Rust

کورس کامل

۱۲ درس · مبتدی تا پیشرفته

انگلیسی و دری · رایگان

- let · mut · نوع‌ها ✓
- مالکیت و قرض ✓
- String و &str ✓
- struct و impl ✓
- enum · Option · match ✓
- Result و ? ✓
- trait و generics ✓
- همزمانی بی‌ترس ✓

همه را یاد بگیرید @MyWebUniversity-Dari

۱۲ درس این کورس



۱. به **Rust** خوش آمدید



۲. متغیرها، تغییرپذیری و نوع‌ها



۳. تابع‌ها و کنترول جریان

۴. مالکیت و قرض گرفتن

۵. رشته‌ها، **slice** ها و **lifetime** ها

۶. **struct** ها و **method** ها



۷. **enum** ها، **Option** و **match**



۸. مدیریت خطا با **Result**



۹. **collection** ها، **closure** ها و **iterator** ها

۱۰. **trait** ها و **generics**

۱۱. **smart pointer** ها و همزمانی بی‌ترس

۱۲. پروژه: یک ابزار خط فرمان

یک کورس رایگان از MyWebUniversity

برنامه‌نویسی Rust

درس ۱

به Rust خوش آمدید



چرا Rust، rustup، Cargo، اولین پروگرام شما و کمپایلر به عنوان معلم

چرا Rust؟

سریع — مثل C و C++

امن — بدون null، بدون dangling

بی‌ترس — همزمانی بررسی‌شده

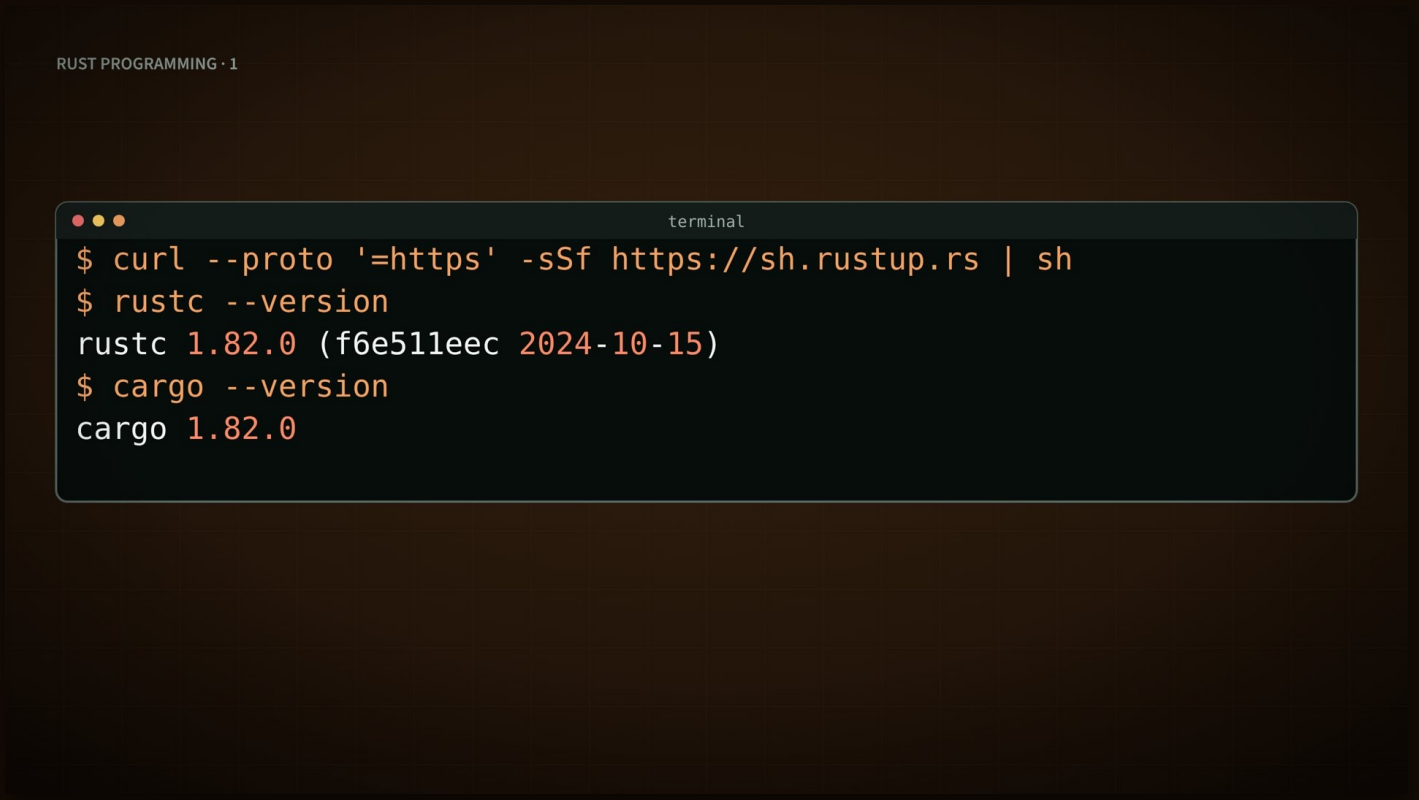


۱.۰۱

rustup و Cargo

Rust را با rustup از rustup.rs نصب کنید.
rustc، یعنی کمپایلر، و Cargo، یعنی ابزار ساخت و ...

rustup update همه چیز را به‌روز نگه می‌دارد.
هر شش هفته یک نسخهٔ stable نو منتشر می‌شود.



۱.۰۲

سلام دنیا با Cargo

RUST PROGRAMMING · 1

cargo new یک پروژه با یک فایل Cargo.toml و src/main.rs می‌سازد. cargo run آن را کمپایل و ...

fn main نقطهٔ شروع است. println! با علامت تعجب یک macro است، به همین دلیل علامت تعجب دارد.

src/main.rs

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

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

۱.۰۳

کمپایلر معلم شما است

کمپایلر Rust سخت‌گیر است، اما پیام‌های خطای آن به مفید بودن مشهور هستند. دقیقاً به مشکل ...

آن‌ها را با دقت بخوانید. بیشتر تازه‌کارها اول با کمپایلر می‌جنگند، بعد می‌فهمند که باگ‌های ...

