- ✓ let · mut · Types
- ✓ Ownership & Borrowing
- ✓ String vs &str
- ✓ Structs & impl
- ✓ Enums · Option · match
- ✓ Result & ?
- ✓ Traits & Generics
- ✓ Fearless Concurrency

# Rust

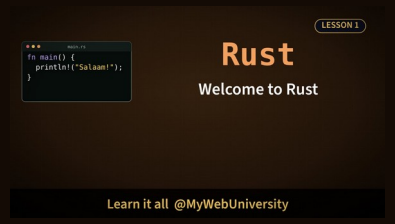
## FULL COURSE

12 Lessons · Beginner to Advanced

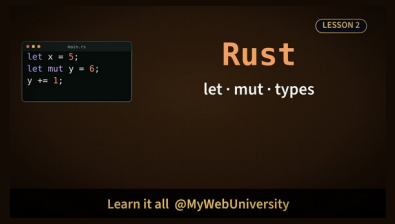
Free · English & Dari

Learn it all @MyWebUniversity

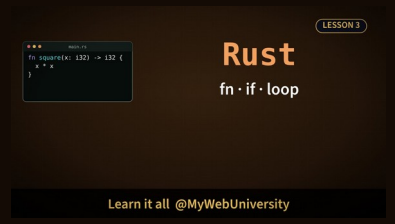
# The 12 Lessons




1. Welcome to Rust




2. Variables, Mutability and Types



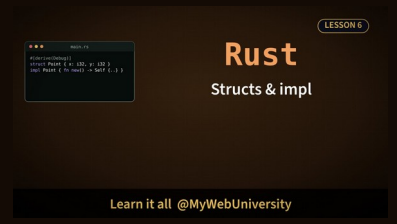

3. Functions and Control Flow



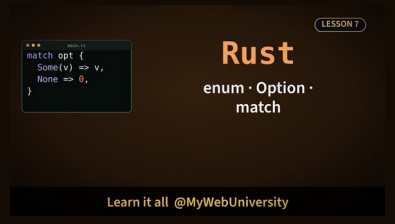

4. Ownership and Borrowing




5. Strings, Slices and Lifetimes




6. Structs and Methods



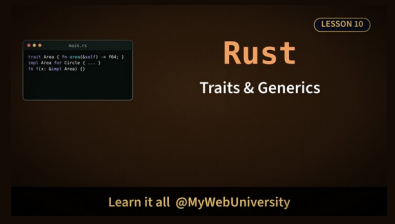

7. Enums, Option and match




8. Error Handling with Result




9. Collections, Closures and Iterators




10. Traits and Generics




11. Smart Pointers and Fearless Concurrency




12. Project: A Command-Line Tool




A FREE MYWEBUNIVERSITY COURSE

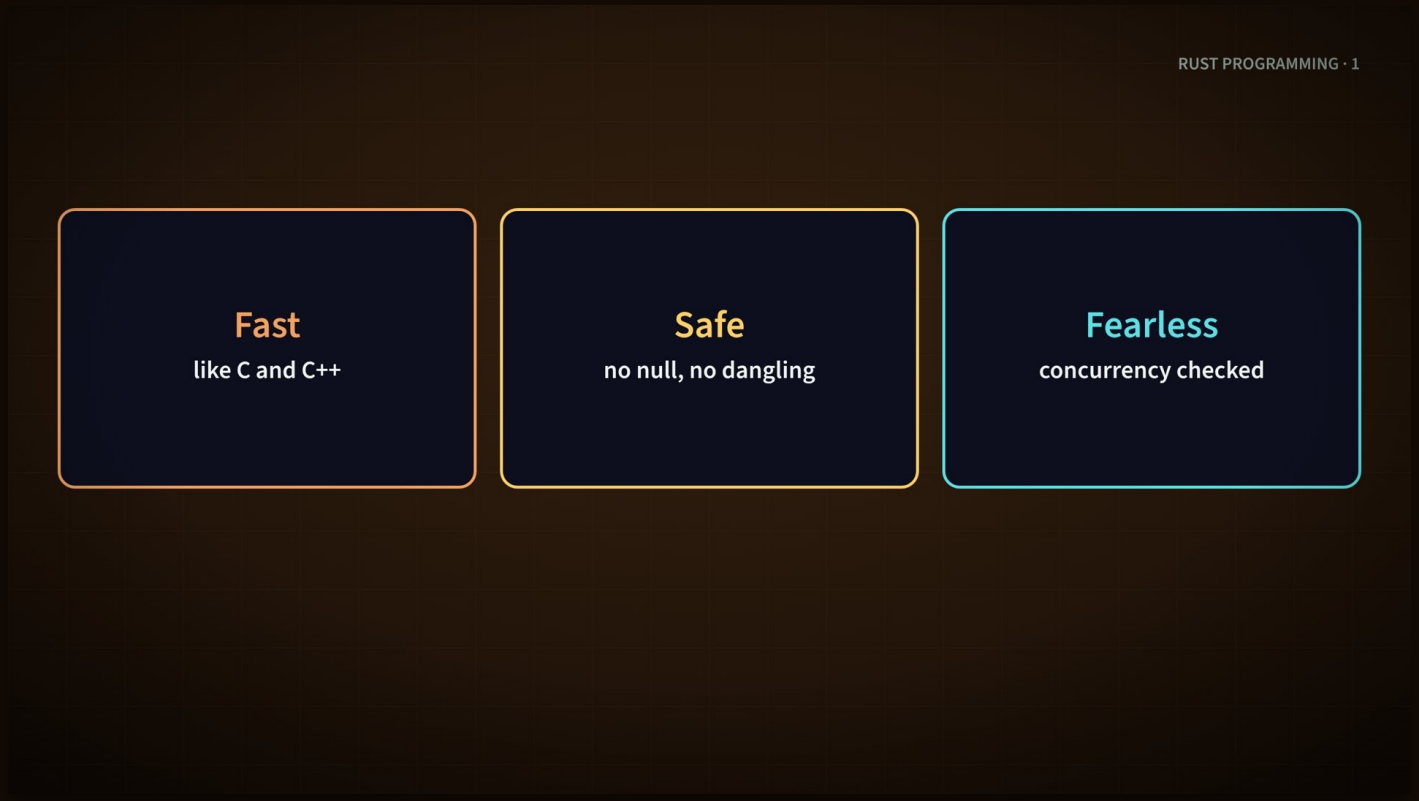
# RUST PROGRAMMING

LESSON 1

## Welcome to Rust

Why Rust, rustup, Cargo, your first program and the compiler as a teacher

# Why Rust?




- Fast — like C and C++
- Safe — no null, no dangling
- Fearless — concurrency checked

