1)کتابخانه های Machine learningدر زبان Rustرا نام ببرید؟یک مثال ساده بنویسید؟

Ndarray.1 linfa. 2rustlearn.3 tch_rs.4 autograd.5

یک مثال ساده:

```
// Cargo.toml

[dependencies]

linfa = "0.7.0"

linfa-linear = "0.7.0"

use linfa::prelude::*;

use linfa_linear::LinearRegression;

use ndarray::Array2;

fn main() {

// داده‌های ورودی (X) و خروجی (y)

let x = Array2::from_shape_vec((4, 2), vec![1.0, 1.0, 1.0, 2.0, 2.0, 2.0, 2.0, 3.0]).unwrap();

let y = Array2::from_shape_vec((4, 1), vec![1.0, 2.0, 2.0, 3.0]).unwrap();

// ساخت مدل رگرسیون خطی

let model = LinearRegression::fit(&x, &y).unwrap();

// پیش‌بینی

let new_data = Array2::from_shape_vec((1, 2), vec![3.0, 3.0]).unwrap();

let prediction = model.predict(&new_data).unwrap();

println!("پیش‌بینی: {:?}", prediction);

}
```

2)برنامه نویسی Multi_threadingدرزبانRustرا با ذکر مثال ساده توضیح دهید؟

در این مثال، یک برنامه ساده ایجاد می‌کنیم که چند نخ را برای شمارش اعداد از 1 تا 10 ایجاد می‌کند. هر نخ یک عدد را به اشتراک می‌گذارد و آن را چاپ می‌کند.

```
rust

use std::thread;

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

fn main() {

// ایجاد یک Mutex برای محافظت از داده‌ها


let counter = Arc::new(Mutex::new(0));

let mut handles = vec![];

// ایجاد 10 نخ

for _ in 0..10 {

let counter = Arc::clone(&counter); // Clone Arc برای به اشتراک‌گذاری
```

```
let handle = thread::spawn(move || {
    let mut num = counter.lock().unwrap();  // قفل کردن Mutex
    *num += 1;  // افزایش شمارنده
    println!("شمارنده: {}", *num);
});
handles.push(handle);
}
// منتظر ماندن برای اتمام تمام نخ‌ها
for handle in handles {
handle.join().unwrap();
}
}
```

توضیحات:

1. **Arc و Mutex**:
   - Arc به ما این امکان را می‌دهد که یک اشاره‌گر به داده‌ها را بین نخ‌ها به اشتراک بگذاریم.
   - Mutex برای محافظت از داده‌ها در برابر دسترسی همزمان استفاده می‌شود.
2. **ایجاد نخ‌ها**:
   - با استفاده از thread::spawn، نخ‌های جدیدی ایجاد می‌کنیم. هر نخ یک کپی از Arc را دریافت می‌کند.
3. **قفل کردن Mutex**:
   - با استفاده از lock()، قفل را به دست می‌آوریم و سپس مقدار شمارنده را افزایش می‌دهیم.
4. **پیشرفت و چاپ شمارنده**:
   - هر نخ مقدار شمارنده را افزایش داده و آن را چاپ می‌کند.
5. **منتظر ماندن برای اتمام نخ‌ها**:
   - با استفاده از join()، منتظر می‌مانیم تا تمام نخ‌ها به پایان برسند.

این مثال ساده نشان می‌دهد که چگونه می‌توان از چندنخی در Rust استفاده کرد و داده‌ها را به‌طور ایمن بین نخ‌ها به اشتراک گذاشت.

3)برنامه نویسی Parallel Programmingدرزبان Rust را باذکر مثال ساده توضیح دهید؟

مثال ساده با استفاده از rayon

در این مثال، ما از rayon برای محاسبه مجموع مربع‌های اعداد از 1 تا 10 استفاده خواهیم کرد.

مراحل کار:

1. **نصب rayon**: ابتدا باید کتابخانه rayon را به پروژه خود اضافه کنید. در فایل Cargo.toml، به شکل زیر عمل کنید:

toml

[dependencies]

rayon = "1.7.1"

2. **نوشتن کد**: سپس می‌توانید از کد زیر برای محاسبه مجموع مربع‌ها استفاده کنید:

rust

use rayon::prelude::*;

fn main() {

// ایجاد یک آرایه از اعداد 1 تا 10

let numbers: Vec<i32> = (1..=10).collect();

// محاسبه مجموع مربع‌ها به صورت موازی

let sum_of_squares: i32 = numbers

.par_iter() *// تبدیل به Iterator موازی*

.map(|&x| x * x) *// محاسبه مربع هر عدد*

.sum(); *// جمع کردن نتایج*

println!("مجموع مربع‌ها: {}", sum_of_squares);

}

4) Lazy Loadingچیست؟باذکر مثال در زبان Rustتوضیح دهید؟

Lazy Loading(بارگذاری تنبل) یک الگوی برنامه‌نویسی است که در آن داده‌ها یا منابع تنها زمانی بارگذاری می‌شوند که به آن‌ها نیاز است، به جای اینکه در زمان شروع برنامه یا بارگذاری اولیه بارگذاری شوند. این الگو می‌تواند به بهینه‌سازی مصرف حافظه و زمان بارگذاری کمک کند، زیرا منابع غیرضروری بارگذاری نمی‌شوند و فقط در صورت نیاز بارگذاری می‌شوند.