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
```

۱۰۰۴

فرمان‌های Cargo

RUST PROGRAMMING · 1

فرمان	کار
`cargo run`	ساختن و اجرا
`cargo check`	بررسی سریع خطا
`cargo build --release`	فایل اجرایی بهینه
`cargo test`	اجرای آزمایش‌ها
`cargo clippy`	مشورهٔ lint

`cargo check` خطاها را بدون ساختن سریع پیدا می‌کند. `cargo build --release` یک فایل اجرایی …

`rust-analyzer` را در VS Code نصب کنید تا نوع‌ها و خطاها را هنگام تایپ ببینید.

۱.۰۵

یک کورس رایگان از MyWebUniversity

برنامه‌نویسی Rust

درس ۲

متغیرها، تغییرپذیری و نوع‌ها

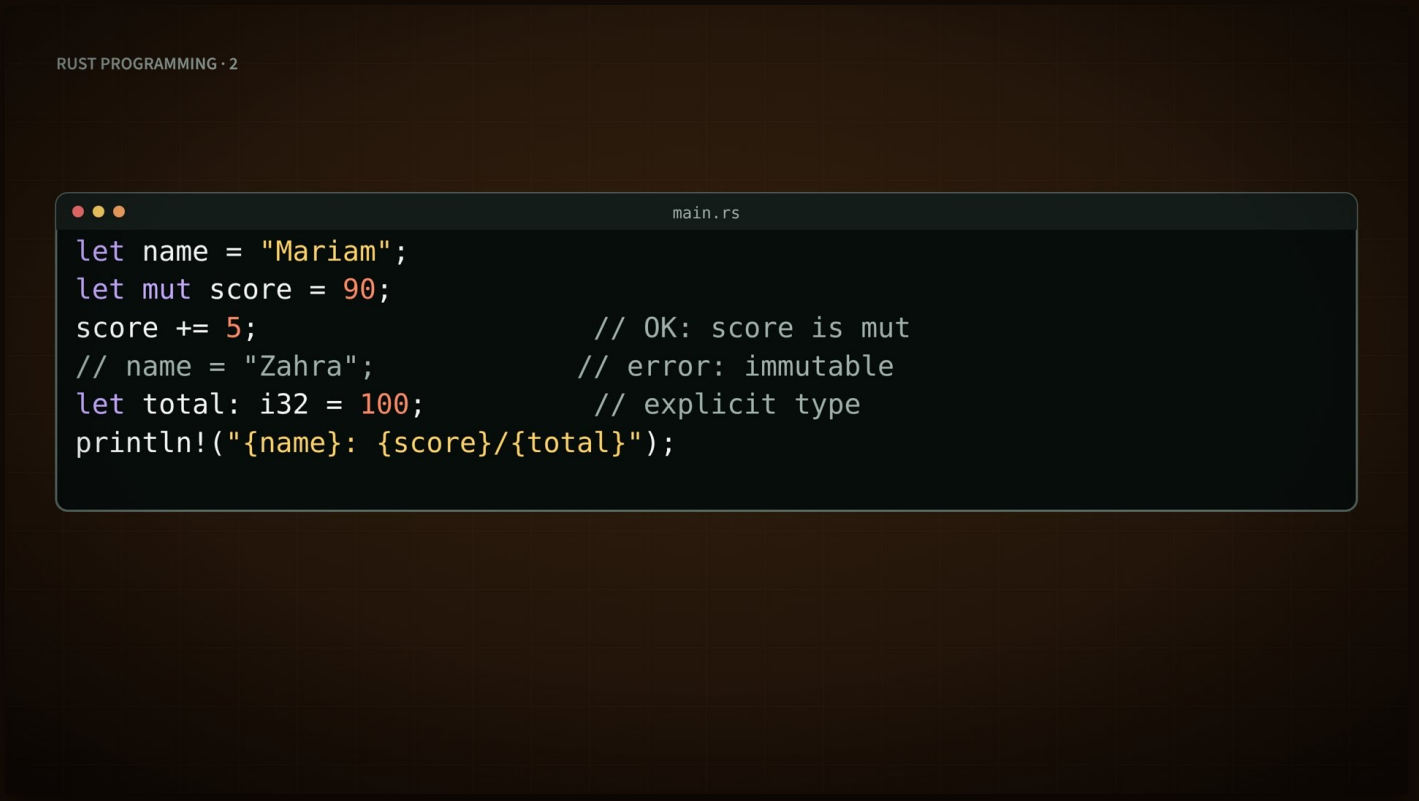
let، mut، shadowing، const، نوع‌های scalar، tuple ها و آرایه‌ها



let و mut

let یک متغیر می‌سازد، و به طور پیش‌فرض تغییرناپذیر است. وقتی لازم دارید آن را تغییر دهید ...

این پیش‌فرض فهمیدن کود را آسان‌تر می‌سازد: اگر mut را نمی‌بینید، مقدار هرگز تغییر نمی‌کند.



۲.۰۱

const و shadowing

می‌توانید با استفادهٔ دوباره از let یک متغیر نو با همان نام اعلان کنید. این shadowing است، ...

مقدارهای const باید نوع داشته باشند، هنگام کمپایل ثابت هستند، و از UPPER_SNAKE_CASE استفاده ...

main.rs

```rust
const MAX_STUDENTS: u32 = 30;

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

۲.۰۲

نوع‌های scalar

integer ها در اندازه‌های علامت‌دار و بی‌علامت می‌آیند: i8 تا u8، i128 تا u128، به علاوهٔ ...

f64 float پیش‌فرض است، bool یعنی true یا false، و char یک حرف کامل Unicode، چهار بایت، است، ...

RUST PROGRAMMING · 2

نوع	مثال	یادداشت
i32 · u8	255 · -7	علامت‌دار / بی‌علامت
usize	3	اندکس، طول
f64	3.14	float پیش‌فرض
bool	true	
char	'ک'	Unicode چهاربایتی

۲.۰۳

بدون غافل‌گیری بی‌صدا

Rust هرگز نوع‌های عدد را بی‌صدا تبدیل نمی‌کند. برای cast از as استفاده کنید، و از method ...

در ساخت‌های debug، overflow عدد صحیح پروگرام را با یک پیام روشن متوقف می‌کند به جای اینکه ...

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
```

۲.۰۴

tuple ها و آرایه‌ها

یک tuple مقدارهای نوع‌های مختلف را گروه‌بندی می‌کند. آن را با let باز کنید، یا فیلدها را با ...

یک آرایه طول ثابت و یک نوع دارد. اندکس بیرون از آخر به جای خواندن حافظهٔ تصادفی panic می‌کند.

main.rs

```rust
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());
```

۲.۰۵

یک کورس رایگان از MyWebUniversity

برنامه‌نویسی Rust

درس ۳

تابع‌ها و کنترول جریان

fn، expression در برابر statement، if به عنوان expression، loop، while، for و range ها



تابع‌ها

تابع‌ها را با fn تعریف کنید. نوع پارامترها لازم است، و نوع برگشتی بعد از یک تیر می‌آید.

آخرین expression، بدون سیمی‌کالن، مقدار برگشتی است. اضافه کردن یک سیمی‌کالن آن را به یک ...

RUST PROGRAMMING · 3

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
}
```

۳.۰۱

if یک expression است

if به پرانتز ضرورت ندارد، و شرط آن باید یک bool واقعی باشد: Rust یک عدد را true حساب نمی‌کند.

چون if یک expression است، می‌توانید نتیجهٔ آن را مستقیم به متغیر بدهید، مثل عملگر ternary ...

