



RUST سوالات

دانشجو : علی قنبری پور

1- زبان برنامه نویسی Rust چیست؟ Rust یک زبان برنامه نویسی سیستمی است که برای عملکرد بالا، امنیت حافظه و همزمانی طراحی شده است. این زبان توسط موزیلا توسعه داده شد و اکنون به صورت مستقل توسط بنیاد Rust مدیریت میشود.

2 - ویژگی های زبان Rust را به اختصار توضیح دهید؟

- مدیریت حافظه ایمن بدون نیاز به (garbage collector)
- همزمانی ایمن عدم وجود (data race)
- سرعت بالا و کارایی بهینه
- قابلیت برنامه نویسی سیستمی و سطح پایین
- پشتیبانی از برنامه نویسی تابعی و شیءگرا
- مدیریت خطا به صورت کارآمد

3- زبان Rust در چه حوزه هایی کاربرد بیشتری دارد؟

- توسعه سیستمهای عامل
- موتورهای بازی
- برنامههای تحت وب با استفاده از WebAssembly
- نرمافزارهای توزیع شده
- ابزارهای خط فرمان
- بلاکچین و قراردادهای هوشمند

4 تفاوت زبان Rust با زبان های C و ++C چیست؟

- ایمنی حافظه بدون نیاز به garbage collector
- جلوگیری از باگهای مربوط به اشاره گر ها (null pointer, dangling pointer)
- مدیریت مالکیت داده ها (Ownership)

. جلوگیری از data race

. کدنویسی مدرنتر و راحتتر

5 محدودیت های زبان Rust را توضیح دهید ؟

. یادگیری دشوارتر نسبت به زبانهای سطح بالا

. زمان کامپایل طولانیتر

. کتابخانههای کمتر نسبت به زبانهایی مانند ++C

6 درمورد ابزار Cargo توضیح دهید؟

Cargo ابزار مدیریت بسته (package manager) و ساخت (build system)

برای Rust است که کارهایی مانند کامپایل، اجرای تستها و مدیریت وابستگیها را انجام میدهد.

7 فایل Cargo.lock چه عملکردی دارد؟

فایل Cargo.lock نسخه دقیق وابستگیها را قفل میکند تا اطمینان حاصل شود که همه وابستگیها در طول توسعه ثابت میمانند.

8 در مورد IDE مطرح RustRover توضیح دهید؟

RustRover یک محیط توسعه یکپارچه (IDE) برای Rust است که توسط JetBrains

توسعه داده شده و امکاناتی مانند تکمیل کد، دیباگ و مدیریت پروژه را ارائه میدهد

9 چند مورد از برنامه هایی که به زبان Rust نوشته شده اند را نام ببرید؟

. سیستمعامل Redox

. موتور بازی Amethyst

. مرورگر Servo

. ابزار خط فرمان ripgrep

. بلاکچین Solana

10 قابلیت های زبان Rust در Concurrency و Parallelism را به اختصار و با ذکر مثال بیان کنید؟

ایمنی حافظه: (Data Race-Free)

- برنامه‌های چند نخه (Multithreaded) نوشتن برنامه‌هایی که همزمان روی داده‌ها کار میکنند بدون نگرانی از خطاهای ناشی از دسترسی همزمان.
- سیستم‌های حساس به پایداری در سیستم‌های Embedded و نرم‌افزارهای سیستمی که خطاهای حافظه خطرناک هستند.

Arc اشتراک امن داده‌ها بین thread ها:

- وبسروورها: اشتراک‌گذاری درخواست‌ها یا داده‌های مشترک بین چندین thread برای پاسخدهی سریعتر.
- پایگاه داده‌ها: دسترسی همزمان به رکوردهای ثابت و بدون تغییر. (Read-Only)

Mutex و مرسائی داده‌های mutable:

- پردازش‌های موازی: به‌روزرسانی داده‌های مشترک مانند cache یا وضعیت کلی سیستم.
- سیستم‌های گیمینگ و گرافیکی: همگام‌سازی داده‌های mutable بین پردازش‌های گرافیکی و فیزیکی.

Rayon پردازش موازی داده‌ها:

- پردازش داده‌های حجیم: پردازش مجموعه‌های بزرگ مانند تصاویر، ویدیو یا داده‌های علمی.
- Machine Learning و AI: پردازش موازی برای بهینه‌سازی الگوریتم‌های یادگیری.
- تحلیل داده (Data Analysis): پردازش سریع‌تر روی حجم زیادی از داده‌ها با توزیع بار بین هسته‌های CPU

11 درمورد امکان برنامه‌نویسی زبان Rust برای Microcontroller ها و Robot ها توضیح دهید؟ **Robot** برای سیستم‌های تعبیه شده (Embedded Systems) از طریق پروژ‌هایی مانند embedded-hal پشتیبانی میکند که برای برنامه‌نویسی میکروکنترلرها و ربات‌ها استفاده میشود.