در زبان Rust، می‌توان از کتابخانه lazy_static برای پیاده‌سازی بارگذاری تنبل استفاده کرد. این کتابخانه به شما این امکان را می‌دهد که متغیرهای استاتیک را به‌طور تنبل بارگذاری کنید.

مثال ساده با استفاده از lazy_static

در این مثال، ما از lazy_static برای بارگذاری یک متغیر استاتیک به‌صورت تنبل استفاده خواهیم کرد. فرض کنید می‌خواهیم یک رشته بزرگ را تنها زمانی بارگذاری کنیم که به آن نیاز داریم.

مراحل کار:

1. **نصب lazy_static**: ابتدا باید کتابخانه lazy_static را به پروژه خود اضافه کنید. در فایل Cargo.toml، به شکل زیر عمل کنید:

toml

[dependencies]

lazy_static = "1.4.0"

2. **نوشتن کد**: سپس می‌توانید از کد زیر برای بارگذاری تنبل یک متغیر استاتیک استفاده کنید:

```rust
#[macro_use]
extern crate lazy_static;

use std::sync::Mutex;
lazy_static! {
static ref LARGE_DATA: Mutex<String> = Mutex::new(load_large_data());
}
// تابعی برای بارگذاری داده‌های بزرگ

fn load_large_data() -> String {
    println!("بارگذاری داده‌های بزرگ...");

    // فرض کنید اینجا داده‌های بزرگ بارگذاری می‌شوند
    "این یک رشته بزرگ است".to_string()
}
fn main() {
    // در اینجا، داده‌ها هنوز بارگذاری نشده‌اند
    println!("قبل از دسترسی به داده‌های بزرگ");
    // دسترسی به داده‌های بزرگ
    let data = LARGE_DATA.lock().unwrap();
    println!("داده‌های بزرگ: {}", *data);
    // در اینجا، داده‌ها بارگذاری شده‌اند
    println!("بعد از دسترسی به داده‌های بزرگ");
}
```

5)ساختمان داده Binary Search Treeرادر زبان Rustپیاده سازی نمایید؟

```rust
#[derive(Debug)]
struct Node {
    value: i32,
    left: Option<Box<Node>>,
    right: Option<Box<Node>>,
}
#[derive(Debug)]
```

```
struct BinarySearchTree {
root: Option<Box<Node>>,
}
impl BinarySearchTree {
// ایجاد یک درخت جدید
fn new() -> Self {
BinarySearchTree { root: None }
}
// درج یک مقدار در درخت
fn insert(&mut self, value: i32) {
self.root = Self::insert_node(self.root.take(), value);
}
fn insert_node(node: Option<Box<Node>>, value: i32) -> Option<Box<Node>> {
match node {
Some(mut n) => {
if value < n.value {
n.left = Self::insert_node(n.left, value);
} else if value > n.value {
n.right = Self::insert_node(n.right, value);
}
Some(n)
}
None => Some(Box::new(Node {
value,
left: None,
right: None,
})),
}
}
// جستجو برای یک مقدار در درخت
fn search(&self, value: i32) -> bool {
Self::search_node(&self.root, value)
}
fn search_node(node: &Option<Box<Node>>, value: i32) -> bool {
match node {
```

```
                Some(n) => {
                    if value == n.value {
                        true
                    } else if value < n.value {
                        Self::search_node(&n.left, value)
                    } else {
                        Self::search_node(&n.right, value)
                    }
                }
                None => false,
            }
        }
        // حذف یک مقدار از درخت
        fn delete(&mut self, value: i32) {
            self.root = Self::delete_node(self.root.take(), value);
        }
    fn delete_node(node: Option<Box<Node>>, value: i32) -> Option<Box<Node>> {
        if let Some(mut n) = node {
            if value < n.value {
                n.left = Self::delete_node(n.left, value);
            } else if value > n.value {
                n.right = Self::delete_node(n.right, value);
            } else {
                // نود مورد نظر پیدا شده است
                if n.left.is_none() {
                    return n.right;
                } else if n.right.is_none() {
                    return n.left;
                }
                // نود با دو فرزند
                let min_larger_node = Self::find_min(&n.right);
                n.value = min_larger_node.value;
                n.right = Self::delete_node(n.right, min_larger_node.value);
            }
            Some(n)
```

```
} else {
None
}
}
fn find_min(node: &Option<Box<Node>>) -> &Node {
match node {
Some(n) => {
if n.left.is_none() {
n
} else {
Self::find_min(&n.left)
}
}
None => panic!("find_min called on an empty node"),
}
}
// پیمایش درخت به صورت inorder
fn inorder_traversal(&self) {
Self::inorder(&self.root);
}

fn inorder(node: &Option<Box<Node>>) {
if let Some(n) = node {
Self::inorder(&n.left);
println!("{}", n.value);
Self::inorder(&n.right);
}
}
}
fn main() {
let mut bst = BinarySearchTree::new();
// درج مقادیر در درخت
bst.insert(5);
bst.insert(3);
bst.insert(7);
```

```
bst.insert(2);
bst.insert(4);
bst.insert(6);
bst.insert(8);
// پیمایش درخت به صورت inorder
println!("پیمایش inorder:");
bst.inorder_traversal();
// جستجو برای مقادیر
let search_value = 4;
println!("آیا {} در درخت وجود دارد؟ {}", search_value, bst.search(search_value));
// حذف یک مقدار از درخت
bst.delete(3);
println!("پیمایش inorder بعد از حذف 3:");
bst.inorder_traversal();
}
```

6)ساختمان داده AvLTree را در زبان Rust پیاده سازی کنید؟

درخت AVL یک نوع درخت جستجوی دودویی است که در آن ارتفاع زیر درخت‌های چپ و راست هر گره حداکثر یک واحد اختلاف دارد. این ویژگی باعث می‌شود که درخت همیشه متوازن باقی بماند و عملیات جستجو، درج و حذف در زمان لگاریتمی انجام شوند.

