ✅
```

Mutability

```
let x = 5;
x = 6; // error: cannot assign twice to immutable variable `x`

let mut x = 5;
x = 6; // ✅
```

Globals

```
1  // "copy-pasted" everywhere (like C's #define)
2  const THREE_HOURS_IN_SECONDS: u32 = 60 * 60 * 3;
3
4  // placed in static memory (text or data segment)
5  static EMBEDDED_TEXT_FILE: &str = include_str!("path/to/some/file.txt");
```

Globals

```
1  // "copy-pasted" everywhere (like C's #define)
2  const THREE_HOURS_IN_SECONDS: u32 = 60 * 60 * 3;
3
4  // placed in static memory (text or data segment)
5  static EMBEDDED_TEXT_FILE: &str = include_str!("path/to/some/file.txt");
```

Scope

```
let x = 5;
{
    let y = 6;
    // x and y available
}
// only x available
```

Basic Types

book chapter 3.2

Integer Types

length	signed	unsigned
8-bit	`i8`	`u8`
16-bit	`i16`	`u16`
32-bit	`i32`	`u32`
64-bit	`i64`	`u64`
128-bit	`i128`	`u128`
arch	`isize`	`usize`

Number Literals

Decimal	`98_222`
Hex	`0xff`
Octal	`0o77`
Binary	`0b1111_0000`
ASCII Byte	`b'A'`
with type suffix	`57_i64`

Floating-point Types

IEEE-754

```
1  let x = 2.0; // default: 64-bit
2  let y: f32 = 3.0; // 32-bit
```

Booleans

```
1  let x = true;
2  let y: bool = false;
```

Characters

unicode, guaranteed 32-bit

```
1  // notice the single quotes
2  let x = 'a';
3  let y: char = '?';
4  let heart_eyed_cat = '😻';
```

Tuples

```
1  let tup: (i32, f64, u8) = (500, 6.4, 1);
2
3  let (x, y, z) = tup;
4  let a: i32 = tup.0;
```

Tuples

```
let tup: (i32, f64, u8) = (500, 6.4, 1);

let (x, y, z) = tup;
let a: i32 = tup.0;
```

Tuples

```
1  let tup: (i32, f64, u8) = (500, 6.4, 1);
2
3  let (x, y, z) = tup;
4  let a: i32 = tup.0;
```

The Empty Tuple

also known as the "unit"

```
1  let rusty_void: () = println!("printing doesn't return anything");
```

Arrays

size known at compile time

```
1  let a: [i32; 5] = [1, 2, 3, 4, 5];
2  let a = [3; 5]; // == [3, 3, 3, 3, 3]
3  let x = a[0];
```

Arrays

size known at compile time

```
let a: [i32; 5] = [1, 2, 3, 4, 5];
let a = [3; 5]; // == [3, 3, 3, 3, 3]
let x = a[0];
```

Arrays

size known at compile time

```
let a: [i32; 5] = [1, 2, 3, 4, 5];
let a = [3; 5]; // == [3, 3, 3, 3, 3]
let x = a[0];
```

Functions

book chapter 3.3

Basic Function

```
1  fn main() {
2      another_function();
3  }
4
5  fn another_function() {
6      println!("Hello from another function!");
7  }
```

Parameters

```
1  fn print_labeled_measurement(value: i32, unit_label: char) {
2      println!("The measurement is: {}{}", value, unit_label);
3  }
4
5  fn main() {
6      print_labeled_measurement(5, 'h');
7  }
```

Expressions

A block is an expression where the last expression of the block becomes the value of the entire block.