12- کاربرد و مفهوم Crate در زبان Rust را بیان کنید؟ **Crate** یک بسته یا

کتابخانه در Rust است که میتواند شامل کد منبع یا باینری باشد.

13 تفاوت مفهوم Iterator و Generator در زبان Rust را با ذکر مثال توضیح دهید؟

- Iterator یک API برای پیمایش روی عناصر است.

- Generator مشابه iterator است اما lazy evaluation دارد.

مثال: Iterator

```
let v = vec![1, 2, 3];
let iter = v.iter();
for val in iter {
    println!("{}", val);
}
```

به 14 Monomorphization در زبان Rust چیست؟
برای **Monomorphization** فرآیندی که طی آن کد جنریک (generic) کد مشخص (concrete) هر نوع تبدیل میشود

15. در مورد مفهوم Ownership در زبان Rust توضیح دهید؟
هر مقدار در Rust دارای مالک (owner) است و در هر لحظه فقط یک مالک وجود دارد.
16- قابلیت Borrowing در زبان Rust را با ذکر مثال توضیح دهید؟ امکان قرض گرفتن مقادیر بدون انتقال مالکیت .

```
fn main() {
    let s = String::from("Hello");
    let len = calculate_length(&s);
    println!("Length: {}", len);
}
fn calculate_length(s: &String) -> usize {
    s.len()
}
```

17. انواع Inference Type در زبان Rust را با ذکر مثال توضیح دهید؟

***استنباط نوع متغیر (Variable Inference)**

- * **استنباط نوع بازگشتی توابع (Function Return Type Inference)**
- * **استنباط نوع کلوزر (Closure Type Inference)**

* استنباط نوع آرایه (Array Inference)

* استنباط نوع وکتور (Vector Inference)

* استنباط نوع Iterator

* استنباط نوع در Match و غیره...

```
// 1. Variable Inference
let x = 42;          // i32
let y = 3.14;        // f64

// 2. Function Return Type
fn add(a: i32, b: i32) -> i32 {
    a + b
}

// 3. Closure Inference
let add = |x, y| x + y; // i32 inferred

// 4. Array Inference
let arr = [1, 2, 3];    // [i32; 3]

// 5. Vector Inference
let v = vec![1, 2, 3];  // Vec<i32>

// 6. Iterator Inference
let nums = vec![1, 2, 3];
let doubled: Vec<_> = nums.iter().map(|x| x * 2).collect();

// 7. Match Inference
let num = 5;
let result = match num {
    1 => "One",
    _ => "Other",
}; // &str
```

18. مفهوم Pointer Smart در زبان Rust را توضیح دهید؟

برای مدیریت حافظه استفاده میشوند Arc و Rc، اشاره گرهای هوشمند مانند

19. ساختار Trait چیست؟ با ذکر مثال توضیح دهید ؟

در زبان Rust، Trait مشابه interface در زبانهای دیگر است و برای تعریف رفتار (behavior) که توسط انواع مختلف پیادهسازی میشود، استفاده میشود

```
trait Greet {
    fn greet(&self);
}
struct Person;
impl Greet for Person {
    fn greet(&self) {
        println!("Hello!");
    }
}
```

20. Closure چیست را ذکر مثال توضیح دهید؟ تابع بدون نام که

میتواند متغیرهای محیط را بگیرد

```
let add = |x, y| x + y;
println!("{}", add(2, 3));
```

21. توضیح دهید زبان Rust با کدهای نا امن (Code unsafe) (چطور

رفتار می کند؟ با ذکر مثال توضیح دهید؟ اجازه میدهد تا از کد نا امن استفاده شود

```
let x: *const i32 = &10;
unsafe {
    println!("{}", *x);
}
```

22. درمورد کاربرد زبان برنامه نویسی Rust در حوزه Contracts

Smart توضیح دهید؟ Rust به دلیل امنیت بالا برای نوشتن قراردادهای

هوشمند در بلاکچین استفاده میشود.

23. درمورد کاربرد زبان برنامه نویسی Rust در حوزه Blockchain

توضیح دهید؟ Rust به دلیل کارایی بالا برای توسعه بلاکچینهایی مانند Solana استفاده شده است.