RUST PROGRAMMING · 3

main.rs

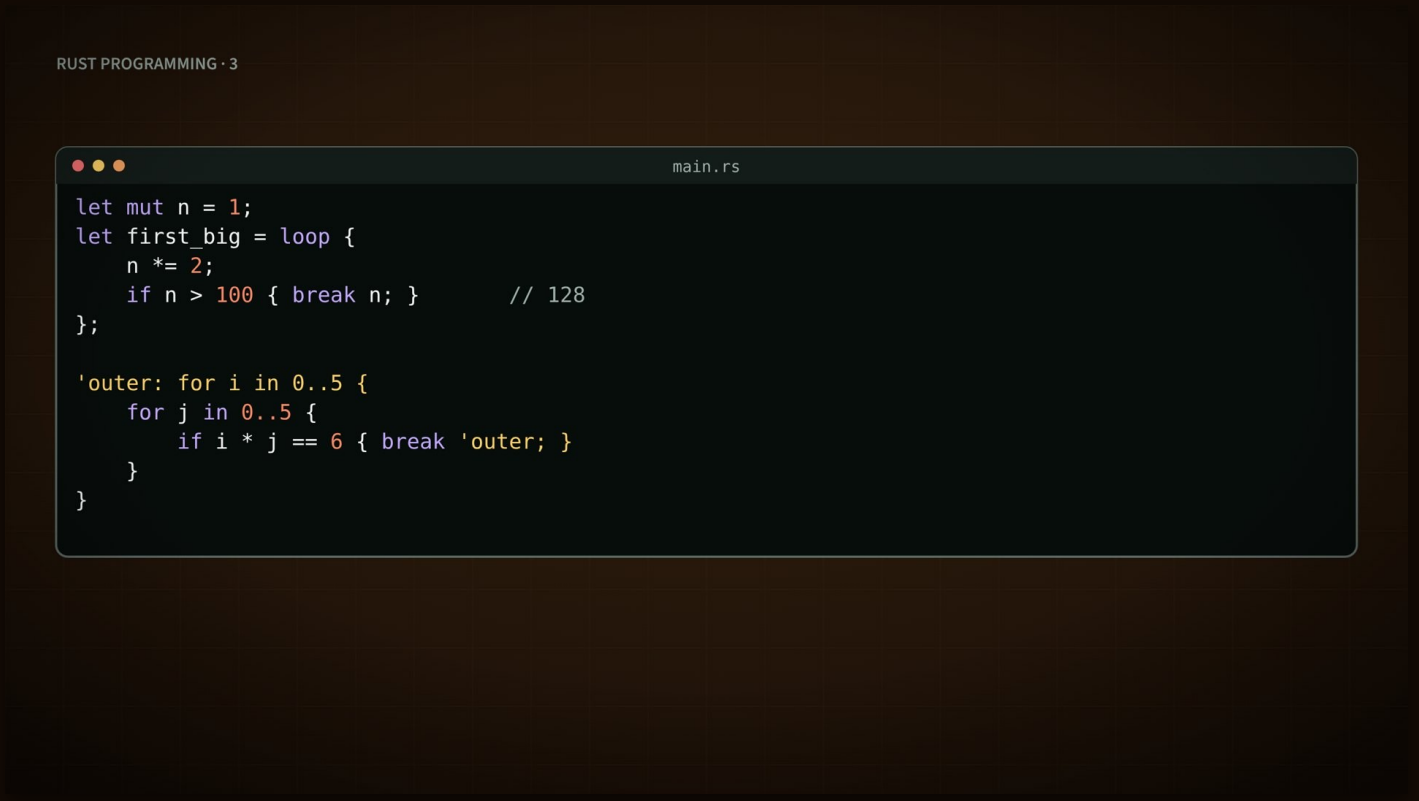
```
let grade = if score >= 90 {
    'A'
} else if score >= 75 {
    'B'
} else {
    'C'
};
let label = if passed { "pass" } else { "fail" };
```

۳.۰۲

loop، while و مقدار break

loop برای همیشه تکرار می‌کند تا break کنید.
break می‌تواند یک مقدار را از حلقه بیرون ببرد، ...

while تا وقتی یک شرط برقرار است حلقه می‌زند.
label ها با یک apostrophe اجازه می‌دهند از یک ...



۳.۰۳

for و range ها

for روی هر چیز قابل پیمایش راه می‌رود: range ها، آرایه‌ها و collection ها. دو نقطه آخر را ...

rev برعکس می‌کند، step_by جا می‌اندازد، و enumerate اندکس را هم می‌دهد.

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);
}
```

۳.۰۴

بلاک‌ها expression هستند

هر بلاک داخل قوس کج یک expression است که مقدار آن آخرین سطر آن است. از این برای حساب کردن ...

متغیرهایی که داخل بلاک ساخته شده‌اند وقتی بلاک تمام می‌شود ناپدید می‌شوند.

main.rs