در زیر یک پیاده‌سازی ساده از درخت AVL در زبان Rust ارائه می‌شود:

```
rust
#[derive(Debug)]
struct Node<T> {
value: T,
height: i32,
left: Option<Box<Node<T>>>,
right: Option<Box<Node<T>>>,
}
impl<T: Ord> Node<T> {
fn new(value: T) -> Self {
Node {
value,
height: 1,
left: None,
```

```
                right: None,
            }
        }
    }
    fn height(node: &Option<Box<Node<T>>>) -> i32 {
        match node {
            Some(n) => n.height,
            None => 0,
        }
    }
    fn update_height(node: &mut Box<Node<T>>) {
        node.height = 1 + i32::max(Node::height(&node.left), Node::height(&node.right));
    }
    fn balance_factor(node: &Option<Box<Node<T>>>) -> i32 {
        match node {
            Some(n) => Node::height(&n.left) - Node::height(&n.right),
            None => 0,
        }
    }
    fn rotate_right(y: &mut Box<Node<T>>) -> Box<Node<T>> {
        let mut x = y.left.take().unwrap();
        y.left = x.right.take();
        Node::update_height(y);
        x.right = Some(y);
        Node::update_height(&mut x);
        x
    }
    fn rotate_left(x: &mut Box<Node<T>>) -> Box<Node<T>> {
        let mut y = x.right.take().unwrap();
        x.right = y.left.take();
        Node::update_height(x);
        y.left = Some(x);
        Node::update_height(&mut y);
        y
```

```
}
fn balance(node: &mut Box<Node<T>>) {
Node::update_height(node);
let balance = Node::balance_factor(&Some(node.clone()));
if balance > 1 {
if Node::balance_factor(&node.left) < 0 {
node.left = Some(Node::rotate_left(node.left.as_mut().unwrap()));
}
*node = Node::rotate_right(node);
} else if balance < -1 {
if Node::balance_factor(&node.right) > 0 {
node.right = Some(Node::rotate_right(node.right.as_mut().unwrap()));
}
*node = Node::rotate_left(node);
}
}
}
struct AVLTree<T> {
root: Option<Box<Node<T>>>,
}
impl<T: Ord> AVLTree<T> {
fn new() -> Self {
AVLTree { root: None }
}
fn insert(&mut self, value: T) {
self.root = Self::insert_node(self.root.take(), value);
}
fn insert_node(node: Option<Box<Node<T>>>, value: T) -> Option<Box<Node<T>>> {
let mut node = match node {
Some(n) => n,
None => return Some(Box::new(Node::new(value))),
};
if value < node.value {
```

```
            node.left = Self::insert_node(node.left, value);
        } else if value > node.value {
            node.right = Self::insert_node(node.right, value);
        } else {
            return Some(node); // Duplicate values are not allowed
        }
        Node::balance(&mut node);
        Some(node)
    }
    fn inorder(&self) {
        Self::inorder_helper(&self.root);
    }
    fn inorder_helper(node: &Option<Box<Node<T>>>) {
        if let Some(n) = node {
            Self::inorder_helper(&n.left);
            println!("{}", n.value);
            Self::inorder_helper(&n.right);
        }
    }
}
fn main() {
    let mut avl_tree = AVLTree::new();
    avl_tree.insert(10);
    avl_tree.insert(20);
    avl_tree.insert(30);
    avl_tree.insert(40);
    avl_tree.insert(50);
    avl_tree.insert(25);
    println!("Inorder traversal of the AVL tree:");
    avl_tree.inorder();
}
```