24. درمورد کاربرد زبان برنامه نویسی Rust در حوزه Intelligence

Artificial توضیح دهید؟ Rust در توسعه کتابخانههای AI و ML مانند torch-rs برای TensorFlow و PyTorch کاربرد دارد.

25. ابزارهای Test نویسی در زبان Rust را نام ببرید و تفاوت آنها را

بیان کنید؟ . cargo test . assert! . assert_eq!

. proptest برای تستهای مبتنی بر property

26-ابزارهای برنامه نویسی Web در زبان Rust :

در زبان Rust برای توسعه وب چندین ابزار وجود دارد که معمولاً از آنها در ساخت وبسایتها و API ها استفاده میشود:

1. **Rocket**: فریمورک توسعه وب است که بر سادگی و سرعت تمرکز دارد Rocket. به توسعه دهندگان این امکان را میدهد که API های پیچیده را به راحتی بسازند و از ویژگیهایی مانند درخواستهای ایمن و فرمتهای داده پشتیبانی میکند.

2. **Actix-web**: یک فریمورک وب سریع و مقاوم است که از مدلهای تطبیق کننده و ناهمگام استفاده میکند Actix. به دلیل سرعت بالا شناخته شده است و برای ساخت سیستمهای مقیاس پذیر مناسب است.

3. **Warp**: یک فریمورک وب دیگر است که بر سادگی، ایمنی و عملکرد تمرکز دارد. Warp از ویژگیهایی مانند فیلترهای ترکیبی برای مدیریت مسیرها و درخواستها پشتیبانی میکند.

4. **Tide**: فریمورک ساده و ناهمگام است که برای ایجاد API های وب ساخته شده است. تفاوتها: Tide بر روی Snafu و async/await تمرکز دارد.

. **Rocket**: بیشتر بر سادگی و استفاده از ویژگیهای Rust تمرکز دارد.

. **Actix-web**: بر سرعت و عملکرد بالا تاکید دارد و برای پروژههای مقیاسپذیر توصیه میشود.

. **Warp**: سادگی را برای توسعهدهندگان و ساختار ترکیبپذیر فیلترها ارائه میدهد.

. **Tide**: برای توسعهدهندگانی که به سادگی و بهره‌برداری از async/await علاقه‌مند هستند مناسب است.

27- مکانیزم کنترل خطا (Error Handling) در زبان Rust

Rust برای مدیریت خطاها از دو نوع اصلی استفاده میکند:

1. **Result<T, E>**: برای مدیریت خطاهای قابل پیشبینی یا خطاهایی که میتوان آنها را کنترل کرد، استفاده میشود.

2. **Option<T>**: برای زمانی که مقدار ممکن است موجود نباشد (مثل null در زبانهای دیگر) مثال:

```
fn divide(a: i32, b: i32) -> Result<i32, String> {
    if b == 0 {
        Err("Division by zero!".to_string())
    } else {
        Ok(a / b)
    }
}

fn main() {
    match divide(10, 0) {
        Ok(result) => println!("Result: {}", result),
        Err(e) => println!("Error: {}", e),
    }
}
```

در اینجا، اگر مقسوم علیه صفر باشد، یک خطا باز می‌گردد که به شکل `Err` نمایش داده می‌شود.

28 پشتیبانی میکند: - قابلیت‌های زبان Rust در Programming Functional

Rust از مفاهیم برنامه‌نویسی تابعی (Functional Programming)

1. توابع مرتبه اول: (**First-Class Functions**) در Rust میتوان توابع را به عنوان مقادیر دریافت یا بازگرداند.

2. توابع ناشناس (**Anonymous Functions**) یا **Closure**: این توابع میتوانند متغیرهایی از محیط خود را به طور خودکار ذخیره کنند.

مثال:

```
fn apply<F>(f: F) -> i32
where
    F: Fn(i32) -> i32,
{
    f(5)
}

fn main() {
    let closure = |x| x + 1;
    let result = apply(closure);
    println!("Result: {}", result);
}
```

در اینجا یک تابع ناشناس (closure) استفاده میشود که بر روی عدد ۵ اعمال میشود.

29 - برنامه‌نویسی شیگرا (Object-Oriented Programming) در زبان Rust

Rust به طور مستقیم از مفاهیم کلاسها و ارثبری مانند سایر زبانهای شیگرا پشتیبانی نمیکند، اما میتواند مفاهیم شیگرای مشابه را با استفاده از ویژگیهایی مانند ساختارها (Structs) و متحولات (Traits) پیاده‌سازی کند .