1.01

# rustup and Cargo



terminal

```
$ curl --proto '=https' -sSf https://sh.rustup.rs | sh
$ rustc --version
rustc 1.82.0 (f6e511eec 2024-10-15)
$ cargo --version
cargo 1.82.0
```

- Install Rust with rustup from rustup dot rs. It installs rustc, the compiler, and Cargo, …
- rustup update keeps everything current. A new stable version comes out every six weeks.

1.02

# Hello, World with Cargo



src/main.rs

```rust
fn main() {
    println!("Salaam, World!");
    let city = "Kabul";
    println!("Hello from {city}");
}

// $ cargo new hello && cd hello
// $ cargo run
```

- cargo new creates a project with a Cargo dot toml file and src slash main dot rs. cargo run …
- fn main is the entry point. println with an exclamation mark is a macro, which is why it …

1.03

# The Compiler Is Your Teacher



terminal

```
error[E0384]: cannot assign twice to immutable variable `x`
 --> src/main.rs:3:5
2 |     let x = 5;
  |         - help: consider making this binding mutable: `mut x`
3 |     x = 6;
  |     ^^^^^ cannot assign twice to immutable variable
```

- Rust's compiler is strict, but its error messages are famous for being helpful. They point …
- Read them carefully. Most beginners fight the compiler at first, then realize it is …

1.04

# Cargo Commands



| Command | Does |
| --- | --- |
| `cargo run` | build and run |
| `cargo check` | fast error check |
| `cargo build --release` | optimized binary |
| `cargo test` | run tests |
| `cargo clippy` | lint advice |

- cargo check finds errors quickly without building. cargo build dash dash release makes an …
- Install rust-analyzer in VS Code for inline types and errors as you type.

1.05




A FREE MYWEBUNIVERSITY COURSE

# RUST PROGRAMMING

LESSON 2

## Variables, Mutability and Types

let, mut, shadowing, const, scalar types, tuples and arrays

# let and mut



main.rs

```
let name = "Mariam";
let mut score = 90;
score += 5;                    // OK: score is mut
// name = "Zahra";            // error: immutable
let total: i32 = 100;          // explicit type
println!("{name}: {score}/{total}");
```

- let creates a variable, and it is immutable by default. Add mut when you need to change it.
- This default makes code easier to reason about: if you do not see mut, the value never ...

2.01

# Shadowing and const



main.rs

```
const MAX_STUDENTS: u32 = 30;

let input = "42";
let input: i32 = input.parse().unwrap();   // shadowed: now a number
let input = input * 2;                     // 84
```

- You can declare a new variable with the same name using let again. This is shadowing, and …
- const values must have a type, are fixed at compile time, and use UPPER underscore SNAKE case.

2.02

# Scalar Types



| Type | Example | Note |
| --- | --- | --- |
| `i32 · u8` | `-7 · 255` | signed / unsigned |
| `usize` | `3` | indexes, lengths |
| `f64` | `3.14` | default float |
| `bool` | `true` | |
| `char` | `'ک'` | 4-byte Unicode |

- Integers come in signed and unsigned sizes: i8 to i128, u8 to u128, plus isize and usize …
- f64 is the default float, bool is true or false, and char is a full Unicode character, four …

2.03

# No Silent Surprises



main.rs

```rust
let a: u8 = 250;
let b = a as u32 + 10;            // 260, explicit cast
a.checked_add(10);                // None (would overflow)
a.saturating_add(10);             // 255
let avg = 7 as f64 / 2.0;         // 3.5
```

- Rust never converts number types silently. Use as to cast, and methods like checked_add or …
- In debug builds, integer overflow stops the program with a clear message instead of giving …

2.04

# Tuples and Arrays



main.rs

```
let student = ("Ahmad", 21, 88.5);
let (name, age, score) = student;
println!("{}", student.0);          // Ahmad

let primes = [2, 3, 5, 7];
let zeros = [0; 5];                    // [0, 0, 0, 0, 0]
println!("{:?} {}", primes, primes.len());
```

- A tuple groups values of different types. Destructure it with let, or read fields with dot …
- An array has a fixed length and one type. Indexing past the end panics instead of reading …

2.05




A FREE MYWEBUNIVERSITY COURSE

# RUST PROGRAMMING

LESSON 3

## Functions and Control Flow

fn, expressions vs statements, if as an expression, loop, while, for and ranges

# Functions



main.rs

```rust
fn square(x: i32) -> i32 {
    x * x                // no semicolon: returned
}

fn average(scores: &[f64]) -> f64 {
    let sum: f64 = scores.iter().sum();
    sum / scores.len() as f64
}
```

- Define functions with fn. Parameter types are required, and the return type follows an arrow.
- The last expression, without a semicolon, is the return value. Adding a semicolon turns it …

3.01

# if Is an Expression



main.rs

```
let grade = if score >= 90 {
    'A'
} else if score >= 75 {
    'B'
} else {
    'C'
};
let label = if passed { "pass" } else { "fail" };
```

- if needs no parentheses, and its condition must be a real bool: Rust will not treat a …
- Because if is an expression, you can assign its result directly, like a ternary operator in …

3.02

# loop, while and break Values



main.rs

```rust
let mut n = 1;
let first_big = loop {
    n *= 2;
    if n > 100 { break n; }       // 128
};

'outer: for i in 0..5 {
    for j in 0..5 {
        if i * j == 6 { break 'outer; }
    }
}
```

- loop repeats forever until you break. break can carry a value out of the loop, so loop is …
- while loops while a condition holds. Labels with an apostrophe let you break out of an …

3.03

# for and Ranges



main.rs

```rust
for i in 1..=5 { print!("{i} "); }         // 1 2 3 4 5
for i in (0..10).step_by(3) { print!("{i} "); }   // 0 3 6 9
for i in (1..=3).rev() { print!("{i} "); }  // 3 2 1

let cities = ["Kabul", "Herat"];
for (i, c) in cities.iter().enumerate() {
    println!("{}. {c}", i + 1);
}
```

- for walks over anything iterable: ranges, arrays and collections. Two dots exclude the end, …
- rev reverses, step_by skips, and enumerate gives the index too.