```rust
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
```

۳.۰۵

یک کورس رایگان از MyWebUniversity

برنامه‌نویسی Rust

درس ۴

مالکیت و قرض گرفتن

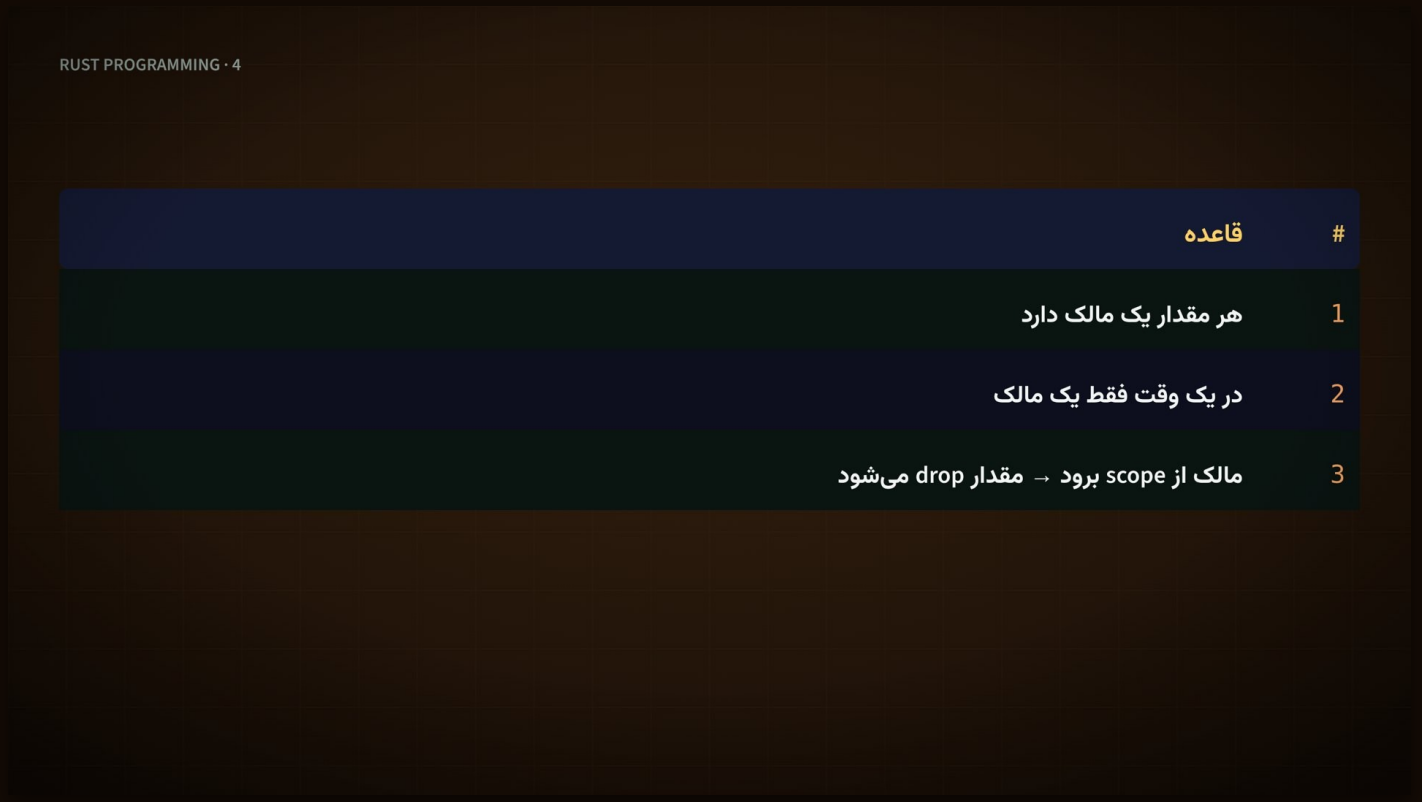


سه قاعدهٔ مالکیت، move، clone، reference ها و borrow checker

قاعده‌های مالکیت

هر مقدار دقیقاً یک مالک دارد. وقتی مالک از scope بیرون می‌رود، مقدار drop می‌شود و حافظهٔ ...

این سه قاعده اجازه می‌دهند Rust حافظه را هنگام کمپایل مدیریت کند، بدون garbage collector و ...

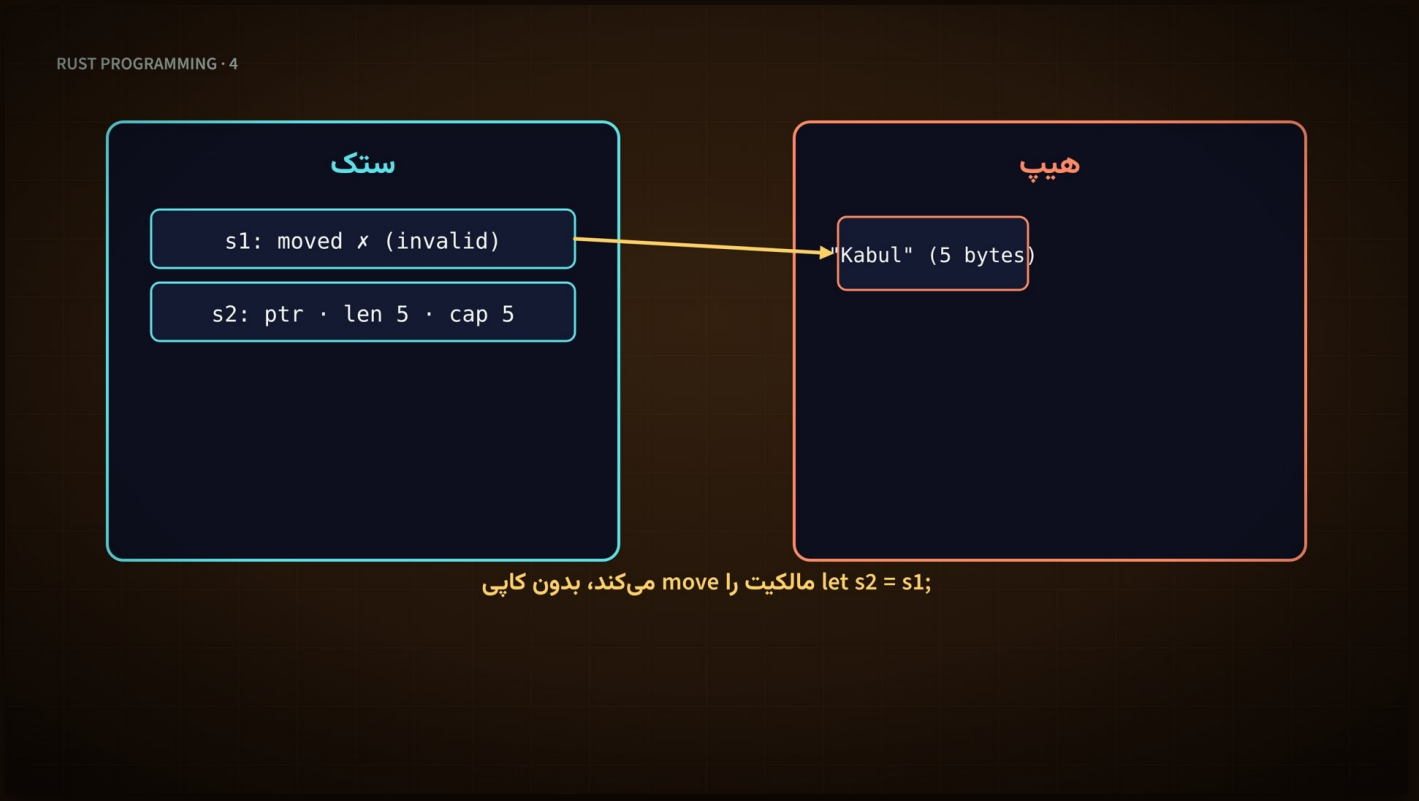


۴.۰۱

move

دادن یک String به متغیر دیگر آن را move می‌کند. متغیر اول دیگر معتبر نیست، پس حافظه هرگز دو ...

نوع‌های ساده مثل integer ها و bool ها Copy هستند، پس به جای move کاپی می‌شوند.



۴.۰۲

clone

وقتی واقعاً دو کاپی مستقل لازم دارید، clone را صدا کنید. دادهٔ heap را هم کاپی می‌کند، پس ...

دیدن clone در کود یک نشانه است: اینجا یک کاپی عمیق انجام می‌شود.

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}");
```

۴.۰۳

قرض گرفتن با reference ها

به جای move کردن یک مقدار به یک تابع، آن را با & قرض بدهید. تابع آن را قرض می‌گیرد، و ...

یک reference از نوع &mut اجازه می‌دهد قرض‌گیرنده مقدار را تغییر دهد.