```
let y = {
    let x = 3;
    x + 1  // <- note the lacking semicolon
};
// y == 4
```

Return Values

```
fn plus_one(x: i32) -> i32 {
    if x == i32::MAX {
        return i32::MAX; // no overflow
    }
    x + 1  // <- last expression of block
}
```

Interlude: Macros

General

Rust macros are a meta-programming feature like the C preprocessor.
Unlike in C, macros operate on *tokens* instead of text.

Rust macros use very specific syntax, so you can identify them easily.

"function-like" macros

```
1  let name = "Joe";
2  let age = 36;
3  println!("My friend {} is {} years old.", name, age);
```

These macros are identified by the exclamation mark.

The tokens within the parentheses are the inputs to the macro.
(string literal, comma, identifier, comma, identifier)

The output of the macro is the actual code
necessary to print the formatted string.

"attribute-like" macros

```
#[my_attribute_macro]
fn add(a: i32, b: i32) -> i32 {
    a + b
}
const PI: usize = 3 // close enough
```

These macros are identified by the `#[]` syntax.

The tokens of the item immediately after the macro are its input.
That includes the entire function definition of `add`,
but NOT the definition of `PI`.

`my_attribute_macro` might output additional code that's related
to `add` or even modify the function itself.
However, it cannot generate code based on `PI`,
since it doesn't know about it.

Control Flow

book chapter 3.5

`if` Expressions

no parentheses around condition, curly brackets mandatory

```
1  let number = 3;
2
3  let size = if number < 5 {
4      "small"
5  } else if number < 10 {
6      "big"
7  } else {
8      "very big"
9  };
```

What is the type of `size` ?

`loop`

`continue` and `break` work as expected

```
1  loop {
2      println!("computer go brrr");
3  }
```

`while` Loops

```
let mut number = 10;

while number != 0 {
    println!("{}!", number);

    number -= 1;
}

println!("LIFTOFF!!!");
```

`for` Loops

more details on day 3

```
let a = [10, 20, 30, 40, 50];

// `..` is the range operator
for i in 0..a.len() {
    let element = a[i];
    println!("the value is: {}", element);
}

for element in a {
    println!("the value is: {}", element);
}
```

Some Operators

some assignment variants exist (`+=`)

Comparison	`==` `!=` `<` `<=` `>` `>=`
Arithmetic	`+` `-` `*` `/` `%`
Boolean	`&&` `||` `!`
Bitwise	`&` `|` `^` `!` (no tilde!)
Range	`..` `..=` (integers and `char`)

Integer Conversions

`as` exists, it but has some footguns

```
1  // infallible
2  let x: i32 = 42_i16.into();
3
4  // fallible
5  let x: u32 = 42_i64.try_into().unwrap();
```

Ownership

book chapter 4.1

Memory Management

approach	properties	
manual	fast & predictable, but unsafe	✅❌
garbage-collection	slow & unpredictable, but safe	❌✅
ownership	fast, predictable, safe and expressive	✅✅✅

C and C++

a short history of manual memory management

Double free

```
1  int *p = malloc(sizeof(int));
2  free(p);
3  free(p); // ⚠
```

Use after free

```
1  int *p = malloc(sizeof(int));
2  free(p);
3  *p = 12; // ⚠
```

Implicit ownership in C

```
1  some_t *foo(some_t *p);
```

- Is the function going to free the pointer, or do I have to?
- Does the function only read from the pointer or does it write to it?
- Can the return value alias the argument?
- Where is the documentation?
- Please let there be documentation.

C++

tools to express ownership

```
1  std::unique_ptr<some_t> foo(some_t const* p);
```

and destructors!

...but no compiler guarantees.

Rust

codify and enforce the rules of ownership

Ownership Rules

1. Every value has exactly one owner.
2. When the owner goes out of scope, the destructor is run.

Heap-allocated Strings

```
let embedded_in_binary: &str = "hello"; // immutable

{
    let mut heap_allocated: String = String::from("hello");
    heap_allocated.push_str(", world!");

} // drop(heap_allocated)
```

One Owner

```
1  let s1 = String::from("hello");
2  let s2 = s1;
3
4  println!("{}, world!", s1); // error
```

Reading the error message