3.04

# Blocks Are Expressions



main.rs

```
let area = {
    let w = 3.0;
    let h = 4.5;
    w * h                // value of the block
};
// w and h no longer exist here
println!("{area}");     // 13.5
```

- Any block in curly braces is an expression whose value is its last line. Use this to …
- Variables created inside the block disappear when it ends.

**3.05**




A FREE MYWEBUNIVERSITY COURSE

# RUST PROGRAMMING

LESSON 4

## Ownership and Borrowing

The three ownership rules, moves, clone, references and the borrow checker

# The Ownership Rules

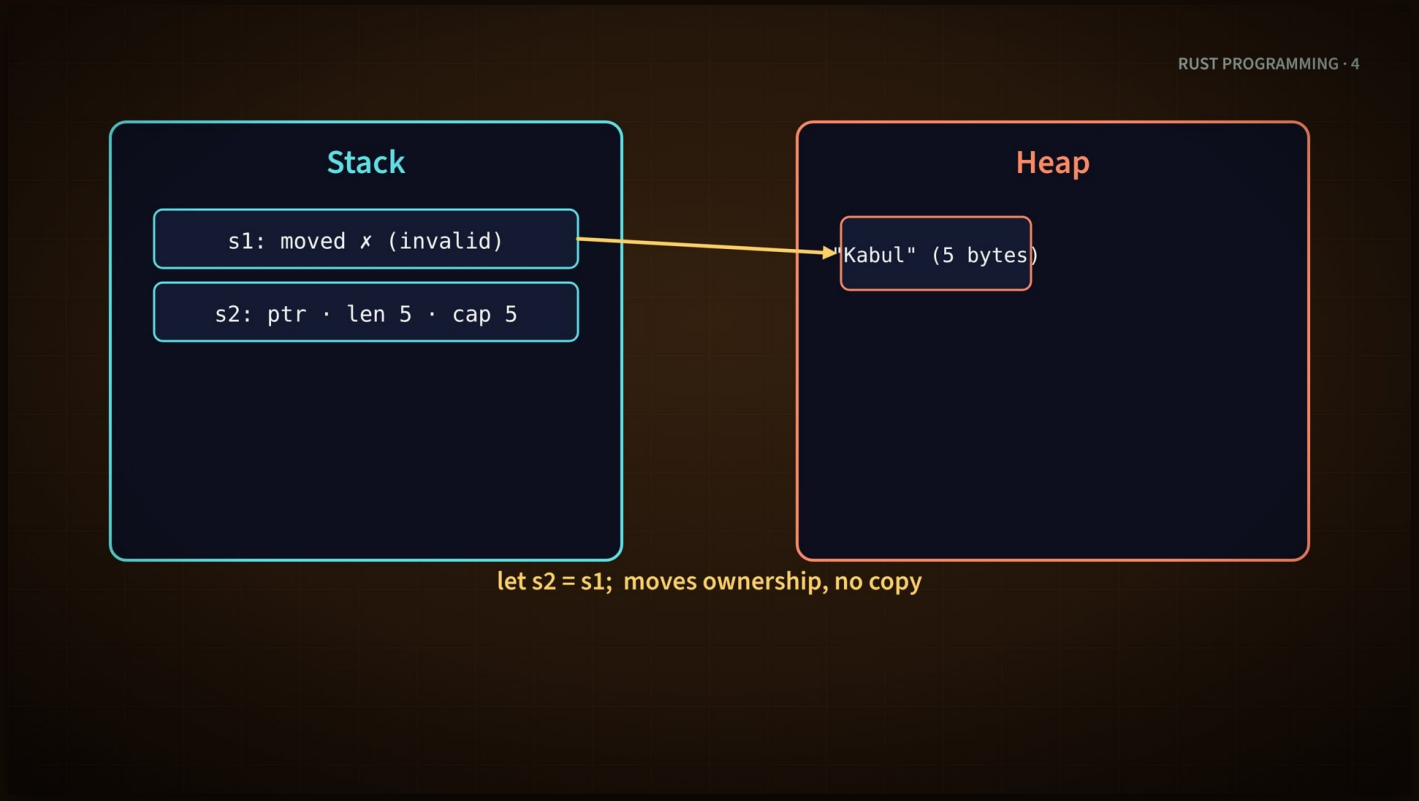


| # | Rule |
|---|---|
| 1 | each value has one owner |
| 2 | only one owner at a time |
| 3 | owner leaves scope → value dropped |

- Every value has exactly one owner. When the owner goes out of scope, the value is dropped …
- These three rules let Rust manage memory at compile time, with no garbage collector and no …

4.01

# Move




- Assigning a String to another variable moves it. The first variable is no longer valid, so …
- Simple types like integers and bools are Copy, so they are copied instead of moved.

4.02

# clone



main.rs

```rust
let s1 = String::from("Kabul");
let s2 = s1.clone();           // deep copy
println!("{s1} {s2}");        // both valid

let a = 5;
let b = a;                     // i32 is Copy
println!("{a} {b}");
```

- When you really need two independent copies, call clone. It copies the heap data too, so it …
- Seeing clone in code is a signal: a deep copy is happening here.

