DOMINANDO RUST EM 1 ANO

Caderno de Exercícios

53 aulas de prática, do #01 ao #53

Rust 1.98 · edição 2024 · Cargo · 11 módulos · 67 questões · 2 desafios integradores

Autor Desconhecido · theknowledgeisle.com

Nome do aluno: Início: Término:

Como usar este caderno

Cada módulo reúne um bloco de aulas do curso, com a lista dos artigos que ele cobre. As perguntas conceituais pedem explicações com as suas palavras; os exercícios de código pedem programas: escreva, compile e execute cada um.

Tente sempre resolver sozinho antes de abrir o **Caderno de Respostas**. É no erro que o conhecimento fixa.

Sumário

Módulo	Tema	Aulas	Questões
MÓDULO 1	Fundamentos da linguagem	#01–#04	Q1–Q6
MÓDULO 2	Ownership e borrowing	#05–#06	Q7–Q11
MÓDULO 3	Tipos próprios e erros	#07–#09	Q12–Q17
MÓDULO 4	Coleções, iteradores e closures	#10–#11	Q18–Q23
MÓDULO 5	Traits, generics e lifetimes	#12–#14	Q24–Q29
MÓDULO 6	Módulos e testes	#15–#16	Q30–Q32
MÓDULO 7	Recursos avançados	#17–#21	Q33–Q42
MÓDULO 8	Ecossistema e web	#22–#26	Q43–Q48
MÓDULO 9	Sistemas e baixo nível	#27–#36	Q49–Q56
MÓDULO 10	Engenharia de projetos	#37–#41	Q57–Q61
MÓDULO 11	Rust em domínios específicos e o futuro	#42–#53	Q62–Q67
DESAFIOS	Desafios integradores	todas	D1–D2

Legenda: nas perguntas, `assim` indica código, nomes de tipos e comandos. As linhas pautadas são para a sua resposta; nos exercícios de código, use também o computador: **todo programa deve ser compilado e executado**.

MÓDULO 1

Fundamentos da linguagem

Por que Rust existe, variáveis e imutabilidade, funções e expressões, e o controle de fluxo.

Artigos 01 a 04

#01 Por que Rust existe e o que ele quer de você
#02 Variáveis, Tipos e a Arte da Imutabilidade
#03 Funções, Expressões e Como Rust Pensa Diferente sobre Retorno de Valores
#04 Controle de Fluxo — if, loop, while e for como você nunca viu antes

Perguntas conceituais

Q1 **O que Rust quer de você**

O artigo #01 diz que Rust oferece segurança de memória sem coletor de lixo. Que duas classes de *bugs* comuns em C e C++ o compilador do Rust impede, e qual é o preço que o programador paga por isso?

Q2 **Imutável por padrão**

Por que `let x = 5; x = 6;` não compila em Rust? Qual a diferença entre `let mut`, *shadowing* (`let x = x + 1;`) e `const`?

Q3 **Expressão ou instrução**

Por que a função abaixo não compila, e como corrigi-la com uma mudança de um caractere?

```
fn dobro(x: i32) -> i32 {
    x * 2;
}
```

Q4 **Estouro de inteiro**

O que acontece em `let x: u8 = 255; let y = x + 1;` no modo *debug* e no modo *release*? Que métodos tornam a intenção explícita?

Exercícios de código

Q5 Tipos e conversões

Escreva um programa que mostre: o maior `u8` e o menor `i16`; a conversão com `as` de `300_i32` para `u8` e de `-1_i32` para `u32`; a conversão segura com `u8::try_from`; e as quatro formas de somar `250_u8 + 10` (`checked`, `wrapping`, `saturating` e `overflowing`).

Q6 Expressões que devolvem valores

Escreva `fn classificar(nota: u32) -> &'static str` usando `match` com faixas (`90..=100`), uma função `fn primeiro_divisor(n: u32) -> u32` que use `loop` com `break valor`, e um laço com rótulo (`'externo`) que pare os dois laços quando achar o primeiro par `(a, b)` com `a * b == 42` e `a < b`.

MÓDULO 2

Ownership e borrowing

Posse, movimento e cópia; referências, empréstimos mutáveis e imutáveis, e slices.