main.rs

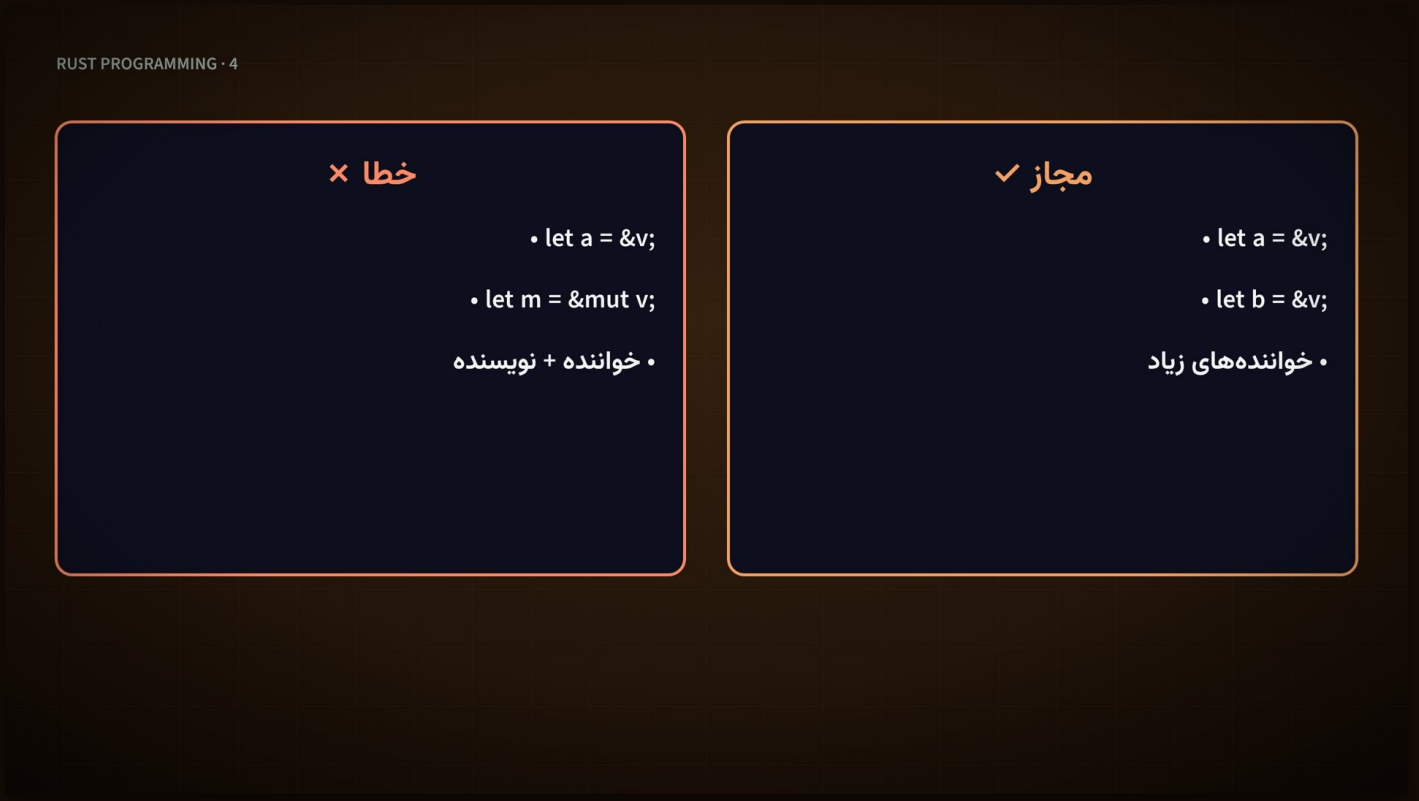
```rust
fn length(s: &String) -> usize {
    s.len()                      // borrows, does not own
}
fn shout(s: &mut String) {
    s.push_str("!");
}
let mut city = String::from("Herat");
let n = length(&city);
shout(&mut city);                // "Herat!"
```

۴۰۴

borrow checker

قاعده: در هر لحظه می‌توانید یا reference های شریک زیاد، یا دقیقاً یک reference قابل تغییر ...

همین یک قاعده، که هنگام کمپایل بررسی می‌شود، باگ‌های data race و use-after-free را کاملاً ...



۴.۰۵

یک کورس رایگان از MyWebUniversity

برنامه‌نویسی Rust

درس ۵

رشته‌ها، slice ها و lifetime ها



String در برابر &str، slice های رشته، slice های آرایه، UTF-8 و annotation های lifetime

String در برابر &str

String یک رشتهٔ مالکیت‌دار و قابل رشد روی heap است. &str یک نمای قرض‌گرفته به داخل دادهٔ ...

تابع‌ها معمولاً باید &str بگیرند: آن وقت هم literal ها و هم String ها را می‌پذیرند.



۵.۰۱

slice های رشته

یک slice یک محدوده از یک رشته را با قوس مربع و دو نقطه قرض می‌گیرد. به داخل دادهٔ اصلی ...

اندکس‌ها موقعیت بایت هستند، و Rust اگر در وسط یک حرف چندبایتی UTF-8 بُرش کنید panic می‌کند.