4.03

# Borrowing with References



main.rs

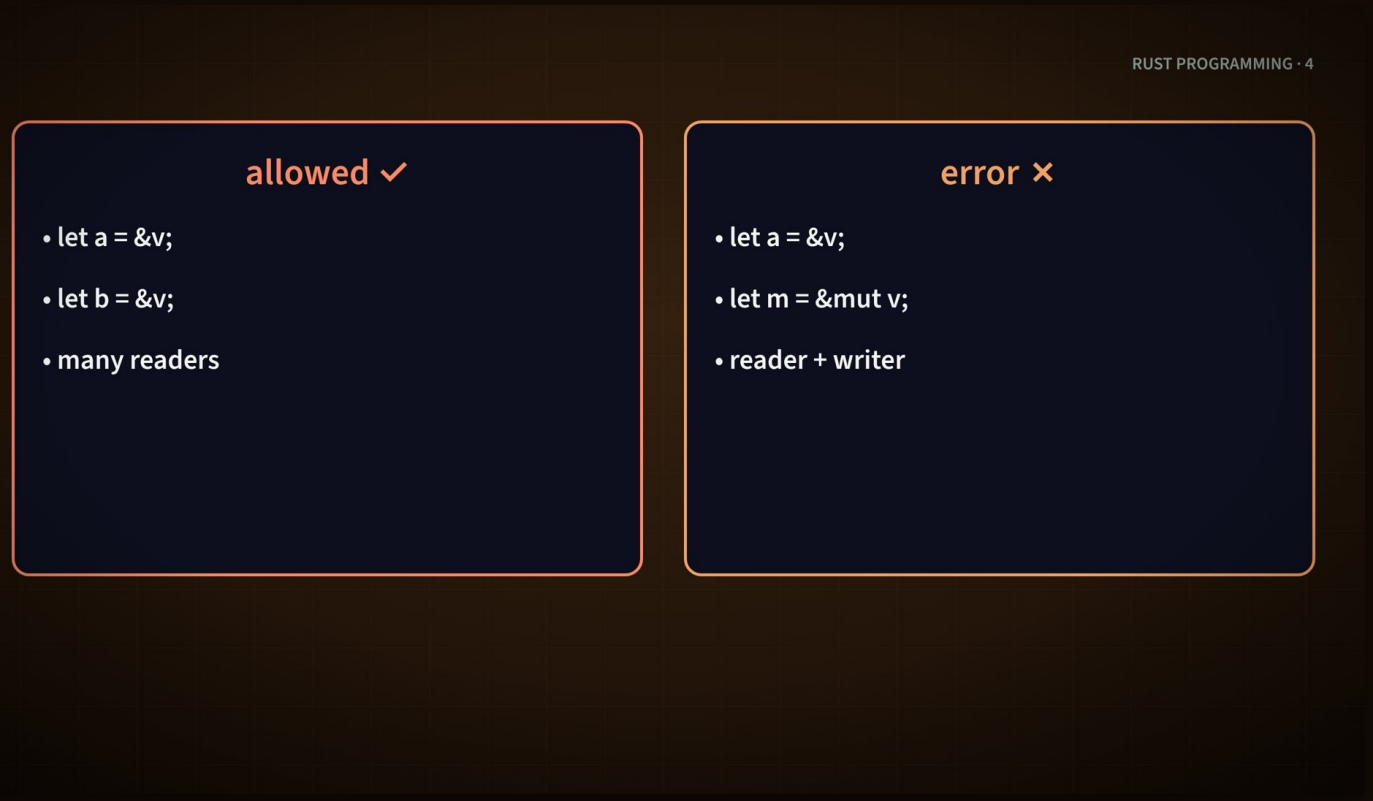
```rust
fn length(s: &String) -> usize {
    s.len()                      // borrows, does not own
}
fn shout(s: &mut String) {
    s.push_str("!");
}
let mut city = String::from("Herat");
let n = length(&city);
shout(&mut city);                // "Herat!"
```

- Instead of moving a value into a function, lend it with an ampersand. The function borrows …
- An ampersand mut reference lets the borrower change the value.

4.04

# The Borrow Checker






- The rule: at any moment you can have either many shared references, or exactly one mutable …
- This single rule, checked at compile time, prevents data races and use-after-free bugs …