Artigos 05 a 06

#05 Ownership — A Ideia que Muda Tudo
#06 Borrowing e Referências — Usando sem Possuir

Perguntas conceituais

Q7 As três regras da posse

Enuncie as regras de *ownership* e explique por que `let s1 = String::from("oi"); let s2 = s1; println!("{s1}");` não compila, enquanto o mesmo código com `i32` compila.

Q8 Uma mutável ou várias imutáveis

Qual é a regra dos empréstimos, e que *bug* clássico de C++ ela impede no código abaixo?

```
let mut v = vec![1, 2, 3];
let primeiro = &v[0];
v.push(4);
println!("{primeiro}");
```

Q9 **String e &str**

Qual a diferença entre `String` e `&str`? Por que uma função que só lê um texto deve receber `&str`?

Exercícios de código

Q10 **Movimento, cópia e empréstimo**

Escreva um programa que mostre: (a) uma `String` movida para uma função e devolvida no retorno; (b) a mesma função reescrita para receber `&str`; (c) uma função que recebe `&mut String` e acrescenta texto; (d) um `struct` com `#[derive(Clone, Copy)]` sendo copiado.

Q11 Slices e a primeira palavra

Escreva `fn primeira_palavra(s: &str) -> &str` que devolva a primeira palavra (até o primeiro espaço) como uma fatia do texto original, sem alocar. Mostre que tentar esvaziar a `String` (`clear`) enquanto a fatia ainda está em uso não compila (em comentário), e use a função com um literal e com uma `String`.

MÓDULO 3

Tipos próprios e erros

Structs e métodos, enums e pattern matching, e o tratamento de erros com Result.

Artigos 07 a 09

#07 Structs — Criando seus próprios tipos de dados
#08 Enums e Pattern Matching — O Sistema Mais Expressivo que Você Já Viu
#09 Tratamento de Erros com Result — Erros como Valores, não Exceções

Perguntas conceituais

Q12 **Option no lugar de null**

Por que Rust não tem `null`? O que o `Option<T>` obriga o programador a fazer que um ponteiro nulo não obriga?

Q13 **O operador ?**

O que o operador `?` faz numa função que devolve `Result<T, E>`? Por que `unwrap()` é aceitável num exemplo e perigoso numa biblioteca?

Q14 **Enum com dados**

Como um `enum` do Rust difere de um `enum` do C? Mostre um `enum Mensagem` com quatro variantes de formatos diferentes.

Exercícios de código

Q15 Uma struct com métodos

Escreva `struct Retangulo { largura: f64, altura: f64 }` com um construtor associado `quadrado(lado)`, os métodos `area(&self)`, `escalar(&mut self, fator)` e `cabe_em(&self, outro: &Retangulo) -> bool`, e implemente o *trait* `Display` para imprimi-lo como `3x4`.

Q16 Uma máquina de estados com enum

Modele um pedido com `enum Estado { Novo, Pago { valor: u32 }, Enviado { codigo: String }, Cancelado { motivo: String } }` e escreva `fn avancar(estado: Estado, evento: Evento) -> Result<Estado, String>` com `match` sobre a tupla `(estado, evento)`. Rode uma sequência válida e duas inválidas.

Q17 **Erros próprios e o operador ?**

Escreva `fn ler_config(texto: &str) -> Result<Config, ErroConfig>` que leia linhas `chave=valor` com as chaves `porta` (número de 1 a 65535) e `host` (obrigatório). Defina `enum ErroConfig` com as variantes `LinhaInvalida(usize)`, `Faltando(&'static str)` e `Porta(std::num::ParseIntError)`, implemente `Display` e `From<ParseIntError>`, e use o `?`.

MÓDULO 4

Coleções, iteradores e closures

Vec, HashMap e HashSet, iteradores, adaptadores e closures.

Artigos 10 a 11

#10 Coleções — Vec, HashMap e HashSet na Prática
#11 Iteradores e Closures — O Estilo Funcional de Rust

Perguntas conceituais

Q18 **iter, iter_mut e into_iter**

Qual a diferença entre percorrer um `Vec<String>` com `v.iter()`, `v.iter_mut()` e `v.into_iter()` (ou `for x in v`)? O que acontece com `v` depois do último?