7) ساختمان داده Maxheap_treeرا در زبان Rustپیاده سازی کنید؟ پیاده‌سازی یک ماکس‌هیپ (Max Heap) در زبان Rust می‌تواند به شما کمک کند تا با مفهوم هیپ و نحوه مدیریت داده‌ها در ساختارهای درختی آشنا شوید. ماکس‌هیپ به شما این امکان را می‌دهد که بزرگ‌ترین عنصر را به سرعت پیدا کنید و آن را استخراج کنید. در زیر یک پیاده‌سازی ساده از ماکس‌هیپ در زبان Rust ارائه می‌شود:

```
#[derive(Debug)]
struct MaxHeap {
data: Vec<i32>,
}
impl MaxHeap {
// ایجاد یک ماکس‌هیپ جدید
fn new() -> Self {
MaxHeap { data: Vec::new() }
}
// پیدا کردن ایندکس والد
fn parent(index: usize) -> usize {
(index - 1) / 2
}
// پیدا کردن ایندکس فرزند چپ
fn left_child(index: usize) -> usize {
2 * index + 1
}
// پیدا کردن ایندکس فرزند راست
fn right_child(index: usize) -> usize {
2 * index + 2
}
// درج یک عنصر جدید در ماکس‌هیپ
fn insert(&mut self, value: i32) {
self.data.push(value);
self.bubble_up(self.data.len() - 1);
}
// بالا آوردن عنصر به سمت ریشه
fn bubble_up(&mut self, index: usize) {
let mut current_index = index;
while current_index > 0 {
```

```
        let parent_index = MaxHeap::parent(current_index);
        if self.data[current_index] > self.data[parent_index] {
            self.data.swap(current_index, parent_index);
            current_index = parent_index;
        } else {
            break;
        }
    }
    // استخراج بزرگ‌ترین عنصر (ریشه)
    fn extract_max(&mut self) -> Option<i32> {
        if self.data.is_empty() {
            return None;
        }
        let max = self.data[0];
        let last = self.data.pop().unwrap();  // حذف آخرین عنصر
        if !self.data.is_empty() {
            self.data[0] = last;  // قرار دادن آخرین عنصر در ریشه
            self.bubble_down(0);  // حفظ ویژگی ماکس‌هیپ
        }
        Some(max)
    }
    // پایین آوردن عنصر به سمت پایین
    fn bubble_down(&mut self, index: usize) {
        let mut current_index = index;
        let length = self.data.len();
        loop {
            let left_index = MaxHeap::left_child(current_index);
            let right_index = MaxHeap::right_child(current_index);
            let mut largest_index = current_index;
            if left_index < length && self.data[left_index] > self.data[largest_index] {
                largest_index = left_index;
            }
            if right_index < length && self.data[right_index] > self.data[largest_index] {
```

```
                largest_index = right_index;
            }
            if largest_index == current_index {
                break;
            }
            self.data.swap(current_index, largest_index);
            current_index = largest_index;
        }
    }
    // مشاهده بزرگ‌ترین عنصر بدون حذف آن
    fn peek(&self) -> Option<&i32> {
        self.data.get(0)
    }
    // بررسی خالی بودن هیپ
    fn is_empty(&self) -> bool {
        self.data.is_empty()
    }
}
fn main() {
    let mut max_heap = MaxHeap::new();
    max_heap.insert(10);
    max_heap.insert(20);
    max_heap.insert(5);
    max_heap.insert(30);
    max_heap.insert(15);
    println!("Max heap: {:?}", max_heap);
    while !max_heap.is_empty() {
        println!("Extracted max: {:?}", max_heap.extract_max());
    }
}
```

8) یک سرویسRESTful APIجهت پردازش درخواست های JSONبنویسید؟ برای ایجاد یک سرویس RESTful API در زبان Rust که قادر به پردازش درخواست‌های JSON باشد، می‌توانیم از کتابخانه‌هایی مانند actix-webیا warpاستفاده کنیم. در اینجا، یک نمونه ساده با استفاده از actix-webارائه می‌شود. این API به ما اجازه می‌دهد تا داده‌های JSON را دریافت کنیم و پاسخ‌های JSON را ارسال کنیم.

1. **یجاد یک پروژه جدید** :ابتدا یک پروژه جدید Rust ایجاد کنید:

2. **اضافه کردن وابستگی‌ها** :در فایل Cargo.toml، وابستگی‌های actix-webو serdeرا اضافه کنید:

3. **نوشتن کد API**:در فایل src/main.rs، کد زیر را قرار دهید

9)یک سرویس ساده جهت پردازش درخواست های مبتنی بر پروتکلgRPCبنویسید؟ برای ایجاد یک سرویس ساده gRPC در زبانRust ، می‌توانیم از کتابخانه‌های tonicو prostاستفاده کنیم .tonic یک کتابخانه gRPC برای Rust است و prostبرای سریال‌سازی و دی‌سریال‌سازی پیام‌های پروتکل استفاده می‌شود.

1.ایجاد یک پروژه جدید :ابتدا یک پروژه جدید Rust ایجاد کنید:

```
cargo new rust_grpc_service

cd rust_grpc_service
```

2. اضافه کردن وابستگی‌ها :در فایل Cargo.toml، وابستگی‌های tonicو prostرا اضافه کنید:

```
[dependencies]

"tonic = "0.7

tokio = { version = "1", features = ["full"] }

"prost = "0.7

[build-dependencies]

"tonic-build = "0.7
```

3. **تعریف پروتکل gRPC**:یک فایل جدید به نام proto/service.protoایجاد کنید و محتوای زیر را در آن قرار دهید:

4. **ایجاد کد Rust از فایل proto**:در فایل build.rsدر ریشه پروژه، کد زیر را قرار دهید تا کد Rust از فایل proto تولید شود:

```
} ()fn main

    ;()tonic_build::compile_protos("proto/service.proto").unwrap

{
```

5. **نوشتن کد سرور gRPC**:در فایل src/main.rs، کد زیر را قرار دهید

6. **اجرای برنامه:**برای اجرای برنامه، از دستور زیر استفاده کنید:

```
cargo run
```

10) یک سرویس ساده جهت پردازش درخواست های مبتنی برWeb Assemblyبنویسید؟ ایجاد یک سرویس ساده مبتنی بر WebAssembly (Wasm)در Rust نیازمند استفاده از چند کتابخانه و ابزار است. در اینجا، ما یک پروژه ساده را با استفاده از wasm-bindgenو wasm-packایجاد خواهیم کرد. این پروژه یک ماژول WebAssembly را ایجاد می‌کند که می‌تواند در مرورگر اجرا شود.