مثال:

```

trait Animal {
    fn sound(&self) -> String;
}

struct Dog;

impl Animal for Dog {
    fn sound(&self) -> String {
        "Woof".to_string()
    }
}

fn main() {
    let dog = Dog;
    println!("The dog says: {}", dog.sound());
}

```

در اینجا از **Traits** برای تعریف رفتارهایی مانند صدای یک حیوان استفاده شده است.

30 مفهوم (Meta-programming (Profiling) ، (Reflection) در زبان Rust

Rust به طور کامل از **Reflection** پشتیبانی نمی‌کند (یعنی نمیتوانید به طور داینامیک

اطلاعات نوع را در زمان اجرا بدست آورید). اما قابلیت‌هایی مانند **Profiling** و

Macros برای تولید کد و انجام برخی کارهای مشابه به متا-برنامهنویسی وجود دارند.

مثال (Macro) برای تولید کد:

```

macro_rules! say_hello {
    () => {
        println!("Hello, world!");
    };
}

fn main() {
    say_hello!();
}

```

در اینجا یک **Macro** برای چاپ پیام «سلام، دنیا!» تعریف شده است که به نوعی متا-

برنامهنویسی را شبیهسازی میکند.

31 - راهکار زبان Rust در مواجهه با مشکلات Safety Memory

چیست؟ با ذکر یک مثال ساده توضیح دهید؟

Rust با استفاده از سیستم مالکیت (Ownership) ، وامگیری (Borrowing) و قوانین طول عمر (Lifetimes) ، از مشکلات رایج ایمنی حافظه جلوگیری میکند. این مکانیزمها در زمان کامپایل بررسی میشوند و باعث میشوند خطاهای حافظه به حداقل برسند.

مشکل حافظه	راهکار Rust
Use after free	سیستم مالکیت اجازه دسترسی به دادهای آزاد شده را نمیدهد.
Double free	فقط یک مالک برای داده وجود دارد، از آزادسازی مجدد جلوگیری میشود.
Memory leaks	استفاده از Drop برای آزادسازی حافظه به صورت خودکار.
Null pointers	استفاده از نوع Option به جای اشارهگر Null.
Buffer overreads/overwrites	ایمنی آرایهها و جلوگیری از دسترسی خارج از محدوده.
Data races	برای اشتراک ایمن داده بین Arc و Mutex استفاده از threadها.

32 - عملکرد کتابخانه های زیر را با ذکر یک مثال ساده بیان کنید؟

1 - Serde - سریالسازی و دیسریالسازی

```

use serde::{Serialize, Deserialize};

#[derive(Serialize, Deserialize)]
struct Person {
    name: String,
    age: u8,
}

fn main() {
    let person = Person { name: "Ali".to_string(), age: 30 };
    let json = serde_json::to_string(&person).unwrap();
    println!("Serialized: {}", json);
}

```

2 – **Lazy_static** تعریف متغیرهای استاتیک که به صورت تنبل مقداردهی میشوند

```

use serde::{Serialize, Deserialize};

#[derive(Serialize, Deserialize)]
struct Person {
    name: String,
    age: u8,
}

fn main() {
    let person = Person { name: "Ali".to_string(), age: 30 };
    let json = serde_json::to_string(&person).unwrap();
    println!("Serialized: {}", json);
}

```

3 – **Thiserror** مدیریت خطاها

```

use serde::{Serialize, Deserialize};

#[derive(Serialize, Deserialize)]
struct Person {
    name: String,
    age: u8,
}

fn main() {
    let person = Person { name: "Ali".to_string(), age: 30 };
    let json = serde_json::to_string(&person).unwrap();
    println!("Serialized: {}", json);
}

```

4 – Itertools افزودن قابلیت‌های بیشتر به Iterator ها

```

use serde::{Serialize, Deserialize};

#[derive(Serialize, Deserialize)]
struct Person {
    name: String,
    age: u8,
}

fn main() {
    let person = Person { name: "Ali".to_string(), age: 30 };
    let json = serde_json::to_string(&person).unwrap();
    println!("Serialized: {}", json);
}

```

5 – Time مدیریت و محاسبات زمان

```

use serde::{Serialize, Deserialize};

#[derive(Serialize, Deserialize)]
struct Person {
    name: String,
    age: u8,
}

fn main() {
    let person = Person { name: "Ali".to_string(), age: 30 };
    let json = serde_json::to_string(&person).unwrap();
    println!("Serialized: {}", json);
}

```

6 – Url تجزیه و ترکیب URL ها

```

use url::Url;

fn main() {
    let parsed = Url::parse("https://www.rust-lang.org").unwrap();
    println!("Host: {:?}", parsed.host_str());
}