Q19 **Iteradores preguiçosos e de custo zero**

Por que `v.iter().map(|x| x * 2);` sozinho não faz nada (e o compilador avisa)? O que significa dizer que os iteradores do Rust são uma "abstração de custo zero"?

Q20 **Fn, FnMut e FnOnce**

Qual a diferença entre os *traits* `Fn`, `FnMut` e `FnOnce`? Dê uma closure de cada tipo.

Exercícios de código

Q21 Contando palavras com HashMap

Conte a frequência das palavras de um texto (minúsculas, sem pontuação) com um `HashMap<String, usize>` e a API `entry`, mostre as três mais frequentes desempatando em ordem alfabética, e use um `HashSet` para listar as palavras que aparecem só uma vez, em ordem.

Q22 Uma cadeia de iteradores

Dado um vetor de pedidos (`cliente`, `valor` em centavos, `pago`), use só iteradores, sem laços explícitos, para obter: o total pago; o maior pedido pago (com `max_by_key`); os clientes com pedido pago, sem repetição e em ordem; os pedidos em lotes de 2 (`chunks`); e se algum pedido passa de R$ 150 (`any`).

Q23 Closures que capturam

Escreva `fn criar_contador(inicio: i32, passo: i32) -> impl FnMut() -> i32`, que devolve uma closure com estado próprio; uma função genérica `fn aplicar_duas_vezes<F: Fn(i32) -> i32>(f: F, x: i32) -> i32`; e mostre o `move` necessário para devolver uma closure que captura uma variável local.

MÓDULO 5

Traits, generics e lifetimes

Comportamento compartilhado com traits, código genérico e a anotação de lifetimes.

Artigos 12 a 14

#12 Traits — Definindo Comportamento Compartilhado
#13 Generics — Código que Funciona para Qualquer Tipo
#14 Lifetimes — Quando o Compilador Precisa de Mais Informação

Perguntas conceituais

Q24 **Genérico ou dyn Trait**

Compare `fn desenhar<T: Forma>(f: &T)` e `fn desenhar(f: &dyn Forma)`. Quando usar cada um?

Q25 **A regra do órfão**

Por que você pode implementar o seu *trait* `Resumo` para `Vec<i32>`, e o *trait* `Display` para o seu `struct Livro`, mas não pode implementar `Display` para `Vec<i32>`? Como contornar?

Q26 **Por que lifetimes**

Por que esta função não compila sem a anotação de *lifetime*? O que `'a` promete?

```
fn maior(a: &str, b: &str) -> &str {
    if a.len() >= b.len() { a } else { b }
}
```

Exercícios de código

Q27 Um trait com método padrão

Defina `trait Descrevivel` com `fn nome(&self) -> String` e um método **padrão** `fn descricao(&self) -> String` que use o `nome`. Implemente para `Livro` (sobrescrevendo a descrição) e para `Revista` (usando o padrão). Escreva uma função com `impl Descrevivel` e outra que percorra um `Vec<Box<dyn Descrevivel>>`.

Q28 Uma função genérica com restrições

Escreva `fn maior_e_menor<T: PartialOrd + Copy>(itens: &[T]) -> Option<(T, T)>` e uma `struct Par<T>` com um método `maior(&self) -> &T` disponível só quando `T: PartialOrd`. Teste com inteiros, `f64` e caracteres, e com um slice vazio.

Q29 Uma struct que empresta

Escreva `struct Analisador<'a> { texto: &'a str }` com um método `palavras_longas(&self, minimo: usize) -> Vec<&'a str>` que devolva fatias do texto original, e uma função `fn maior<'a>(a: &'a str, b: &'a str) -> &'a str`. Mostre (em comentário) o uso que o compilador recusa.

MÓDULO 6

Módulos e testes

Módulos, visibilidade e organização de projetos; testes unitários, de integração e de documentação.