1. نصب ابزارهای مورد نیاز :ابتدا مطمئن شوید که Rust و wasm-packرا نصب کرده‌اید. اگر wasm-packرا ندارید، می‌توانید با استفاده از دستور زیر آن را نصب کنید:

```
cargo install wasm-pack
```

2. **ایجاد یک پروژه جدید** :یک پروژه جدید Rust ایجاد کنید

```
cargo new wasm_example --lib
```

cd wasm_example

3. **تنظیمات Cargo.toml**: در فایل Cargo.toml، وابستگی‌های wasm-bindgen را اضافه کنید:

[package]

"name = "wasm_example

"version = "0.1.0

"edition = "2021

[lib]

crate-type = ["cdylib"]

[dependencies]

"wasm-bindgen = "0.2

4. **نوشتن کد WebAssembly**: در فایل src/lib.rs، کد زیر را قرار دهید

;*::use wasm_bindgen::prelude

#[wasm_bindgen]

} pub fn greet(name: &str) -> String

format!("Hello, {}!", name)

{

5. ساخت پروژه :برای ساخت پروژه و تولید فایلWebAssembly ، از دستور زیر استفاده کنید:

wasm-pack build --target web

6. ایجاد یک پروژه HTML برای تست :یک پوشه جدید به نام www ایجاد کنید و در آن یک فایل HTML به نام index.html ایجاد کنید

mkdir www

touch www/index.html

11)Sochet Programming در زبان Rustرا بهمراه یک مثال بیان کنید؟

برنامه‌نویسی سوکت (Socket Programming) در زبان Rust به ما این امکان را می‌دهد که ارتباطات شبکه‌ای را پیاده‌سازی کنیم .با استفاده از کتابخانه استاندارد std::netامکاناتی برای ایجاد و مدیریت سوکت‌ها فراهم می‌کند. در اینجا یک مثال ساده از یک برنامه سرور و کلاینت TCP در Rust ارائه می‌شود.

server.rs //

;use std::net::{TcpListener, TcpStream}

;use std::io::{Read, Write}

;use std::thread

} fn handle_client(mut stream: TcpStream)

let mut buffer = [0; 1024]; // بافر برای دریافت داده‌ها

} match stream.read(&mut buffer)

```
} <= Ok(size)
;let received = String::from_utf8_lossy(&buffer[0..size])
;println!("Received: {}", received)
;()stream.write(b"Message received").unwrap // ارسال پاسخ به کلاینت
{
} <= Err(e)
;eprintln!("Failed to read from client: {}", e)
{
{
{


} ()fn main
;()let listener = TcpListener::bind("127.0.0.1:7878").unwrap // ایجاد Listener در پورت 7878
;println!("Server listening on port 7878")


} ()for stream in listener.incoming
} match stream
} <= Ok(stream)
;{ || thread::spawn(move // هر اتصال را در یک رشته جدید پردازش می‌کند
;handle_client(stream)
;({
{
} <= Err(e)
;eprintln!("Failed to accept connection: {}", e)
{
{
{
{
```

12) برنامه ای به زبان Rust بنویسید که عملیات CRUDرا بر روی یک پایگاه داده انجام دهد؟ برای ایجاد یک برنامه CRUD (Create, Read, Update, Delete)در زبانRust ، می‌توانیم از کتابخانه rusqliteبرای کار با پایگاه داده SQLite استفاده کنیم. در اینجا یک مثال ساده از یک برنامه CRUDبرای مدیریت یک لیست از کاربران (با نام و سن) ارائه می‌شود.

1. **نصب وابستگی‌ها** :ابتدا باید کتابخانه rusqliteرا به پروژه خود اضافه کنید. برای این کار، فایل Cargo.tomlرا باز کرده و خط زیر را به بخش [dependencies]اضافه کنید:

```
[dependencies]
```

"0.26 = rusqlite" # یا آخرین نسخه موجود

2. **نوشتن کد CRUD**: در فایل main.rs،

توضیحات کد

1. **ساختار User**: یک ساختار User برای نگهداری اطلاعات کاربر (شامل id, name, و age) تعریف شده است.
2. **عملیات CRUD**:
   - **create_user**: یک کاربر جدید به جدول users اضافه می‌کند.
   - **read_users**: تمام کاربران موجود در جدول را می‌خواند و به صورت یک وکتور از User برمی‌گرداند.
   - **update_user**: اطلاعات یک کاربر را با استفاده از id به‌روزرسانی می‌کند.
   - **delete_user**: یک کاربر را با استفاده از id حذف می‌کند.

تابع main: در این تابع، ابتدا یک اتصال به پایگاه داده برقرار می‌شود و جدول users ایجاد می‌شود (اگر وجود نداشته باشد). سپس عملیات CRUD به ترتیب انجام می‌شود و نتایج در کنسول چاپ می‌شود

اجرای برنامه

برای اجرای برنامه، ابتدا اطمینان حاصل کنید که Rust و Cargo را نصب کرده‌اید. سپس در دایرکتوری پروژه، از دستورات زیر استفاده کنید:

cargo build

cargo run

13) با استفاده از یکORMدر زبان Rustبرنامه ای بنویسید که عملیات CRUDرا بر روی یک پایگاه داده انجام دهد؟ برای ایجاد یک برنامه CRUD در زبان Rust با استفاده از یک(Object-Relational Mapping) ORM ، می‌توانیم از کتابخانه Dieselاستفاده کنیم Diesel .یکی از محبوب‌ترین ORMها در Rust است و به شما این امکان را می‌دهد که با پایگاه‌های داده مختلف به راحتی کار کنید.