```

7 – Request ارسال درخواستهای HTTP

```

use request;

#[tokio::main]
async fn main() {
    let body = request::get("https://www.rust-lang.org")
        .await
        .unwrap()
        .text()
        .await
        .unwrap();
    println!("{}", body);
}

```

8 – Anyhow مدیریت خطاها به صورت ساده


```

        use request;

        #[tokio::main]
        async fn main() {
            let body = request::get("https://www.rust-lang.org")
                .await
                .unwrap()
                .text()
                .await
                .unwrap();
            println!("{}", body);
        }

```

9 – Digest محاسبه هش

```

        use sha2::{Sha256, Digest};

        fn main() {
            let mut hasher = Sha256::new();
            hasher.update(b"hello world");
            let result = hasher.finalize();
            println!("{:x}", result);
        }

```

10 – Signature مدیریت امضاهای دیجیتال

```

        use sha2::{Sha256, Digest};

        fn main() {
            let mut hasher = Sha256::new();
            hasher.update(b"hello world");
            let result = hasher.finalize();
            println!("{:x}", result);
        }

```

11 – Openssl رمزنگاری و SSL

```
use openssl::rsa::Rsa;

fn main() {
    let rsa = Rsa::generate(2048).unwrap();
    println!("RSA Key Generated");
}
```

12 – Chrono مدیریت تاریخ و زمان

```
use chrono::Local;

fn main() {
    let now = Local::now();
    println!("{}", now);
}
```

13 – Geo کار با موقعیتهای جغرافیایی

```
use chrono::Local;

fn main() {
    let now = Local::now();
    println!("{}", now);
}
```

14 – Hound کار با فایل‌های صوتی

```
use hound;

fn main() {
    WAV باز کردن و پردازش فایل
}
```

15 – Rustls پیاده‌سازی TLS امن

```
use hound;

fn main() {
    WAV باز کردن و پردازش فایل
}
```

```
use hound;

fn main() {
    WAV باز کردن و پردازش فایل
}
```

33 درمورد امکان فراخوانی کدهای دیگرزبان ها در زبان Rust توضیح دهید؟

Rust امکان فراخوانی کدهای نوشته شده به زبانهایی مانند C و C++ را از طریق FFI

(**رابط تابع خارجی**) فراهم میکند. این قابلیت به Rust اجازه میدهد تا از کتابخانههای موجود و کدهای سیستمی استفاده کند.

روش:

. استفاده از "C" extern برای تعریف توابع خارجی.

. پیاده سازی توابع با استفاده از unsafe برای اطمینان از ایمنی.

مثال فراخوانی کد C :

```
extern "C" {
    fn printf(format: *const i8, ...);
}

fn main() {
    let msg = b"Hello from C!\n\0";
    unsafe {
        printf(msg.as_ptr() as *const i8);
    }
}
```

34 -درمورد امکان فراخوانی کدهای زبان Rust در زبان هایی مانند Java ، #C ، C

و Python توضیح دهید؟ Rust امکان استفاده از کتابخانههای Rust در زبانهایی

مانند C ، Python ، Java و #C را با تولید کتابخانههای مشترک (**Shared Library**) یا

Rust DLL فراهم میکند.

روشها:

- ایجاد کتابخانه‌های `cdylib` یا `staticlib` برای تولید فایل `so` یا `dll`.
- استفاده از `[no_mangle]` برای جلوگیری از تغییر نام توابع توسط کامپایلر.
- ابزارهایی مانند `PyO3` برای `Python` و `jni` برای `Java`.

مثال ایجاد کتابخانه برای (Python)

```
use pyo3::prelude::*;

#[pyfunction]
fn add(a: i32, b: i32) -> i32 {
    a + b
}

#[pymodule]
fn mylib(py: Python, m: &PyModule) -> PyResult<()> {
    m.add_function(wrap_pyfunction!(add, m))?;
    Ok(())
}
```

35 در قالب یک مثال `Programming Asynchronous` در زبان `Rust` را توضیح دهید؟ `Rust` از برنامه‌نویسی ناهمگام (`Asynchronous`) با استفاده از `async` و `await` پشتیبانی میکند. این قابلیت اجازه میدهد تا بدون مسدود کردن `thread` ، عملیات ورودی/خروجی انجام شود.

مثال (برنامه ناهمگام):

```
use tokio::time::{sleep, Duration};

#[tokio::main]
async fn main() {
    let task1 = async_task(1);
    let task2 = async_task(2);

    tokio::join!(task1, task2);
}

async fn async_task(id: u8) {
    println!("Task {} started", id);
    sleep(Duration::from_secs(2)).await;
    println!("Task {} finished", id);
}
```