Artigos 15 a 16

#15 Módulos e Organização de Código — Estruturando Projetos Reais
#16 Testes — Escrevendo Código que Prova que Seu Código Funciona

Perguntas conceituais

Q30 Crate, módulo e visibilidade

Qual a diferença entre *crate*, *package* e módulo? O que `pub`, `pub(crate)` e a ausência de modificador significam para uma função dentro de `mod banco`?

Q31 Onde ficam os testes

No Rust, por que os testes unitários ficam no mesmo arquivo do código (`#[cfg(test)] mod tests`), e os de integração na pasta `tests/`? O que um teste de integração não consegue acessar?

Exercícios de código

Q32 **Módulos e testes num arquivo só**

Num único `main.rs`, crie o módulo `carrinho` com a `struct Carrinho` pública, um campo privado com os itens, os métodos públicos `novo`, `adicionar` e `total`, e uma função privada `aplicar_desconto` (10% a partir de 3 itens). Escreva, dentro do módulo, um `mod tests` com três testes, inclusive um da função privada e um com `#[should_panic]`. Rode o programa e, separadamente, os testes com `rustc --test`.

MÓDULO 7

Recursos avançados

Closures avançadas, smart pointers (Box, Rc e RefCell), concorrência com threads e canais, async/await com Tokio e macros.

Artigos 17 a 21

#17 Closures Avançadas e Programação Funcional — Indo Além do map e filter
#18 Smart Pointers — Box, Rc e RefCell
#19 Concorrência — Threads, Canais e a Promessa de Rust de Concorrência Sem Medo
#20 Async/Await — Concorrência sem Threads com Tokio
#21 Macros — Escrevendo Código que Escreve Código

Perguntas conceituais

Q33 **Box, Rc e RefCell**

Para que serve cada um: `Box<T>`, `Rc<T>` e `RefCell<T>`? Por que `Rc<RefCell<T>>` é uma combinação comum, e qual é o risco dela?

Q34 **Send e Sync**

O que significam os *traits* `Send` e `Sync`? Por que `thread::spawn` recusa uma closure que captura um `Rc<T>`, e aceita uma com `Arc<T>`?

Q35 **Thread ou async**

Quando usar threads e quando usar `async` com Tokio? Por que chamar uma função bloqueante (`std::thread::sleep`, leitura síncrona de arquivo grande) dentro de uma tarefa `async` é um erro?

Q36 **macro_rules ou função**

Quando uma macro declarativa (`macro_rules!`) é justificada, e quando uma função resolve? Por que `println!` e `vec!` precisam ser macros?

Exercícios de código

Q37 **Uma expressão recursiva com Box**

Defina `enum Expr { Num(f64), Soma(Box<Expr>, Box<Expr>), Mult(Box<Expr>, Box<Expr>), Neg(Box<Expr>) }`, uma função `avaliar(&Expr) -> f64` e uma `mostrar(&Expr) -> String` que imprima com parênteses. Monte e avalie `-(2 + 3) * 4`.

Q38 Uma árvore com Rc, RefCell e Weak

Monte uma árvore de categorias (`Livros → Ficção → Romance`) em que cada nó tem `filhos: RefCell<Vec<Rc<No>>>` e `pai: RefCell<Weak<No>>`. Mostre o caminho de um nó até a raiz, as contagens forte e fraca, e que todos os nós são liberados no fim (com um `Drop` que imprime).

Q39 Soma em paralelo com scoped threads

Some os números de 1 a 10 milhões dividindo o intervalo entre 4 threads com `std::thread::scope`, que permite às threads emprestar dados da pilha do `main` sem `Arc`. Depois faça um contador compartilhado com `Arc<Mutex<u64>>` e outro com `AtomicU64`, e confira os três resultados.

Q40 Produtor e consumidor com canais

Use `std::sync::mpsc` para que três produtores enviem pedidos (`(produtor, número)`) a um consumidor, que os processa até todos os produtores terminarem. Mostre como o consumidor sabe que acabou e por que o `drop` do transmissor original é necessário.

Q41 Tarefas assíncronas com Tokio

Num projeto Cargo com `tokio = { version = "1", features = ["full"] }`, escreva um programa que dispare três "consultas" a fornecedores com tempos diferentes (`tokio::time::sleep`), espere todas com `join!`, e outra rodada em que só vale a primeira que responder (`select!`), com um tempo limite (`timeout`) para a mais lenta.