1. **نصب وابستگی‌ها** :ابتدا باید کتابخانه‌های Dieselو dotenvرا به پروژه خود اضافه کنید. برای این کار، فایل Cargo.tomlرا باز کرده و خطوط زیر را به بخش [dependencies]اضافه کنید:

[dependencies]

diesel = { version = "2.0", features = ["sqlite"] }

"dotenv = "0.15

2. **تنظیم پایگاه داده** :برای استفاده از Diesel، ابتدا باید پایگاه داده SQLite را تنظیم کنیم. برای این کار، از ابزار diesel_cliاستفاده خواهیم کرد. می‌توانید آن را با استفاده از دستور زیر نصب کنید

cargo install diesel_cli --no-default-features --features sqlite

3. **ایجاد پایگاه داده و جدول** :در دایرکتوری پروژه، یک فایل .envایجاد کنید و در آن مسیر پایگاه داده را مشخص کنید

DATABASE_URL=users.db

سپس، با استفاده از Dieselجدول usersرا ایجاد کنیدdiesel setup

و یک فایل مهاجرت برای ایجاد جدول usersایجاد کنیدdiesel migration generate create_users

سپس، در فایل مهاجرت (در دایرکتوری migrations)، کد زیر را اضافه کنید:

CREATE TABLE users (

id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT NOT NULL,

```
age INTEGER NOT NULL
);
```

و مهاجرت را اجرا کنیدdiesel migration run :

نوشتن کد :**CRUD**حالا می‌توانیم کد CRUD را در فایل main.rsبنویسیم

14) مفهوم Regular Expressionچیست؟در زبان Rustبا بیان یک مثال توضیح دهید؟ عبارات باقاعده Regular Expressions) یا به اختصار (Regex الگوهایی هستند که برای جستجو و تطبیق الگوهای خاصی در متن استفاده می‌شوند. این الگوها می‌توانند شامل حروف، اعداد و نمادهای خاص باشند و به شما این امکان را می‌دهند که به سرعت و به طور مؤثر متن را تجزیه و تحلیل کنید، داده‌ها را استخراج کنید یا الگوهای خاصی را در متن جستجو کنید.

مثال:

```
use regex::Regex;

fn main() {
    // تعریف یک الگوی عبارات باقاعده
    let re = Regex::new(r"(\w+)@(\w+)\.(\w+)").unwrap();

    // رشته‌ای که می‌خواهیم بررسی کنیم
    let text = "Contact us at support@example.com or sales@company.org.";

    // جستجو در رشته
    for cap in re.captures_iter(text) {
        println!("Found email: {}", &cap[0]);  // کل الگو
        println!("Username: {}", &cap[1]);  // بخش قبل از '@'
        println!("Domain: {}", &cap[2]);  // بخش بین '@' و '.'
        println!("TLD: {}", &cap[3]);  // بخش بعد از '.'
    }
}
```

15) عملکرد کتابخانه SysInfoدرزبانRustچیست؟با ذکر مثال ساده توضیح دهید؟ کتابخانه sysinfoدر زبان Rust به شما این امکان را می‌دهد که اطلاعات سیستم را به راحتی به دست آورید. این اطلاعات شامل جزئیاتی درباره پردازنده‌ها، حافظه، دیسک‌ها، و فرآیندهای در حال اجرا در سیستم شما است. این کتابخانه می‌تواند برای نظارت بر عملکرد سیستم، تجزیه و تحلیل منابع، و توسعه ابزارهای مدیریتی مفید باشد.

مثال:

```
use sysinfo::{ProcessorExt, System, SystemExt};

fn main() {
    // ایجاد یک شیء از نوع System
    let mut system = System::new_all();

    // بارگذاری اطلاعات سیستم
```

```
system.refresh_all();

// دریافت و نمایش اطلاعات پردازنده‌ها
println!("Processors:");
for processor in system.get_processors() {
println!("{}: {}%", processor.get_name(), processor.get_cpu_usage());
}

// دریافت و نمایش اطلاعات حافظه
println!("\nMemory:");
println!("Total memory: {} KB", system.get_total_memory());
println!("Used memory: {} KB", system.get_used_memory());
println!("Free memory: {} KB", system.get_free_memory());
}
```

16) عملکرد کتابخانه windwosو native-windowsدر زبان Rustچیست؟با ذکر یک مثال ساده توضیح دهید؟ کتابخانه‌های native-windows و windowsدر زبان Rust به شما این امکان را می‌دهند که باAPI های ویندوز به راحتی تعامل داشته باشید. این کتابخانه‌ها به شما اجازه می‌دهند که برنامه‌های ویندوزی را با استفاده از امکانات و ویژگی‌های خاص سیستم عامل ویندوز توسعه دهید.