4.05




A FREE MYWEBUNIVERSITY COURSE

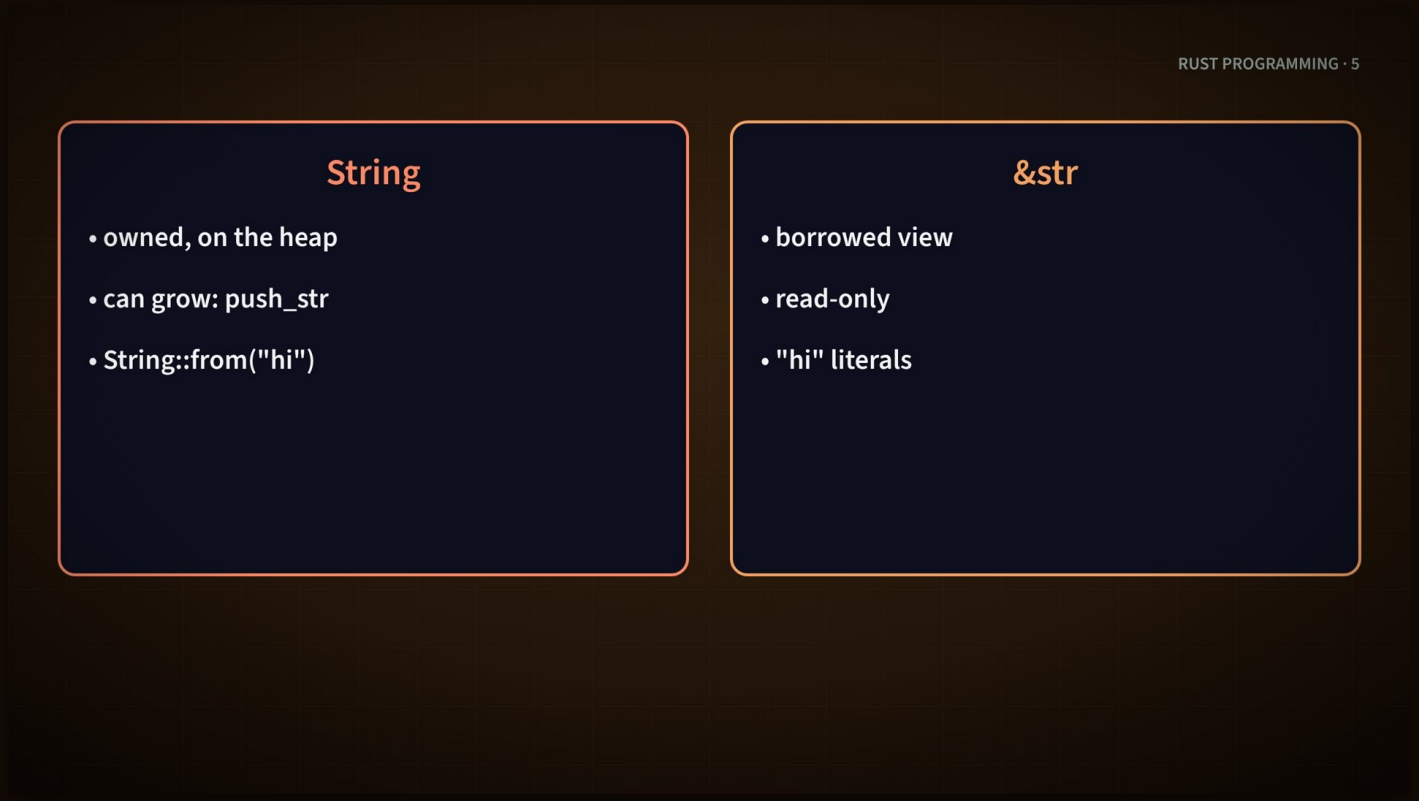
# RUST PROGRAMMING

LESSON 5

## Strings, Slices and Lifetimes

String vs &str, string slices, array slices, UTF-8 and lifetime annotations

# String vs &str




- String is an owned, growable string on the heap. ampersand str is a borrowed view into …
- Functions should usually take ampersand str: then they accept both literals and Strings.

5.01

# String Slices



main.rs

```rust
let s = String::from("Salaam World");
let hello = &s[0..6];          // "Salaam"
let world = &s[7..];           // "World"

fn first_word(s: &str) -> &str {
    s.split(' ').next().unwrap_or("")
}
```

- A slice borrows a range of a string with square brackets and two dots. It points into the …
- Indexes are byte positions, and Rust panics if you slice in the middle of a multi-byte …

5.02

# UTF-8 and chars



main.rs

```
let word = "کابل";
word.len();                    // 8 bytes
word.chars().count();          // 4 characters
for c in word.chars() { print!("[{c}]"); }
let upper = "kabul".to_uppercase();
let joined = format!("{}-{}", "Kabul", 2026);
```

- Rust strings are always valid UTF-8. You cannot index one character with s at zero, because …
- Use chars to walk characters, and len for the byte length. Dari text works correctly.

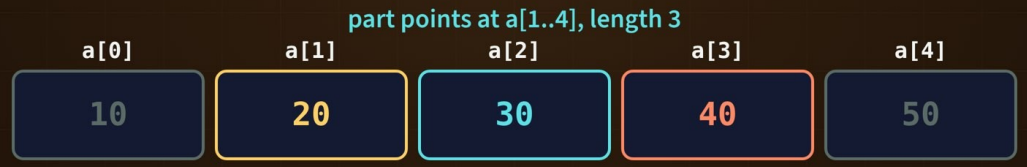
5.03

# Array Slices



main.rs

```
let a = [10, 20, 30, 40, 50];
let part: &[i32] = &a[1..4];
```




- Slices work for arrays and vectors too: ampersand square brackets of T. A function taking a …
- Slices know their own length, so there is no buffer overflow.

5.04

# Lifetimes



main.rs

```rust
fn longest<'a>(x: &'a str, y: &'a str) -> &'a str {
    if x.len() > y.len() { x } else { y }
}

let a = String::from("Mazar-i-Sharif");
let result;
{
    let b = String::from("Herat");
    result = longest(&a, &b);
}   // error if result is used here: b does not live long enough
```

- A reference must never outlive the data it points to. Usually the compiler figures this out …
- When a function returns a reference from one of several inputs, you write a lifetime like …

5.05




A FREE MYWEBUNIVERSITY COURSE

# RUST PROGRAMMING

LESSON 6

## Structs and Methods

Defining structs, impl blocks, &self, associated functions and derive

# Defining Structs



main.rs

```rust
struct Student {
    name: String,
    city: String,
    score: f64,
}
let name = String::from("Mariam");
let mut s = Student { name, city: "Kabul".into(), score: 95.0 };
s.score += 2.0;
```

- A struct groups named fields. Create an instance by giving every field a value. Field init …
- The whole instance must be mut to change any field.

6.01

# impl and Methods



main.rs

```
impl Student {
    fn grade(&self) -> char {
        if self.score >= 90.0 { 'A' } else { 'B' }
    }
    fn add_bonus(&mut self, points: f64) {
        self.score = (self.score + points).min(100.0);
    }
}
s.add_bonus(5.0);
println!("{}", s.grade());
```

- Methods live in an impl block. The first parameter, ampersand self, borrows the instance. …
- Call methods with a dot. Rust automatically adds the ampersand for you.

6.02

# Associated Functions



main.rs

```
impl Student {
    fn new(name: &str, city: &str) -> Self {
        Self { name: name.to_string(), city: city.to_string(), score: 0.0 }
    }
}
let s = Student::new("Ahmad", "Herat");
```

- Functions in impl without self are associated functions, called with two colons. new is the …
- Self with a capital S means the type itself.