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("")
}
```

۵.۰۲

UTF-8 و char ها

رشته‌های Rust همیشه UTF-8 معتبر هستند. نمی‌توانید با s[0] یک حرف را اندکس کنید، چون حروف ...

از chars برای پیمایش حروف، و از len برای طول بایت استفاده کنید. متن دری درست کار می‌کند.

RUST PROGRAMMING · 5

main.rs

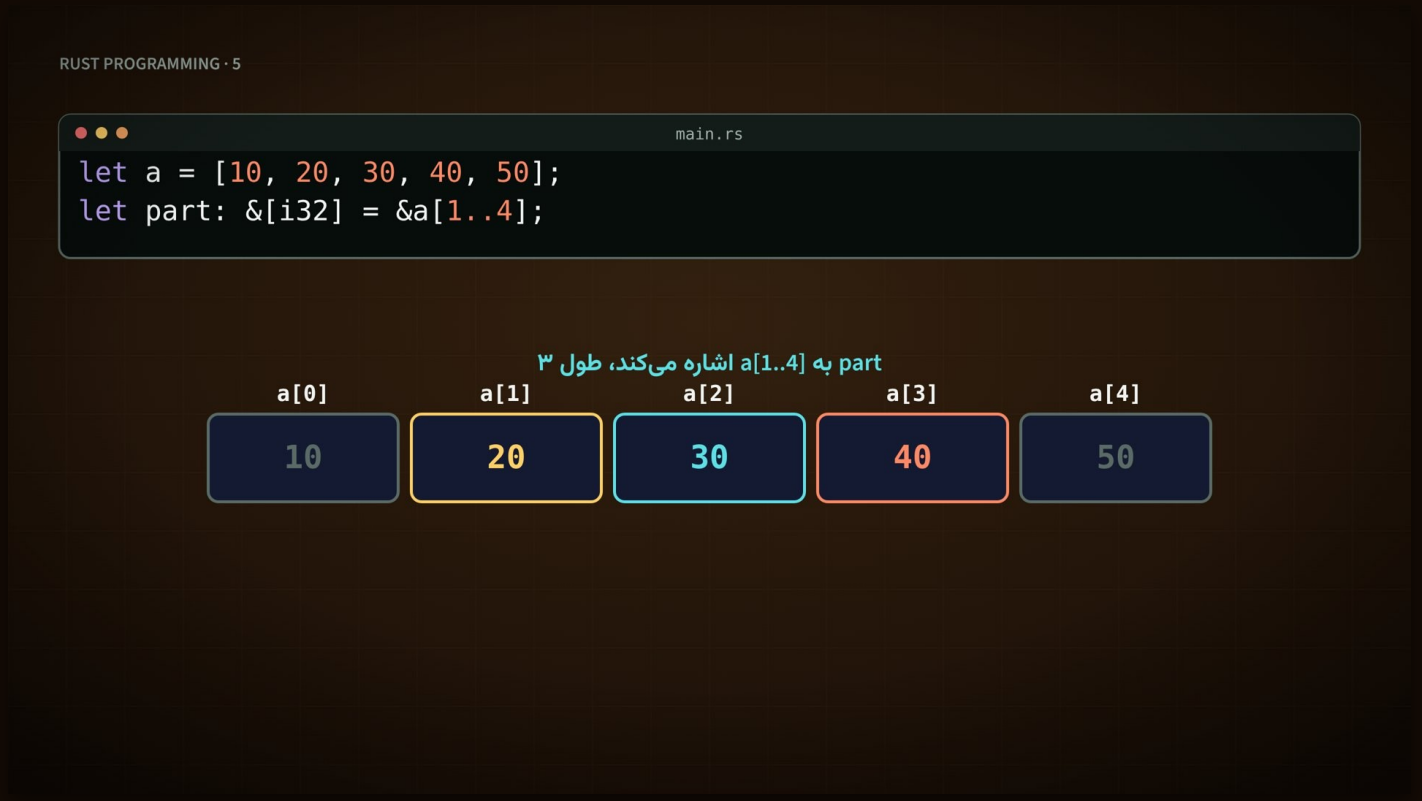
```rust
let word = "کابل";
word.len();                    // 8 bytes
word.chars().count();          // 4 characters
for c in word.chars() { print!("[{c}]"); }
let upper = "kabul".to_uppercase();
let joined = format!("{}-{}", "Kabul", 2026);
```

۵.۰۳

slice های آرایه

slice ها برای آرایه‌ها و vector ها هم کار می‌کنند:
&[T] یک تابع که یک slice می‌گیرد با ...

slice ها طول خودشان را می‌دانند، پس هیچ buffer overflow وجود ندارد.

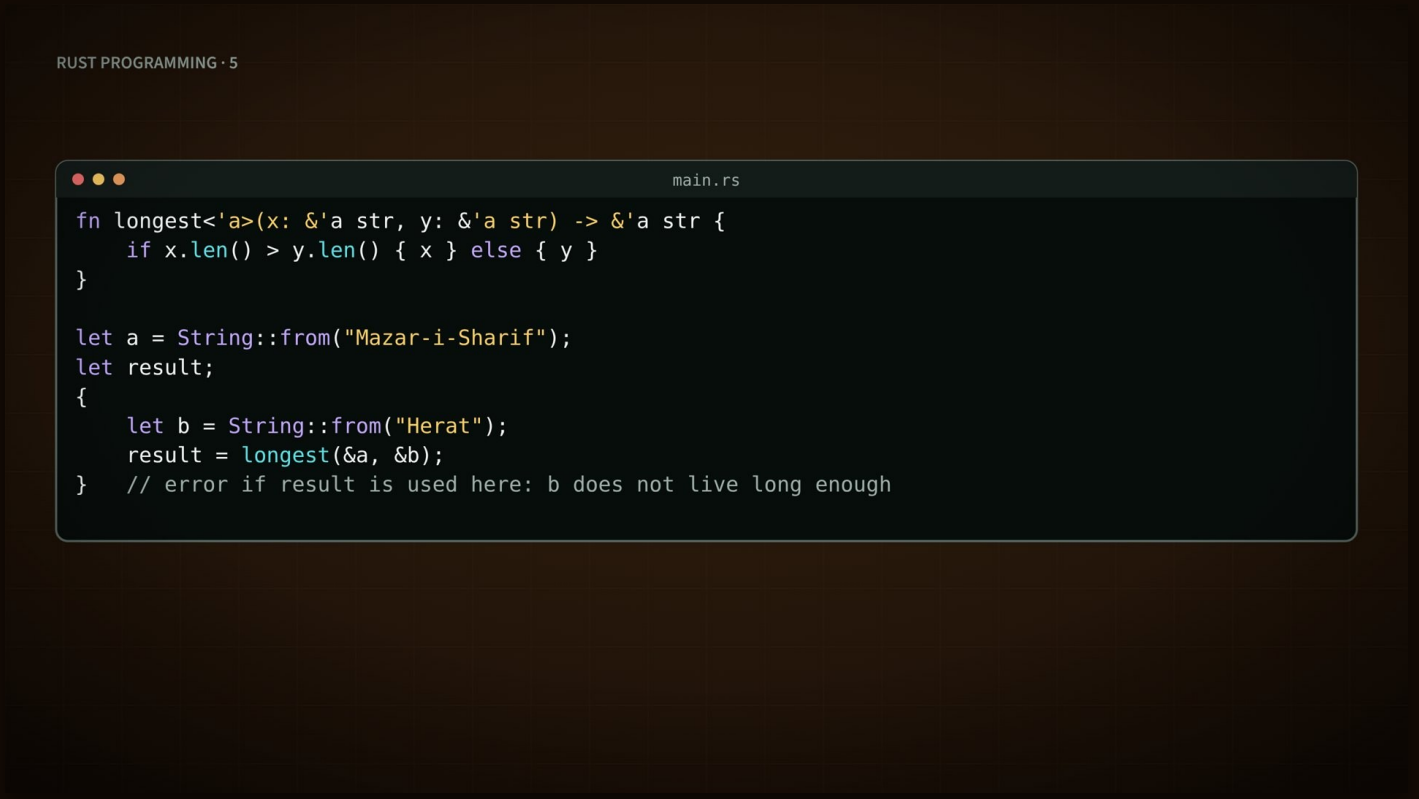


۵.۰۴

lifetime ها

یک reference هرگز نباید بیشتر از داده‌ای که به آن اشاره می‌کند زنده بماند. معمولاً کمپایلر ...

وقتی یک تابع یک referenceرا از یکی از چند ورودی برمی‌گرداند، یک lifetime مثل 'a می‌نویسید ...



۵.۰۵

یک کورس رایگان از MyWebUniversity

برنامه‌نویسی Rust

درس ۶

struct ها و method ها

تعریف struct ها، بلاک‌های impl، &self، تابع‌های associated و derive



تعریف struct ها

یک struct فیلدهای نام‌دار را گروه‌بندی می‌کند. با دادن مقدار به هر فیلد یک نمونه بسازید. ...

کل نمونه باید mut باشد تا هر فیلدی را تغییر دهید.

RUST PROGRAMMING · 6

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;
```

۶.۰۱

impl و method ها

method ها در یک بلاک impl زندگی می‌کنند.
پارامتر اول، یعنی &self، نمونه را قرض می‌گیرد. ...

method ها را با نقطه صدا کنید. Rust خودکار & را برای شما اضافه می‌کند.

RUST PROGRAMMING · 6

main.rs

```rust
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());
```

۶.۰۲

تابع‌های associated

تابع‌هایی در impl بدون self تابع‌های associated هستند، که با دو کالن صدا می‌شوند. new رسم ...

Self با S بزرگ یعنی خود نوع.

main.rs

```rust
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");
```

۶.۰۳

derive: Debug، Clone، PartialEq

attribute derive کود رایج را برای شما می‌نویسد. Debug اجازه می‌دهد با {:?} چاپ کنید، Clone ...

macro dbg! یک مقدار را با فایل و سطر آن چاپ می‌کند، برای debug سریع عالی است.

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);
```

۶.۰۴

tuple struct ها و الگوی newtype

tuple struct ها فیلدهای بدون نام دارند. یک tuple struct یک‌فیلدی، که newtype نامیده می‌شود، ...

حالا کمپایلر جلوی قاطی کردن متر و ثانیه را می‌گیرد.

RUST PROGRAMMING · 6

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 ✓
```

۶.۰۵

یک کورس رایگان از MyWebUniversity

برنامه‌نویسی Rust

درس ۷

enum ها، Option و match

enum ها با داده، Option به جای null، match، if let و let else



enum ها با داده

یک enum در Rust حالت‌های ممکن را فهرست می‌کند، و هر حالت می‌تواند دادهٔ خودش را داشته باشد. ...

این باعث می‌شود حالت‌های ناممکن قابل نمایش نباشند.

main.rs

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

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

۷.۰۱

Option: دیگر null نیست

Rust، null ندارد. مقداری که ممکن است نباشد نوع Option دارد: یا Some با یک مقدار، یا None.

کمپایلر شما را مجبور می‌کند None را مدیریت کنید، پس crash های null pointer اصلاً نمی‌توانند ...