مثال:
```
use windows::Win32::UI::WindowsAndMessaging::{MessageBoxW, MB_OK};

fn main() {
// نمایش یک پیام با استفاده از MessageBox
unsafe {
MessageBoxW(
None,
"سلام، این یک پیام است!".into(),
"پیام",
MB_OK,
);
}
}
```

17) برنامه ی برای انجام یک پردازش ساده بر روی یک Imageبنویسید؟

```
use image::{DynamicImage, GenericImageView, GrayImage, ImageBuffer, Luma};
```

```
} ()fn main

// بارگذاری تصویر

;let img = image::open("input.png").expect("Failed to open image")


// نمایش ابعاد تصویر

;()let (width, height) = img.dimensions

;println!("Image dimensions: {}x{}", width, height)


// تبدیل تصویر به مقیاس خاکستری

;()let gray_img: GrayImage = img.to_luma8


// ذخیره تصویر خاکستری

;gray_img.save("output.png").expect("Failed to save image")

;println!("Gray image saved as output.png")

{
```

18) برنامه ای برای انحام یک پردازش ساده بر روی یکAudioبنویسید؟ برای انجام پردازش‌های ساده بر روی فایل‌های صوتی در زبانRust ، می‌توانیم از کتابخانه rodioبرای پخش صدا و از houndبرای خواندن و نوشتن فایل‌های WAV استفاده کنیم. در اینجا یک مثال ساده از نحوه بارگذاری یک فایل صوتیWAV ، تغییر حجم آن و ذخیره آن به عنوان یک فایل جدید آورده شده است.

مثال:

```
use hound::{WavReader, WavWriter, SampleFormat};

use std::fs::File;


fn main() {

    // بارگذاری فایل صوتی

    let input_file = "input.wav";

    let output_file = "output.wav";


    let reader = WavReader::open(input_file).expect("Failed to open WAV file");

    let spec = reader.spec();


    // ایجاد نویسنده برای فایل جدید

    let mut writer = WavWriter::create(output_file, spec).expect("Failed to create WAV file");


    // تغییر حجم صدا
```

```
    let volume_factor = 0.5; // تغییر حجم به نصف

    for sample in reader.samples::<i16>() {
        let sample = sample.expect("Failed to read sample");
        let modified_sample = (sample as f32 * volume_factor) as i16; // تغییر حجم
        writer.write_sample(modified_sample).expect("Failed to write sample");
    }

    println!("Audio processing completed. Output saved as {}", output_file);
}
```

19) برنامه ای برای انجام یک پردازش ساده بر روی یکvideo بنویسید؟ برای انجام پردازش‌های ساده بر روی ویدئوها در زبانRust ، می‌توانیم از کتابخانه ffmpegیا gstreamerاستفاده کنیم. در اینجا، از ffmpegبه عنوان یک کتابخانه محبوب برای پردازش ویدئو استفاده خواهیم کرد. با استفاده از این کتابخانه می‌توانیم ویدئوها را بخوانیم، ویرایش کنیم و ذخیره کنیم.

```
use ffmpeg_next::{format, media, packet, software, codec, Rational};

fn main() {
    // راه‌اندازی FFmpeg
    ffmpeg_next::init().unwrap();

    // بارگذاری ویدئو
    let input_file = "input.mp4";
    let output_file = "output.mp4";

    // باز کردن فایل ورودی
    let mut input = format::input(&input_file).unwrap();

    // ایجاد فایل خروجی
    let mut output = format::output(&output_file).unwrap();

    // ایجاد استریم جدید برای ویدئو
    let mut stream = output.add_stream(media::Type::Video).unwrap();

    // تنظیم پارامترهای ویدئو
    stream.set_codec(codec::Id::H264);
```

```
stream.set_bit_rate(400000); // تنظیم بیت ریت
stream.set_width(640); // عرض جدید
stream.set_height(480); // ارتفاع جدید
stream.set_frame_rate(Rational::new(30, 1)); // فریم ریت جدید

// باز کردن استریم
output.open().unwrap();

// پردازش ویدئو
for (stream_index, packet) in input.packets() {
    if stream_index == 0 { // فقط پردازش استریم ویدئو
        let mut new_packet = packet.clone();
        // تغییر اندازه یا پردازش دیگر می‌تواند اینجا انجام شود
        output.write_packet(&new_packet).unwrap();
    }
}

// بستن فایل خروجی
output.finalize().unwrap();
println!("Video processing completed. Output saved as {}", output_file);
}
```

20) برنامه ای برای دانلود یک فایل ازinternetبنویسید؟

برای دانلود یک فایل از اینترنت در زبانRust ، می‌توانیم از کتابخانه reqwestاستفاده کنیم. این کتابخانه به شما این امکان را می‌دهد که به راحتی درخواست‌های HTTP را ارسال کنید و پاسخ‌ها را مدیریت کنید. در اینجا یک مثال ساده از نحوه دانلود یک فایل از اینترنت آورده شده است.

```
use reqwest::Error;
use std::fs::File;
use std::io::copy;

#[tokio::main]
async fn main() -> Result<(), Error> {
    let url = "https://example.com/file.zip"; // URL فایل مورد نظر
    let response = reqwest::get(url).await?;

    // بررسی وضعیت پاسخ
```

```
} ()if response.status().is_success
    (")let mut dest = File::create("downloaded_file.zip"?; // نام فایل ذخیره شده
    ;?let content = response.bytes().await
    ;?copy(&mut content.as_ref(), &mut dest)
    ;println!("File downloaded successfully!")
} else {
    ;println!("Failed to download file: {}", response.status())
{


(())Ok
{
```

21) برنامه ای برای خواندن و نوشتن یک فایل ازCSVساده بنویسید؟رای خواندن و نوشتن یک فایل CSV در زبان برنامه‌نویسی پایتون، می‌توانید از ماژول csvاستفاده کنید. در زیر یک برنامه ساده برای خواندن و نوشتن فایل CSV ارائه می‌شود.