6.03

# derive: Debug, Clone, PartialEq



main.rs

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

let p = Point { x: 3, y: 4 };
println!("{:?}", p);       // Point { x: 3, y: 4 }
println!("{:#?}", p);      // pretty, multi-line
let q = p.clone();
assert_eq!(p, q);
dbg!(&p);
```

- The derive attribute writes common code for you. Debug lets you print with colon question …
- The dbg macro prints a value with its file and line, great for quick debugging.

6.04

# Tuple Structs and the Newtype Pattern



main.rs

```rust
struct Meters(f64);
struct Seconds(f64);

fn speed(d: Meters, t: Seconds) -> f64 {
    d.0 / t.0
}
speed(Meters(100.0), Seconds(9.6));
// speed(Seconds(9.6), Meters(100.0));   // compile error ✓
```

- Tuple structs have fields without names. A one-field tuple struct, called a newtype, gives …
- Now the compiler stops you from mixing up meters and seconds.

6.05




A FREE MYWEBUNIVERSITY COURSE

# RUST PROGRAMMING

LESSON 7

## Enums, Option and match

Enums with data, Option instead of null, match, if let and let else

# Enums with Data



main.rs

```
enum Shape {
    Circle { r: f64 },
    Rect { w: f64, h: f64 },
    Triangle(f64, f64, f64),
}

enum Message {
    Quit,
    Move { x: i32, y: i32 },
    Write(String),
}
```

- A Rust enum lists possible variants, and each variant can carry its own data. One type can …
- This makes impossible states impossible to represent.

7.01

# Option: No More null



main.rs

```rust
fn find_student(id: u32) -> Option<String> {
    if id == 1 { Some("Mariam".to_string()) } else { None }
}

let name = find_student(7).unwrap_or("unknown".to_string());
let len = find_student(1).map(|n| n.len());   // Some(6)
```

- Rust has no null. A value that might be missing has the type Option: either Some with a …
- The compiler forces you to handle None, so null pointer crashes simply cannot happen.

7.02

# match



main.rs

```rust
fn area(s: &Shape) -> f64 {
    match s {
        Shape::Circle { r } => 3.14159 * r * r,
        Shape::Rect { w, h } => w * h,
        Shape::Triangle(a, b, c) => heron(*a, *b, *c),
    }
}
let label = match score {
    90..=100 => "A", 75..=89 => "B", _ => "C",
};
```

- match compares a value against patterns and runs the first arm that fits. It must be …
- Patterns can pull data out of enums, and the underscore matches anything else.

7.03

# if let and let else



main.rs

```rust
if let Some(name) = find_student(1) {
    println!("Found {name}");
}

fn parse_age(s: &str) -> u32 {
    let Ok(age) = s.parse::<u32>() else {
        return 0;
    };
    age
}
```

- When you care about only one pattern, if let is shorter than a full match.
- let else binds a value or exits early, which keeps the happy path unindented.

7.04

# Methods on Enums



main.rs

```rust
#[derive(Debug, Clone, Copy)]
enum Light { Red, Yellow, Green }

impl Light {
    fn next(self) -> Light {
        match self {
            Light::Red => Light::Green,
            Light::Green => Light::Yellow,
            Light::Yellow => Light::Red,
        }
    }
}
```

- Enums can have impl blocks with methods, just like structs. Match on self inside them.
- Option and Result themselves are just enums from the standard library, with many helpful …

7.05




A FREE MYWEBUNIVERSITY COURSE

# RUST PROGRAMMING

LESSON 8

## Error Handling with Result

panic!, Result, the ? operator, custom errors, thiserror and anyhow

# panic!: Unrecoverable Errors



terminal

```
$ cargo run
thread 'main' panicked at src/main.rs:4:37:
config.toml must exist: Os { code: 2, kind: NotFound }
note: run with `RUST_BACKTRACE=1` to display a backtrace
```

- panic stops the program with a message. It is for bugs, situations that should never happen.
- unwrap and expect panic on None or Err. They are fine in quick examples and tests, but …

8.01

# Result



main.rs

```rust
use std::fs;

match fs::read_to_string("scores.txt") {
    Ok(text) => println!("{} lines", text.lines().count()),
    Err(e) => eprintln!("Could not read file: {e}"),
}
let n: i32 = "42".parse().unwrap_or(0);
```

- Functions that can fail return Result: Ok with the value, or Err with an error. Like …
- match on the result, or use methods like unwrap_or_else and map_err.

8.02

# The ? Operator



main.rs

```rust
fn average_from_file(path: &str) -> Result<f64, Box<dyn std::error::Error>> {
    let text = std::fs::read_to_string(path)?;
    let nums: Vec<f64> = text.lines()
        .map(|l| l.trim().parse::<f64>())
        .collect::<Result<_, _>>()?;
    Ok(nums.iter().sum::<f64>() / nums.len() as f64)
}
```

- The question mark after a Result returns the error early if there is one, or gives you the …
- It turns long match chains into clean, short code. main can return Result too.

8.03

# Custom Error Types



error.rs

```rust
#[derive(Debug, thiserror::Error)]
enum GradeError {
    #[error("cannot read file: {0}")]
    Io(#[from] std::io::Error),
    #[error("bad score on line {line}")]
    BadScore { line: usize },
}

fn load(p: &str) -> Result<Vec<f64>, GradeError> { /* ... ? ... */ }
```

- For libraries, define an error enum with one variant per failure. The thiserror crate …
- From conversions let the question mark convert lower-level errors automatically.

8.04

# anyhow for Applications



main.rs

```rust
use anyhow::{Context, Result};

fn main() -> Result<()> {
    let cfg = std::fs::read_to_string("config.toml")
        .context("reading config.toml")?;
    let port: u16 = cfg.trim().parse().context("port must be a number")?;
    println!("port {port}");
    Ok(())
}
```

- In applications, the anyhow crate gives one easy error type. context adds a helpful message …
- Rule of thumb: thiserror for libraries, anyhow for apps.

8.05




A FREE MYWEBUNIVERSITY COURSE

# RUST PROGRAMMING

LESSON 9

## Collections, Closures and Iterators

Vec, HashMap, closures, iterator adapters and collect

# Vec



main.rs

```rust
let mut scores = vec![90, 85, 77];
scores.push(92);
scores.len();                    // 4
scores[0];                       // 90
scores.get(10);                  // None, no panic
scores.sort();
scores.retain(|&s| s >= 80);     // keep 85, 90, 92
```

- Vec is a growable array on the heap. The vec macro creates one with values. push adds to …
- Borrow a Vec as a slice when a function only needs to read it.

9.01

# HashMap



main.rs

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

let mut counts: HashMap<&str, u32> = HashMap::new();
for w in text.split_whitespace() {
    *counts.entry(w).or_insert(0) += 1;
}
if let Some(n) = counts.get("kabul") { println!("{n}"); }
```

- HashMap stores key-value pairs. The entry API makes counting and grouping one line: …
- Iteration order is not defined. Use BTreeMap when you need sorted keys.