Q42 Uma macro declarativa

Escreva `macro_rules! mapa` que crie um `HashMap` a partir de `mapa!{ "a" => 1, "b" => 2 }`, aceitando vírgula final, e uma macro `contar!` que conte quantos argumentos recebeu, em tempo de compilação. Use as duas.

MÓDULO 8

Ecossistema e web

Cargo e as crates essenciais, uma API REST com Axum, banco de dados com SQLx, autenticação JWT e testes avançados.

Artigos 22 a 26

#22 O Ecossistema Rust — Cargo, crates.io e as Bibliotecas Essenciais
#23 Construindo uma API REST com Axum
#24 Banco de Dados com SQLx — Persistência Real para sua API
#25 Autenticação JWT — Protegendo sua API com Tokens
#26 Testes Avançados — Propriedades, Mocks e Benchmarks

Perguntas conceituais

Q43 **Cargo.toml e Cargo.lock**

O que significa `serde = "1.0"` no `Cargo.toml`? Por que o `Cargo.lock` é versionado num executável e, tradicionalmente, não era numa biblioteca?

Q44 **Extractors do Axum**

No Axum, um *handler* como `async fn criar(State(db): State<Pool>, Json(novo): Json<NovoLivro>) -> impl IntoResponse` não lê a requisição manualmente. Como os parâmetros são preenchidos, e o que acontece quando o JSON do corpo é inválido?

Q45 Consultas verificadas na compilação

O que a macro `sqlx::query_as!` faz de diferente de uma consulta montada em texto? Qual a exigência dela para compilar?

Q46 Testes de propriedade

O que é um teste baseado em propriedades (com `proptest` ou `quickcheck`), e que propriedade testaria uma função que codifica e decodifica Base64?

Exercícios de código

Q47 Uma API com Axum

Escreva uma API Axum com `GET /livros`, `GET /livros/{id}` (404 quando não existe) e `POST /livros` (201 com o livro criado, ou 422 se o título estiver vazio), guardando os livros num `Arc<Mutex<Vec<Livro>>>` como estado. Use `serde` para o JSON.

Q48 **Um teste de propriedade feito à mão**

Sem bibliotecas, escreva um gerador pseudoaleatório simples (*xorshift*), uma função `codificar_rle(&str) -> String` e `decodificar_rle(&str) -> String` (codificação por repetição: "aaab" vira "3a1b"), e teste a propriedade de ida e volta em 1000 textos aleatórios feitos das letras `a`, `b` e `c`. Mostre o que acontece com um texto que contém dígitos, e por quê.

MÓDULO 9

Sistemas e baixo nível

WebAssembly, programação de sistemas e de redes, parsing, criptografia, embarcados, unsafe, desempenho, FFI e macros procedurais.

Artigos 27 a 36

#27 WebAssembly com Rust — Executando Rust no Navegador
#28 Programação de Sistemas — Arquivos, Processos e Sinais
#29 Programação de Redes — TCP, UDP e Sockets
#30 Parsing e Processamento de Texto — nom, pest e Expressões Regulares
#31 Criptografia e Segurança — Hashing, Criptografia Simétrica e TLS
#32 Embarcados e Bare Metal — Rust sem Sistema Operacional
#33 Unsafe Rust — Quando e Como Usar Código Inseguro com Responsabilidade
#34 Performance e Otimização — Profiling, SIMD e Técnicas Avançadas
#35 FFI — Integrando Rust com C e Outras Linguagens
#36 Macros Procedurais — Criando Derive Macros e Atributos Customizados

Perguntas conceituais

Q49 **Quando unsafe**

O que um bloco `unsafe` permite que o Rust seguro não permite? Ele desliga o *borrow checker*? Qual a responsabilidade de quem escreve uma função segura que usa `unsafe` por dentro?

Q50 **no_std**

O que muda num programa `#![no_std]` para um microcontrolador? O que deixa de existir, e o que continua disponível?

Q51 Um protocolo sobre TCP

Por que um servidor TCP não pode assumir que um `read` traz uma mensagem completa? Que duas formas de delimitar mensagens são comuns?