الف) نوشتن داده‌ها به فایل CSV

```
import csv

# داده‌هایی که می‌خواهیم به فایل CSV بنویسیم
] = data
    ['نام', 'سن', 'شغل'],
    ['علی', 30, 'برنامه‌نویس'],
    ['مریم', 25, 'طراح'],
    ['حسین', 35, 'مدیر']
[

# نوشتن داده‌ها به فایل CSV
:with open('data.csv', mode='w', newline='', encoding='utf-8') as file
    writer = csv.writer(file)
    writer.writerows(data)

print("داده‌ها با موفقیت به فایل CSV نوشته شدند.")
```

ب) خواندن داده‌ها از فایل CSV

```
import csv
```

```
# خواندن داده‌ها از فایل CSV
:with open('data.csv', mode='r', encoding='utf-8') as file
    reader = csv.reader(file)
    :for row in reader
        print(row)
```

22) برنامه ای برای خواندن و نوشتن یک فایل Excelساده بنویسید؟ برای خواندن و نوشتن فایل‌های Excel در زبان برنامه‌نویسی پایتون، می‌توانید از کتابخانه pandasو openpyxlاستفاده کنید. این کتابخانه‌ها امکانات خوبی برای کار با فایل‌های Excel فراهم می‌کنند

الف) نوشتن داده‌ها به فایل Excel

```
import pandas as pd

# داده‌هایی که می‌خواهیم به فایل Excel بنویسیم
} = data
    'نام': ['علی', 'مریم', 'حسین'],
    'سن': [30, 25, 35],
    'شغل': ['برنامه‌نویس', 'طراح', 'مدیر']
{

# ایجاد DataFrame از داده‌ها
df = pd.DataFrame(data)

# نوشتن DataFrame به فایل Excel
df.to_excel('data.xlsx', index=False)

print("داده‌ها با موفقیت به فایل Excel نوشته شدند.")
```

ب) خواندن داده‌ها از فایل Excel

```
import pandas as pd

# خواندن داده‌ها از فایل Excel
df = pd.read_excel('data.xlsx')

# نمایش داده‌ها
print(df)
```

23) مدل mvcرا در قالب یک برنامه پیاده سازی کنید؟ مدل MVC (Model-View-Controller) یک الگوی طراحی نرم‌افزاری است که برای جداسازی مسئولیت‌ها در یک برنامه استفاده می‌شود. این الگو به ما کمک می‌کند تا کدهای خود را سازمان‌دهی کنیم و قابلیت نگهداری و توسعه برنامه را افزایش دهیم.

:class Model

:def __init__(self)

[] = self.names

:def add_name(self, name)

self.names.append(name)

:def get_names(self)

return self.names

24) یک الگوی طراحی نرم‌افزاری است که به Clean Architectureرا در قالب یک برنامه پیاده سازی کنید؟ معماری Clean Architecture24) معماری جداسازی مسئولیت‌ها و ایجاد کدی قابل تست و نگهداری کمک می‌کند. این معماری به چهار لایه اصلی تقسیم می‌شود:

1. **Entities**: مدل‌های دامنه و منطق کسب‌وکار.
2. **Use Cases**: منطق کاربردی که عملیات خاصی را انجام می‌دهد.
3. **Interface Adapters**: تبدیل داده‌ها بین لایه‌های مختلف.
4. **Frameworks and Drivers**: لایه‌های خارجی مانند پایگاه داده‌ها و رابط‌های کاربری

/clean_architecture_example

|

/entities —├

| └— book.py

|

/use_cases —├

| └— book_use_case.py

|

/interface_adapters —├

| └— book_controller.py

|

└— main.py

25) اصول SOLiidرا در زبان Rustپیاده سازی نمایید؟

صول SOLID مجموعه‌ای از اصول طراحی نرم‌افزاری هستند که به توسعه‌دهندگان کمک می‌کنند تا کدهای قابل نگهداری، قابل تست و انعطاف‌پذیر بنویسند. این اصول شامل موارد زیر هستند:

1. **Single Responsibility Principle (SRP)**: هر کلاس باید تنها یک مسئولیت داشته باشد.
2. **Open/Closed Principle (OCP)**: کلاس‌ها باید برای گسترش باز و برای تغییر بسته باشند.
3. **Liskov Substitution Principle (LSP)**: اشیاء باید بتوانند با اشیاء زیر کلاس خود جایگزین شوند بدون اینکه رفتار برنامه تغییر کند.
4. **Interface Segregation Principle (ISP)**: کاربران نباید مجبور به وابستگی به رابط‌هایی باشند که از آن‌ها استفاده نمی‌کنند.
5. **Dependency Inversion Principle (DIP)**: وابستگی‌ها باید به انتزاع‌ها وابسته باشند، نه به کلاس‌های خاص

```
} pub struct Payment
    ,amount: f64
{
} impl Payment
} pub fn new(amount: f64) -> Self
    Payment { amount }
{
{

;pub struct PaymentProcessor

} impl PaymentProcessor
} pub fn process_payment(&self, payment: &Payment)
    ;println!("Processing payment of ${:.2}", payment.amount)
{
{

} pub struct Receipt
    ,payment: Payment
{
} impl Receipt
} pub fn new(payment: Payment) -> Self
    Receipt { payment }
{
} pub fn print(&self)
    ;println!("Receipt: ${:.2}", self.payment.amount)
{
{
```