9.02

# Closures



main.rs

```rust
let bonus = 5;
let add_bonus = |s: i32| s + bonus;       // captures bonus
add_bonus(90);                            // 95

let mut count = 0;
let mut inc = || count += 1;              // FnMut
inc(); inc();

let owned = move || println!("{bonus}");
```

- A closure is an anonymous function written with pipes around its parameters. It can capture …
- move forces the closure to take ownership of what it captures, which you need for threads.

9.03

# Iterator Adapters




- scores.iter() — iterator
- filter — |s| **s >= 80
- map — |s| s * 2

9.04

# Putting It Together



main.rs

```
let mut by_city: BTreeMap<&str, Vec<f64>> = BTreeMap::new();
for s in students.iter().filter(|s| s.score >= 50.0) {
    by_city.entry(&s.city).or_default().push(s.score);
}
for (city, v) in &by_city {
    let avg = v.iter().sum::<f64>() / v.len() as f64;
    println!("{city:<10} {avg:.1}");
}
```

- Here we read student records, keep those who passed, group average scores by city, and …
- into_iter consumes a collection, iter borrows it, and iter_mut borrows it mutably.

**9.05**




A FREE MYWEBUNIVERSITY COURSE

# RUST PROGRAMMING

LESSON 10

## Traits and Generics

Defining traits, default methods, generics, trait bounds, impl Trait and dyn Trait

# Traits



main.rs

```rust
trait Describe {
    fn name(&self) -> String;
    fn describe(&self) -> String {          // default method
        format!("This is {}", self.name())
    }
}
impl Describe for Student {
    fn name(&self) -> String { self.name.clone() }
}
```

- A trait defines behavior that types can share, like an interface. Implement it for a type …
- Traits can provide default method bodies that implementers may override.

10.01

# Standard Traits: Display and From



main.rs

```rust
use std::fmt;

impl fmt::Display for Student {
    fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
        write!(f, "{} ({:.1})", self.name, self.score)
    }
}
println!("{s}");           // Mariam (95.0)
let c: Celsius = Fahrenheit(98.6).into();
```

- Implement Display to control how your type prints with curly braces. Implement From to …
- Iterator, Default, PartialOrd and Hash are other traits you will meet often.

10.02

# Generics and Trait Bounds



main.rs

```rust
fn largest<T: PartialOrd + Copy>(items: &[T]) -> T {
    let mut max = items[0];
    for &x in items {
        if x > max { max = x; }
    }
    max
}
largest(&[3, 9, 2]);          // 9
largest(&[1.5, 0.2]);         // 1.5
```

- Generic functions work for many types. A trait bound, T colon PartialOrd, says what the …
- Rust generates specialized code for each type you use, so generics cost nothing at run time.

10.03

# impl Trait and where Clauses



main.rs

```rust
fn print_all(items: &[impl Describe]) {
    for i in items { println!("{}", i.describe()); }
}

fn evens(limit: u32) -> impl Iterator<Item = u32> {
    (0..limit).filter(|n| n % 2 == 0)
}

fn report<T>(x: T) where T: Describe + fmt::Display { /* ... */ }
```

- impl Trait in a parameter accepts any type with that trait. In a return position, it hides …
- where clauses move long bounds after the signature to keep it readable.

10.04

# Trait Objects: dyn Trait




- Generics pick the type at compile time. When you need a list of different types together, …
- Method calls go through a table at run time. It is slightly slower but very flexible.

10.05




A FREE MYWEBUNIVERSITY COURSE

# RUST PROGRAMMING

LESSON 11

## Smart Pointers and Fearless Concurrency

Box, Rc, RefCell, threads, move closures, channels, Arc and Mutex

# Box, Rc and RefCell



| Type | Gives you | Threads? |
|---|---|---|
| `Box<T>` | heap, one owner | ✓ |
| `Rc<T>` | shared owners | ✗ |
| `RefCell<T>` | mutate via & | ✗ |
| `Arc<T>` | shared, atomic | ✓ |
| `Mutex<T>` | locked mutation | ✓ |

- Box puts a value on the heap with one owner, needed for recursive types like trees. Rc lets …
- RefCell allows changing data behind a shared reference, with the borrow rules checked at …

11.01

# Spawning Threads



main.rs

```rust
use std::thread;

let files = vec!["a.log", "b.log", "c.log"];
let handles: Vec<_> = files.into_iter().map(|f| {
    thread::spawn(move || count_lines(f))
}).collect();

let total: usize = handles.into_iter().map(|h| h.join().unwrap()).sum();
```

- thread spawn runs a closure on a new operating system thread. The handle's join waits for …
- The closure needs move to own the data it uses, because the thread may outlive the current …

11.02

# Channels



main.rs

```rust
use std::sync::mpsc;

let (tx, rx) = mpsc::channel();
for id in 0..3 {
    let tx = tx.clone();
    thread::spawn(move || tx.send(format!("worker {id} done")).unwrap());
}
drop(tx);
for msg in rx { println!("{msg}"); }
```

- mpsc channels send values between threads: many producers, single consumer. The sender …
- Sending a value moves it, so two threads can never use it at the same time.

11.03

# Arc and Mutex



main.rs

```rust
use std::sync::{Arc, Mutex};

let counter = Arc::new(Mutex::new(0));
let mut hs = vec![];
for _ in 0..8 {
    let c = Arc::clone(&counter);
    hs.push(thread::spawn(move || { *c.lock().unwrap() += 1; }));
}
for h in hs { h.join().unwrap(); }
println!("{}", *counter.lock().unwrap());    // 8
```

- To share changeable state between threads, wrap it in Arc of Mutex. Arc shares ownership …
- lock returns a guard, and the lock is released automatically when the guard goes out of scope.