```
error[E0382]: borrow of moved value: `s1`
 --> src/main.rs:5:28
  |
2 |     let s1 = String::from("hello");
  |         -- move occurs because `s1` has type String, which doesn't implement the Copy trait
3 |     let s2 = s1;
  |              -- value moved here
4 |
5 |     println!("{}, world!", s1);
  |                            ^^ value borrowed here after move
  |
  = note: this error originates in the macro `$crate::format_args_nl` which comes from ...
help: consider cloning the value if the performance cost is acceptable
  |
3 |     let s2 = s1.clone();
  |                ++++++++

For more information about this error, try `rustc --explain E0382`.
error: could not compile `ownership` due to previous error
```

Reading the error message

```
1  let s1 = String::from("hello");
2  let s2 = s1;
3
4  println!("{}, world!", s1); // error
```

- borrow of moved value: `s1`
- move occurs because `s1` has type `String`, which does not implement the `Copy` trait
- help: consider cloning the value if the performance cost is acceptable

Scope and Destructors

demo

Ownership and Functions

```
1  // recall: Foo's destructor prints "drop!"
2
3  fn main() {
4      let x = Foo;
5      take_foo(x);
6      println!("Hello, world!");
7  }
8  fn take_foo(arg: Foo) {
9      // empty function body
10 }
```

What's the output of this program?

Ownership is expressive

file handles, mutexes etc.
ownership applies to all kinds of resources

```
fn foo(m: &Mutex<i32>, random_choice: bool) -> Option<MutexGuard<i32>> {
    let guard: MutexGuard<i32> = m.lock().unwrap();

    if random_choice {
        Some(guard)
    } else {
        None
    }
}
```

Limitations

```rust
fn calculate_length(s: String) -> (String, usize) {
    let length = s.len(); // len() returns the length of a String
    (s, length)
}
fn main() {
    let s1 = String::from("hello");
    let (s2, len) = calculate_length(s1);
    println!("The length of '{}' is {}.", s2, len);
}
```

References and Borrowing

book chapter 4.2

What are references?

- basically, pointers with seat belts
- cannot be null
- guaranteed to point to valid memory

Syntax

```
1  let x = 42;
2  let r: &i32 = &x;
3  let y: i32 = *r;
```

Fixing the earlier example

```
fn calculate_length(s: &String) -> usize {
    s.len()
    // s goes out of scope, but its destructor is not run.
}
fn main() {
    let s1 = String::from("hello");
    let len = calculate_length(&s1);
    println!("The length of '{}' is {}.", s1, len);
}
```

Mutable References

```
1  let mut x = 42;
2  let r = &mut x;
3  *r = 43;
```

Mutable References

demo

Mutable references are exclusive

```
1  let mut x = 12;
2
3  let r1 = &mut x;
4  let r2 = &mut x; // error
5
6  *r1 = 13;
```

compiler says:

```
cannot borrow  x  as mutable more than once at a time
```

Mutable references are exclusive

```
1  let mut x = 12;
2
3  let r1 = &mut x;
4  let r2 = &mut x; // error
5
6  *r1 = 13;
```

compiler says:

> cannot borrow `x` as mutable more than once at a time

Mutable references are exclusive

```
1  let mut x = 12;
2
3  let r1 = &mut x;
4  let r2 = &x;      // error
5
6  *r1 = 13;
```

compiler says:

> cannot borrow `x` as immutable because it is also borrowed as mutable

Dangling References

```
1  let r;
2  {
3      let s = String::from("hello");
4      r = &s;
5  }
6  println!("{}", r); // error
```

compiler says:

```
s   does not live long enough
```

Borrrowing Rules

1. At any given time, you can have either one mutable reference or any number of immutable references.
2. References must always be valid.

Bending the Rules

book chapter 15

This is advanced and not needed in most Rust programs.