Q52 Medir antes de otimizar

Que ferramentas o curso usa para achar o gargalo de um programa Rust lento, e por que olhar o código e "adivinhar" costuma falhar?

Exercícios de código

Q53 Arquivos com buffer e processos

Escreva um programa que grave 10 mil linhas num arquivo com `BufWriter`, releia contando as linhas e as que contêm "7" com `BufReader`, e execute um processo externo (`wc -l`) com `std::process::Command`, conferindo que a contagem bate. Trate os erros com `?` e `Box<dyn Error>`.

Q54 Um servidor TCP de eco por linhas

Com `std::net`, suba um servidor TCP numa porta escolhida pelo sistema (`127.0.0.1:0`), que atenda cada conexão numa thread e responda cada linha recebida em maiúsculas, encerrando a conexão com a linha `sair`. No mesmo programa, conecte dois clientes e mostre as respostas.

Q55 split_at_mut com unsafe

A biblioteca padrão implementa `split_at_mut`, que divide um slice mutável em duas partes mutáveis. Escreva a sua versão, `dividir_mut(v: &mut [i32], meio: usize) -> (&mut [i32], &mut [i32])`, mostrando por que a versão segura não compila (em comentário) e usando `unsafe` com ponteiros crus e um comentário `SAFETY`. Use-a para alterar as duas metades.

Q56 Chamando C com FFI

Declare as funções `strlen` e `abs` da biblioteca C num bloco `unsafe extern "C"` (a sintaxe da edição 2024) e chame-as a partir do Rust, convertendo uma `&str` com `CString`. Mostre o que acontece ao tentar criar uma `CString` com um `\0` no meio.

MÓDULO 10

Engenharia de projetos

Padrões de projeto em Rust, compilação condicional e features, documentação e doc tests, publicação de crates e ferramentas de linha de comando.

Artigos 37 a 41

#37 Design Patterns em Rust — Adaptando Padrões Clássicos à Linguagem
#38 Compilação Condicional e Feature Flags — Gerenciando Variações do Código
#39 Documentação e Doc Tests — Escrevendo Documentação que Não Fica Desatualizada
#40 Gerenciamento de Dependências e Publicação de Crates — Do Projeto Local ao crates.io
#41 Rust para CLI — Construindo Ferramentas de Linha de Comando Profissionais com clap

Perguntas conceituais

Q57 Padrões que mudam em Rust

Por que alguns padrões clássicos da orientação a objetos mudam de forma em Rust? Dê o exemplo do Builder e do State (*typestate*).

Q58 Features no Cargo

O que são *features* no `Cargo.toml`, e por que elas devem ser **aditivas**? Dê um exemplo de *feature* bem e mal projetada.

Q59 Documentação que não mente

O que são os *doc tests*, e por que eles evitam que a documentação fique desatualizada?

Exercícios de código

Q60 Typestate: transições que não compilam

Implemente um pedido com *typestate*: `struct Pedido<E> { id: u32, estado: E }` com os estados `Novo`, `Pago { valor: u32 }` e `Enviado { codigo: String }` como tipos, e os métodos `pagar` (só em `Pedido<Novo>`), `enviar` (só em `Pedido<Pago>`) e `rastreio` (só em `Pedido<Enviado>`). Mostre em comentários as chamadas que o compilador recusa.

Q61 Uma CLI sem dependências

Escreva o núcleo de uma ferramenta de linha de comando que conte linhas, palavras e bytes (como o `wc`), com as opções `-l`, `-w`, `-c` e `--help`, analisando os argumentos com um `match` sobre slices e devolvendo `Result` para opções inválidas. Para o exemplo ser reprodutível, passe os argumentos como um vetor e o texto como uma string.

MÓDULO 11

Rust em domínios específicos e o futuro

Ciência de dados, jogos, sistemas distribuídos, compiladores, blockchain, machine learning, segurança, produção, comunidade, o futuro da linguagem e os projetos integradores.