**11.04**

# Why It Is Fearless



terminal

```
error[E0277]: `Rc<i32>` cannot be sent between threads safely
   |
8  |     thread::spawn(move || println!("{}", rc));
   |     ^^^^^^^^^^^^^ `Rc<i32>` cannot be sent between threads safely
   = help: use `Arc<i32>` instead
```

- The Send and Sync traits mark which types are safe to move or share between threads. Rc is …
- Data races become compile errors, not bugs you hunt for at three in the morning. For async …

11.05




A FREE MYWEBUNIVERSITY COURSE

# RUST PROGRAMMING

LESSON 12

## Project: A Command-Line Tool

Crates, clap, serde, modules, tests, release builds and next steps

# Crates and Cargo.toml



Cargo.toml

```toml
[package]
name = "grades"
version = "0.1.0"
edition = "2021"

[dependencies]
anyhow = "1"
clap = { version = "4", features = ["derive"] }
serde = { version = "1", features = ["derive"] }
serde_json = "1"
```

- Libraries in Rust are called crates, published on crates dot io. cargo add puts one in …
- We will use clap for command-line arguments, serde and serde_json for JSON, and anyhow for …

12.01

# Arguments with clap



src/main.rs

```rust
use clap::Parser;

/// Summarize student grades from a JSON file
#[derive(Parser)]
struct Args {
    /// Path to the JSON file
    file: std::path::PathBuf,
    /// Minimum passing score
    #[arg(long, default_value_t = 50.0)]
    pass: f64,
}
```

- With clap's derive, a struct describes your command-line options. clap parses them, …
- Doc comments become the help descriptions.

12.02

# JSON with serde



src/main.rs

```rust
#[derive(serde::Deserialize)]
struct Student { name: String, city: String, score: f64 }

fn main() -> anyhow::Result<()> {
    let args = Args::parse();
    let text = std::fs::read_to_string(&args.file)?;
    let students: Vec<Student> = serde_json::from_str(&text)?;
    let passed = students.iter().filter(|s| s.score >= args.pass).count();
    println!("{passed}/{} passed", students.len());
    Ok(())
}
```

- serde turns Rust structs into JSON and back. Add derive Deserialize, and serde_json …
- The main function reads the file, parses it, and prints a report, with errors handled by …

12.03

# Tests and Release Builds



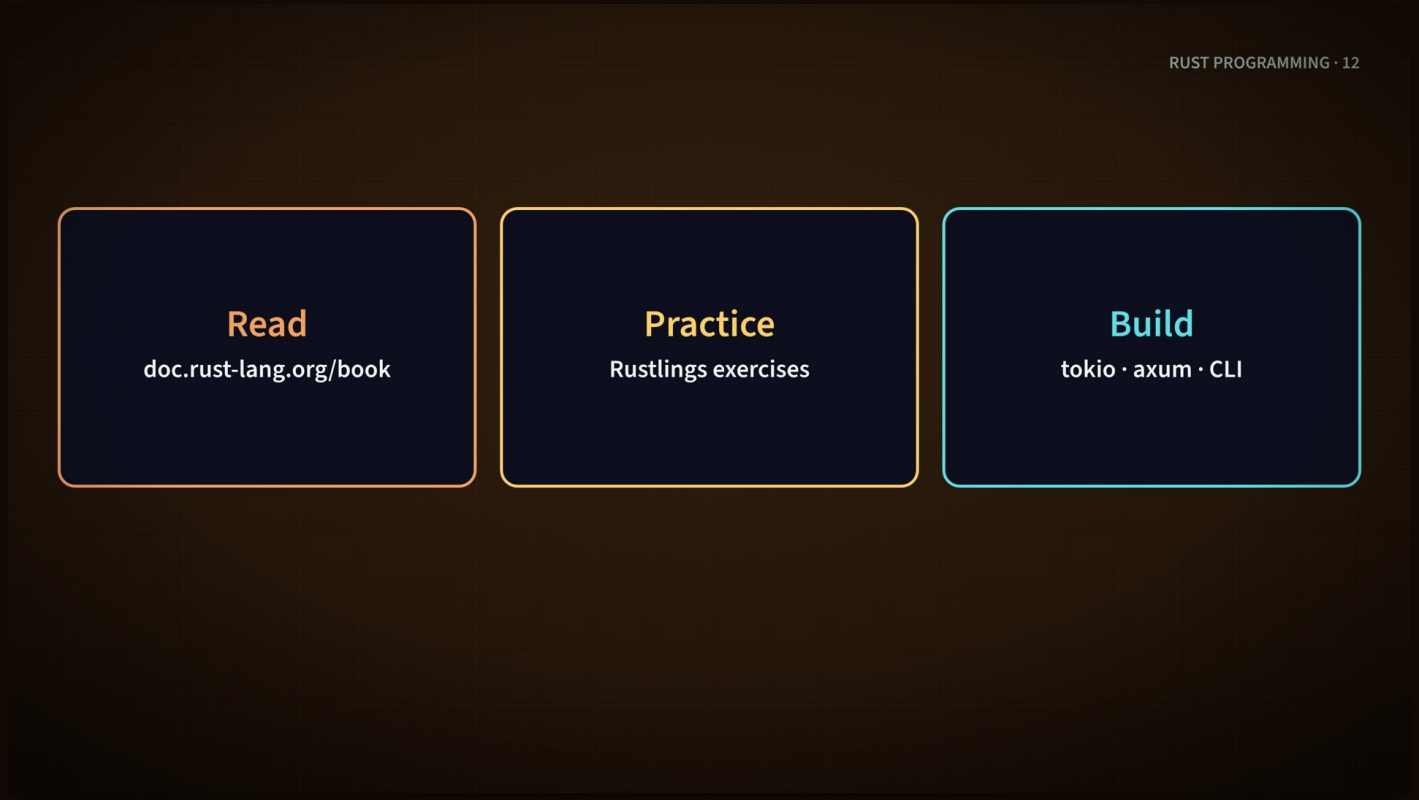
terminal

```
$ cargo test
running 3 tests
test tests::average_works ... ok
test result: ok. 3 passed; 0 failed
$ cargo build --release
$ ./target/release/grades students.json --pass 60
7/9 passed
```

- Write tests in a module marked cfg test, and functions marked with the test attribute. …
- cargo build dash dash release creates a fast, optimized binary in target slash release.

12.04

# Where to Go Next




- Read — doc.rust-lang.org/book
- Practice — Rustlings exercises
- Build — tokio · axum · CLI

12.05

Thank you for learning with us

# MyWebUniversity.com

Learn it all  @MyWebUniversity

Free technical education since 2000 — English and Dari