```
let r1 = Rc::new(RefCell::new(String::from("Hello")));
let r2 = Rc::clone(&r1);

// both owners can do interleaved mutations!
r1.borrow_mut().push_str(" Venus");
r2.borrow_mut().push_str(", Mars");
r1.borrow_mut().push_str(" and Jupiter!");

println!("{}", r1.borrow()); // -> Hello Venus, Mars and Jupiter!
```

`Rc` : reference counting

`RefCell` : interior mutability

The Slice Type

book chapter 4.3

Slices in C ?

```c
void print_slice(int *start, size_t len) {
    for (size_t i = 0; i < len; i++) {
        printf("%d ", start[i]);
    }
}
void main() {
    int numbers[5] = {1, 2, 3, 4, 5};
    print_slice(numbers + 1, 3);  // 2 3 4
    print_slice(numbers + 3, 10); // ⚠
}
```

Start pointer and length are disconnected,
the compiler cannot reason about memory safety.
➔ buffer overflow

Rust Slices

```rust
fn print_int_list(list: &[i32]) {
    for elem in list {
        print!("{} ", elem);
    }
}
fn main() {
    let numbers: [i32; 5] = [1, 2, 3, 4, 5];
    print_int_list(&numbers[1..4]); // 2 3 4
    print_int_list(&numbers[3..13]); // panic: index out of range
}
```

Rust slices store their length alongside the start pointer.
The full length of a slice is guaranteed valid memory.
➔ no buffer overflow

The String Slice

```
1  let owned = String::from("Hello, world!");
2  let s: &str = &owned[3..9]; // "lo, wo"
3
4  // Range boundaries must be valid UTF-8 offsets!
5  let s: &str = &"😱"[1..];
```

computer says:

> byte index 1 is not a char boundary; it is inside '😱' (bytes 0...4) of 😱

Borrowing rules apply to slices

```
let mut owned = String::from("hello");
let s: &str = &owned[2..];
owned.pop(); // error: cannot borrow as mutable
println!("{}", s);
```

String Literals

```
1  let greeting: &str = "Hello, world!";
```

Off-Topic: Vectors

needed for exercises

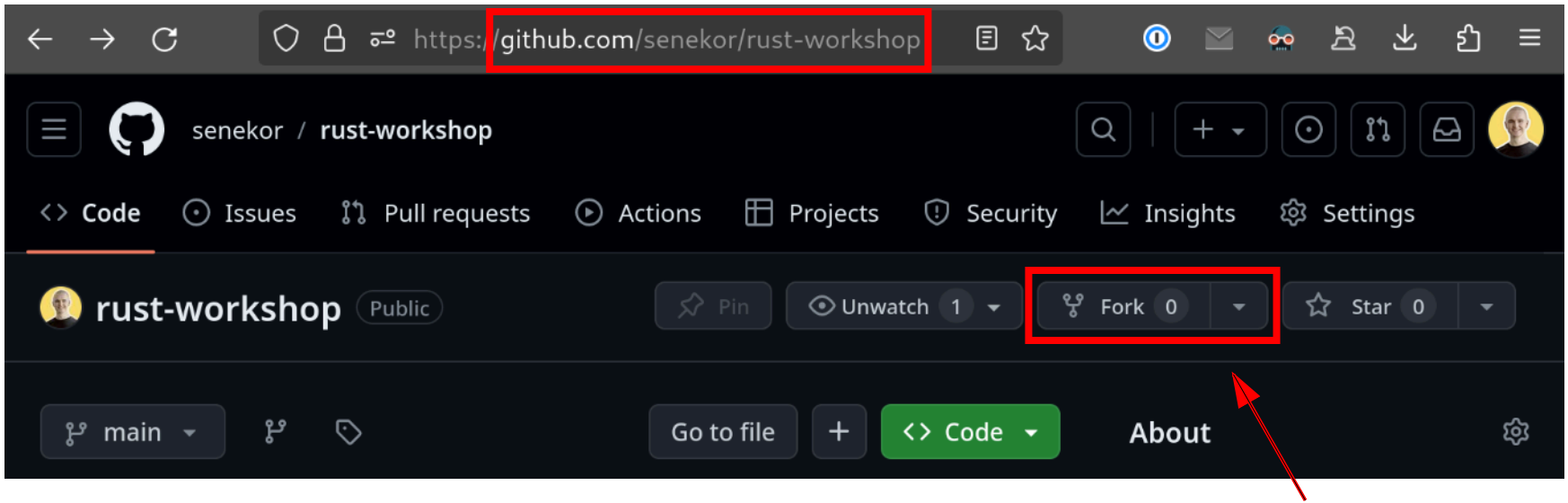
```
let v: Vec<f64> = Vec::new(); // create empty, elems of type f64