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)
```

۷.۰۲

match

match یک مقدار را با pattern ها مقایسه می‌کند و اولین شاخه‌ای را که جور شود اجرا می‌کند. ...

pattern ها می‌توانند داده را از enum ها بیرون بکشند، و آندرسکور با هر چیز دیگر جور می‌شود.

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",
};
```

۷.۰۳

if let و let else

وقتی فقط به یک pattern اهمیت می‌دهید، if let از یک match کامل کوتاه‌تر است.

let else یک مقدار را می‌گیرد یا زود بیرون می‌شود، که مسیر اصلی را بدون تورفتگی نگه می‌دارد.

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
}
```

۷.۰۴

method ها روی enum ها

enum ها، درست مثل struct ها، می‌توانند بلاک‌های impl با method داشته باشند. داخل آن‌ها روی ...

خود Option و Result فقط enum هایی از کتابخانهٔ استاندارد هستند، با method های مفید زیاد.

RUST PROGRAMMING · 7

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,
        }
    }
}
```

۷.۰۵

یک کورس رایگان از MyWebUniversity

برنامه‌نویسی Rust

درس ۸

مدیریت خطا با Result

panic!، Result، عملگر ?، خطاهای سفارشی، thiserror و anyhow



panic!: خطاهای غیرقابل جبران

panic! پروگرام را با یک پیام متوقف می‌کند. برای باگ‌ها است، موقعیت‌هایی که هرگز نباید رخ دهند.

unwrap و expect روی None یا Err، panic می‌کنند. در مثال‌های سریع و آزمایش‌ها خوب هستند، اما ...

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
```

۸.۰۱

Result

تابع‌هایی که ممکن است ناکام شوند Result برمی‌گردانند: Ok با مقدار، یا Err با یک خطا. مثل ...

روی نتیجه match کنید، یا از method هایی مثل unwrap_or_else و map_err استفاده کنید.

RUST PROGRAMMING · 8

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);
```

۸.۰۲

عملگر ؟

علامت سوال بعد از یک Result، اگر خطایی باشد آن را زود برمی‌گرداند، یا اگر Ok باشد مقدار را ...

زنجیره‌های طولانی match را به کود پاک و کوتاه تبدیل می‌کند. main هم می‌تواند Result برگرداند.

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)
}
```

۸.۰۳

نوع‌های خطای سفارشی

برای کتابخانه‌ها، یک enum خطا با یک حالت برای هر ناکامی تعریف کنید. crate thiserror کود ...

تبدیل‌های From اجازه می‌دهند علامت سوال خطاهای سطح پایین‌تر را خودکار تبدیل کند.

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> { /* ... ? ... */ }
```

۸.۰۴

anyhow برای اپلیکیشن‌ها

در اپلیکیشن‌ها، crate anyhow یک نوع خطای آسان می‌دهد. context به هر خطا هنگام بالا رفتن یک ...

قاعدهٔ ساده: thiserror برای کتابخانه‌ها، anyhow برای اپ‌ها.

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(())
}
```

۸.۰۵

یک کورس رایگان از MyWebUniversity

برنامه‌نویسی Rust

درس ۹

collection ها، closure ها و iterator ها

Vec، HashMap، closure ها، adapter های iterator و collect



Vec

Vec یک آرایهٔ قابل رشد روی heap است. macro vec! یکی را با مقدارها می‌سازد. push به آخر ...

وقتی یک تابع فقط لازم دارد بخواند، Vec را به عنوان یک slice قرض بدهید.

main.rs

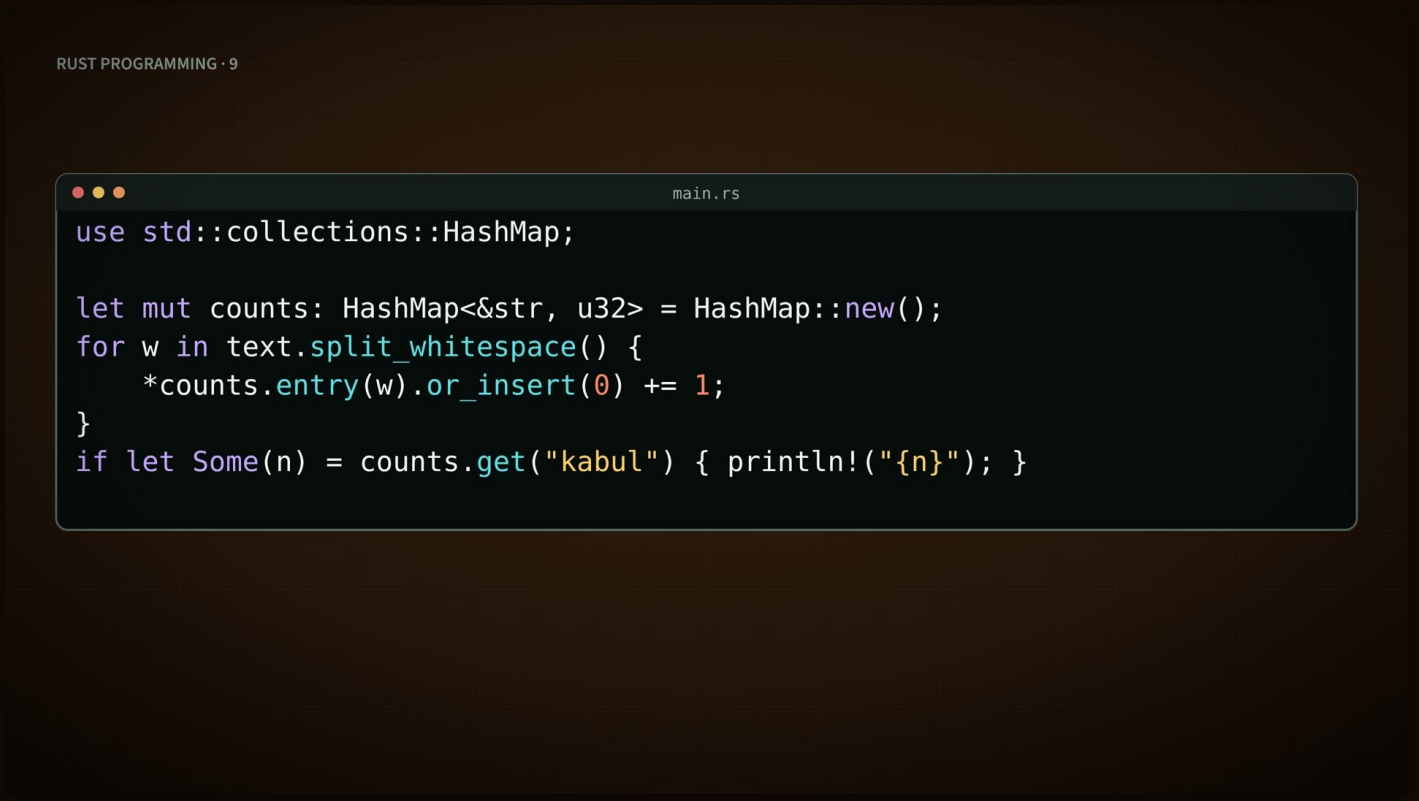
```rust
let mut scores = vec![90, 85, 77];
scores.push(92);
scores.len();                    // 4
scores[0];                       // 90
scores.get(10);                  // None, no panic
scores.sort();
scores.retain(|&s| s >= 80);     // keep 85, 90, 92
```

۹.۰۱

HashMap

HashMap جفت‌های کلید-مقدار را نگه می‌دارد. API entry شمردن و گروه‌بندی را یک سطر می‌سازد: ...

ترتیب پیمایش تعریف نشده است. وقتی کلیدهای مرتب لازم دارید از BTreeMap استفاده کنید.



۹.۰۲

closure ها

یک closure یک تابع بی‌نام است که پارامترهای آن بین دو خط عمودی نوشته می‌شوند. می‌تواند ...

closure، move را مجبور می‌کند مالکیت آنچه را می‌گیرد بگیرد، که برای thread ها لازم دارید.

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}");
```