Artigos 42 a 53

#42 Rust para Data Science — Processamento de Dados com Polars e ndarray
#43 Rust para GameDev — Motores de Jogo, ECS e Bevy
#44 Sistemas Distribuídos — Consenso, Mensageria e Tolerância a Falhas
#45 Compiladores e Interpretadores — Construindo uma Linguagem com Rust
#46 Rust e Blockchain — Smart Contracts, Criptografia Aplicada e Web3
#47 Rust e Machine Learning — Inferência, ONNX e Aceleração de Modelos
#48 Segurança e Auditoria — Fuzzing, Análise Estática e Hardening de Código Rust
#49 Rust em Produção — Observabilidade, Métricas e Operação de Sistemas Rust
#50 Contribuindo com o Ecossistema Rust — Open Source, RFCs e a Comunidade
#51 O Futuro do Rust — Async Traits, GATs, Especialização e o Roadmap da Linguagem
#52 Projetos Integradores — Construindo Sistemas Completos do Zero
#53 Epílogo — O que Vem Depois, Como Continuar Crescendo, e a Filosofia do Rust

Perguntas conceituais

Q62 **ECS no Bevy**

O Bevy organiza os jogos com ECS. Como um sistema do Bevy declara os dados de que precisa, e por que isso permite ao motor rodar sistemas em paralelo com segurança?

Q63 **Consenso**

Num sistema distribuído com três réplicas de um banco, por que é preciso um algoritmo de consenso como o Raft? O que acontece quando a rede separa uma réplica das outras duas?

Q64 Fuzzing

O que é *fuzzing*, e por que ele encontra *bugs* em código Rust "seguro"? Que tipo de função é a melhor candidata?

Q65 Como o Rust evolui

O que são as **edições** do Rust (2018, 2021, 2024), e como elas permitem mudar a linguagem sem quebrar o código existente? Qual o papel das RFCs?

Exercícios de código

Q66 Uma cadeia de blocos com hash

Para entender a ideia por trás de um *blockchain*, monte uma cadeia de blocos em que cada bloco guarda os dados e o *hash* do anterior, calculado com o `DefaultHasher` da biblioteca padrão (que **não** é criptográfico: serve só para o exemplo). Escreva `valida(&cadeia) -> Result<(), usize>` e mostre que alterar um bloco do meio invalida a cadeia a partir dele.

Q67 Um interpretador de expressões

Escreva um analisador de expressões aritméticas com precedência e parênteses, por **descida recursiva**: um *tokenizer* e as funções `expressao` (soma e subtração), `termo` (multiplicação e divisão) e `fator` (número, negativo ou parênteses), que devolvem `Result<f64, String>`. Avalie quatro expressões, uma com erro.

PARA IR ALÉM

Desafios integradores

Projetos maiores, que juntam vários módulos. Não há uma única resposta certa: o gabarito mostra um caminho.

D1

Uma calculadora de terminal

Escreva uma calculadora de terminal com sete operações: soma, subtração, multiplicação, divisão, potência (expoente inteiro), porcentagem (`15 % 200` é 15% de 200) e raiz quadrada (`raiz 2`). Ela deve: (a) aceitar vírgula ou ponto como separador decimal; (b) guardar o último resultado em `ans` , utilizável na conta seguinte; (c) recusar divisão por zero, raiz de negativo, expoente fracionário, resultado infinito e texto que não é número com uma mensagem clara, **sem encerrar o programa**; (d) ter os comandos `hist` (histórico) e `sair` . Separe o núcleo (um módulo `calc` , sem E/S, com um `enum` de erros e `Result`) da interface (`main`), e escreva testes para o núcleo.

Sua solução (arquitetura, decisões e código principal):

D2

Um servidor chave-valor concorrente

Escreva um servidor TCP que guarde pares chave-valor em memória, com os comandos de texto `SET chave valor`, `GET chave`, `DEL chave` e `KEYS`, uma linha por comando, atendendo vários clientes ao mesmo tempo. Acrescente expiração opcional (`SET chave valor EX 10`), persistência num arquivo de log e testes de integração que conectem clientes reais.

Sua solução (arquitetura, decisões e código principal):

Bom estudo e bons exercícios!

Dominando Rust em 1 Ano · theknowledgeisle.com