let mut v = vec![1, 2, 3]; // macro for "vector literals"

v.push(4);
assert_eq!(v.pop().unwrap(), 4); // `unwrap` because `pop` might return "nothing"
```

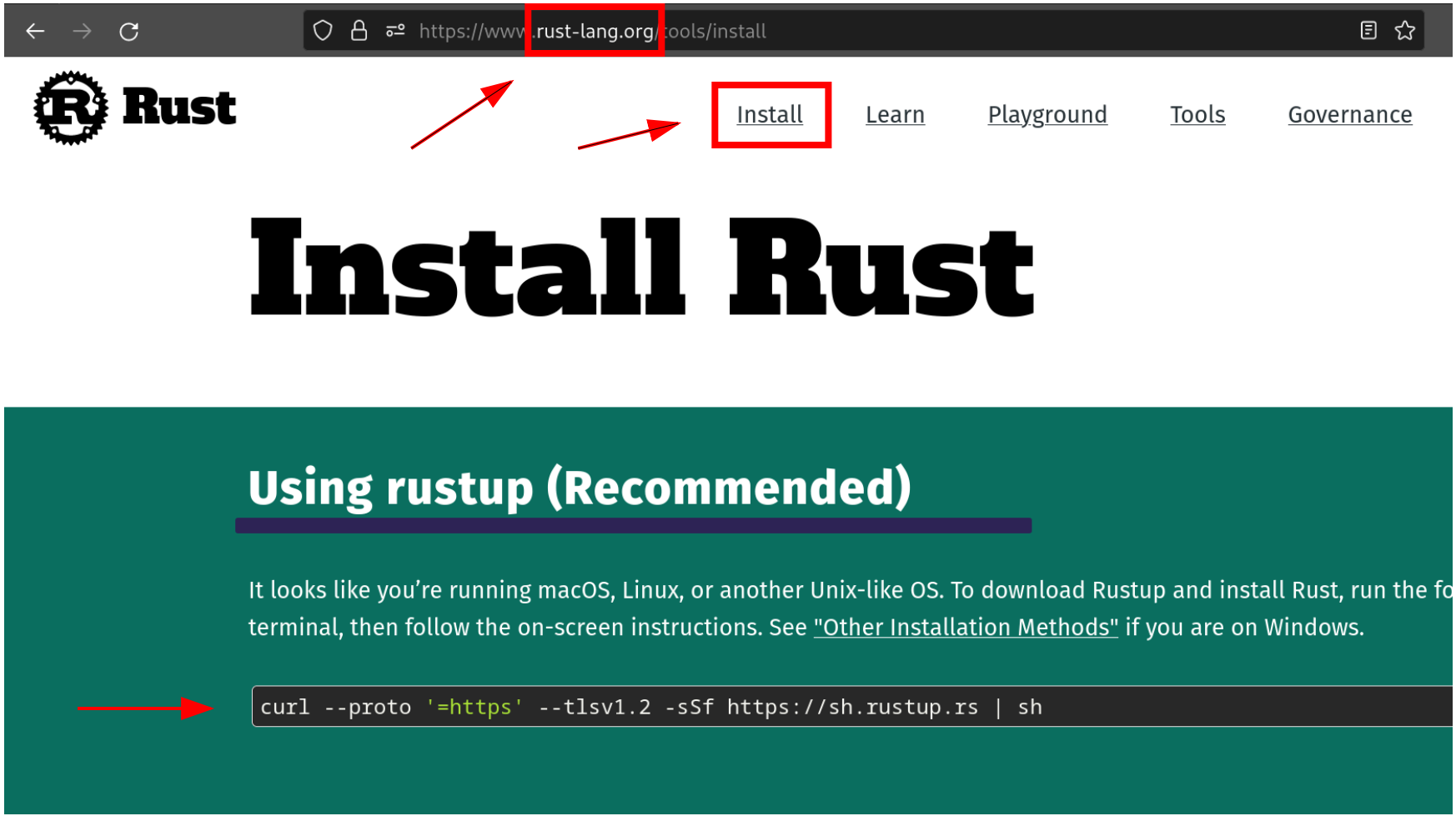
Practice

Ready your laptops!

I will quickly explain all the setup steps.

You'll receive step-by-step instructions in writing as well.


https://github.com/senekor/rust-workshop
senekor / rust-workshop
Code
Issues
Pull requests
Actions
Projects
Security
Insights
Settings
rust-workshop
Public
Pin
Unwatch 1
Fork 0
Star 0
main
Go to file
Code
About


https://www.rust-lang.org/tools/install
Rust
Install
Learn
Playground
Tools
Governance
Install Rust
Using rustup (Recommended)
It looks like you're running macOS, Linux, or another Unix-like OS. To download Rustup and install Rust, run the fo
terminal, then follow the on-screen instructions. See "Other Installation Methods" if you are on Windows.
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh

Choose your `$PATH`

should be handled by the install script

Make sure you you can call `cargo`.
If you can't, try some of these:

```
# POSIX shell
source ~/.cargo/env
# or
export PATH="$HOME/.cargo/bin:$PATH"
```

```
# fish
fish_add_path ~/.cargo/bin
```

Batteries included

component	purpose	example
rustup	toolchain manager	`rustup update`
cargo	package manager	`cargo add my-fav-library`
rustc	compiler	`cargo run` , `cargo build`
rustdoc	documentation generator	`cargo doc --open`
rustfmt	formatter	`cargo fmt`
clippy	linter	`cargo clippy`
rust-analyzer	LSP implementation	N/A

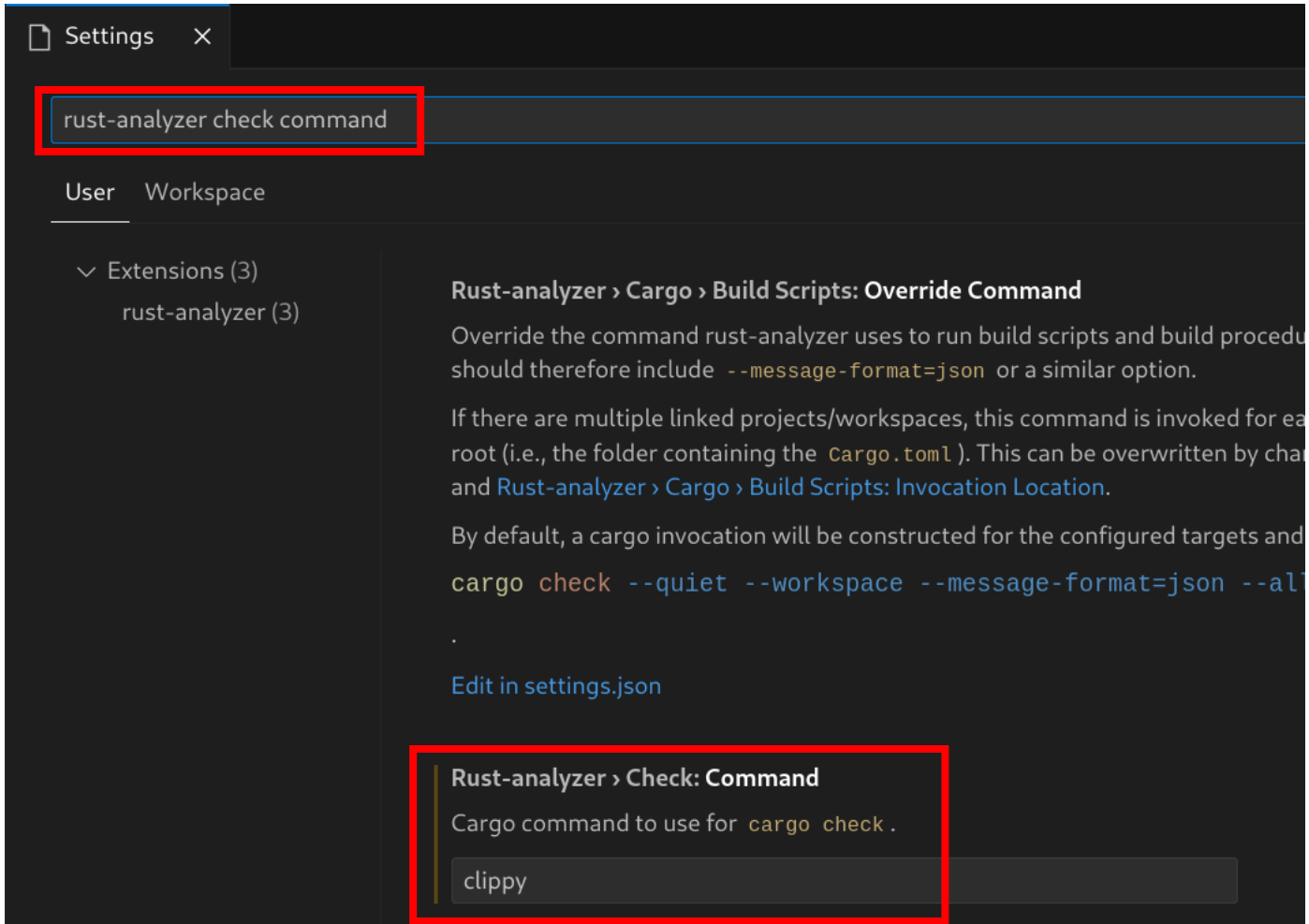
Ensure you have a linker

System	install command
Debian-based	`sudo apt install gcc`
MacOS	`xcode-select --install`

Visual Studio Code Extensions

recommendations cover:

- syntax-highlighting
- autocomplete
- diagnostics
- debugging
- `toml` syntax-highlighting


Settings
rust-analyzer check command
User
Workspace
Extensions (3)
rust-analyzer (3)
Rust-analyzer › Cargo › Build Scripts: Override Command
Override the command rust-analyzer uses to run build scripts and build procedu
should therefore include --message-format=json or a similar option.
If there are multiple linked projects/workspaces, this command is invoked for ea
root (i.e., the folder containing the Cargo.toml). This can be overwritten by cha
and Rust-analyzer › Cargo › Build Scripts: Invocation Location.
By default, a cargo invocation will be constructed for the configured targets and
cargo check --quiet --workspace --message-format=json --al
.
Edit in settings.json
Rust-analyzer › Check: Command
Cargo command to use for cargo check .
clippy

Exercise Organization

custom exercises	in the repo you forked
rustlings	download a folder, e.g. inside fork
Exercism	online editor or need to configure CLI
Advent of Code	locally wherever you want

Practice 

github.com/senekor/rust-exerices

`rust-exerices/day_1/README.md`

Please suggest improvements for next week!

Check the readme of your repository for the form link.