۹.۰۳

adapter های iterator

scores.iter() — iterator

filter — |s| **s >= 80

map — |s| s * 2



۹.۰۴

یکجا کردن همه

اینجا رکوردهای شاگردان را می‌خوانیم، آن‌هایی را که کامیاب شدند نگه می‌داریم، اوسط نمره‌ها ...

into_iter یک collection را مصرف می‌کند، iter آن را قرض می‌گیرد، و iter_mut آن را به شکل ...

main.rs

```rust
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}");
}
```

۹.۰۵

یک کورس رایگان از MyWebUniversity

برنامه‌نویسی Rust

درس ۱۰

trait ها و generics

تعریف trait ها، method های پیش‌فرض، generics، trait bound ها، impl Trait و dyn Trait



trait ها

یک trait رفتاری را تعریف می‌کند که نوع‌ها می‌توانند شریک باشند، مثل یک interface. آن را ...

trait ها می‌توانند بدنهٔ پیش‌فرض method ها را فراهم کنند که پیاده‌کننده‌ها ممکن است ...

RUST PROGRAMMING · 10

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() }
}
```

۱۰.۰۱

trait های استاندارد: Display و From

Display را پیاده کنید تا کنترول کنید نوع شما با {} چطور چاپ شود. From را پیاده کنید تا ...

Iterator، Default، PartialOrd و Hash trait های دیگری هستند که اغلب می‌بینید.

RUST PROGRAMMING · 10

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();
```

۱۰.۰۲

generics و trait bound ها

تابع‌های generic برای نوع‌های زیاد کار می‌کنند.
یک trait bound، یعنی T: PartialOrd، می‌گوید ...

Rust برای هر نوعی که استفاده می‌کنید کود اختصاصی تولید می‌کند، پس generics هنگام اجرا هیچ ...

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
```

۱۰.۰۳

impl Trait و where

impl Trait در یک پارامتر هر نوع با آن trait را می‌پذیرد. در جای برگشت، نوع واقعی را پنهان ...

where، bound های طولانی را بعد از امضا می‌برد تا خوانا بماند.

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 { /* ... */ }
```

۱۰.۰۴

trait object ها: dyn Trait

generics نوع را هنگام کمپایل انتخاب می‌کنند.
وقتی یک لیست از نوع‌های مختلف با هم لازم ...

صدا کردن method ها هنگام اجرا از یک جدول می‌گذرد. کمی کندتر است اما خیلی انعطاف‌پذیر.



۱۰.۰۵

یک کورس رایگان از MyWebUniversity

برنامه‌نویسی Rust

درس ۱۱

smart pointer ها و همزمانی بی‌ترس

Box، Rc، RefCell، thread ها، closure های move، channel ها، Arc و Mutex



Box، Rc و RefCell

Box یک مقدار را با یک مالک روی heap می‌گذارد، برای نوع‌های بازگشتی مثل درخت‌ها لازم است. Rc ...

RefCell اجازه می‌دهد داده پشت یک reference شریک تغییر کند، با قاعده‌های قرض که هنگام اجرا ...

RUST PROGRAMMING · 11

نوع	به شما می‌دهد	thread؟
Box<T>	heap، یک مالک	✓
Rc<T>	مالکان شریک	✗
RefCell<T>	تغییر از طریق &	✗
Arc<T>	شریک، atomic	✓
Mutex<T>	تغییر با قفل	✓

۱۱.۰۱

ساختن thread ها

thread::spawn یک closure را روی یک thread نو سیستم عامل اجرا می‌کند. join روی handle منتظر …

closure به move ضرورت دارد تا مالک داده‌ای باشد که استفاده می‌کند، چون thread ممکن است …

RUST PROGRAMMING · 11

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();
```

۱۱.۰۲

channel ها

channel های mpsc مقدارها را بین thread ها می‌فرستند: تولیدکننده‌های زیاد، یک مصرف‌کننده. …

فرستادن یک مقدار آن را move می‌کند، پس دو thread هرگز نمی‌توانند همزمان از آن استفاده کنند.

۱۱.۰۳

RUST PROGRAMMING · 11

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}"); }
```

Arc و Mutex

برای شریک کردن وضعیت قابل تغییر بین thread ها، آن را در Arc<Mutex> بپیچانید. Arc مالکیت را …

lock یک guard برمی‌گرداند، و قفل وقتی guard از scope بیرون می‌رود خودکار آزاد می‌شود.

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
```

۱۱.۰۴

چرا بی‌ترس است

trait های Send و Sync نشان می‌دهند کدام نوع‌ها برای move یا شریک شدن بین thread ها امن ...

data race ها خطای کمپایل می‌شوند، نه باگ‌هایی که ساعت سه صبح دنبال آن‌ها می‌گردید. برای کود ...

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
```

۱۱.۰۵

یک کورس رایگان از MyWebUniversity

برنامه‌نویسی Rust

درس ۱۲

پروژه: یک ابزار خط فرمان



crate ها، clap، serde، module ها، آزمایش‌ها، ساخت release و قدم‌های بعدی

crate ها و Cargo.toml

کتابخانه‌ها در Rust، crate نامیده می‌شوند، که در crates.io منتشر می‌شوند. cargo add یکی را ...

از clap برای آرگومان‌های خط فرمان، از serde و serde_json برای JSON، و از anyhow برای خطاها ...

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"
```

۱۲.۰۱

آرگومان‌ها با clap

با derive در clap، یک struct گزینه‌های خط فرمان شما را توصیف می‌کند. clap آن‌ها را تحلیل و ...

doc comment ها توضیحات کمک می‌شوند.

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,
}
```

۱۲.۰۲

JSON با serde

RUST PROGRAMMING · 12

serde، struct های Rust را به JSON و برعکس تبدیل می‌کند. derive Deserialize را اضافه کنید، و ...

تابع main فایل را می‌خواند، تحلیل می‌کند، و یک گزارش چاپ می‌کند، با خطاهایی که علامت سوال ...

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(())
}
```

۱۲.۰۳

آزمایش‌ها و ساخت release

آزمایش‌ها را در یک module با نشانی cfg(test)، و تابع‌هایی با attribute test بنویسید. cargo ...

cargo build --release یک فایل اجرایی سریع و بهینه در target/release می‌سازد.

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
```

۱۲.۰۴

قدم‌های بعدی

بخوانید — doc.rust-lang.org/book

تمرین — تمرین‌های Rustlings

بسازید — tokio · axum · CLI



۱۲.۰۵

تشکر از توجه شما

MyWebUniversity.com

@MyWebUniversity-Dari همه‌را یاد بگیرید

آموزش رایگان تکنالوژی از سال ۲۰۰۰ — انگلیسی و دری