

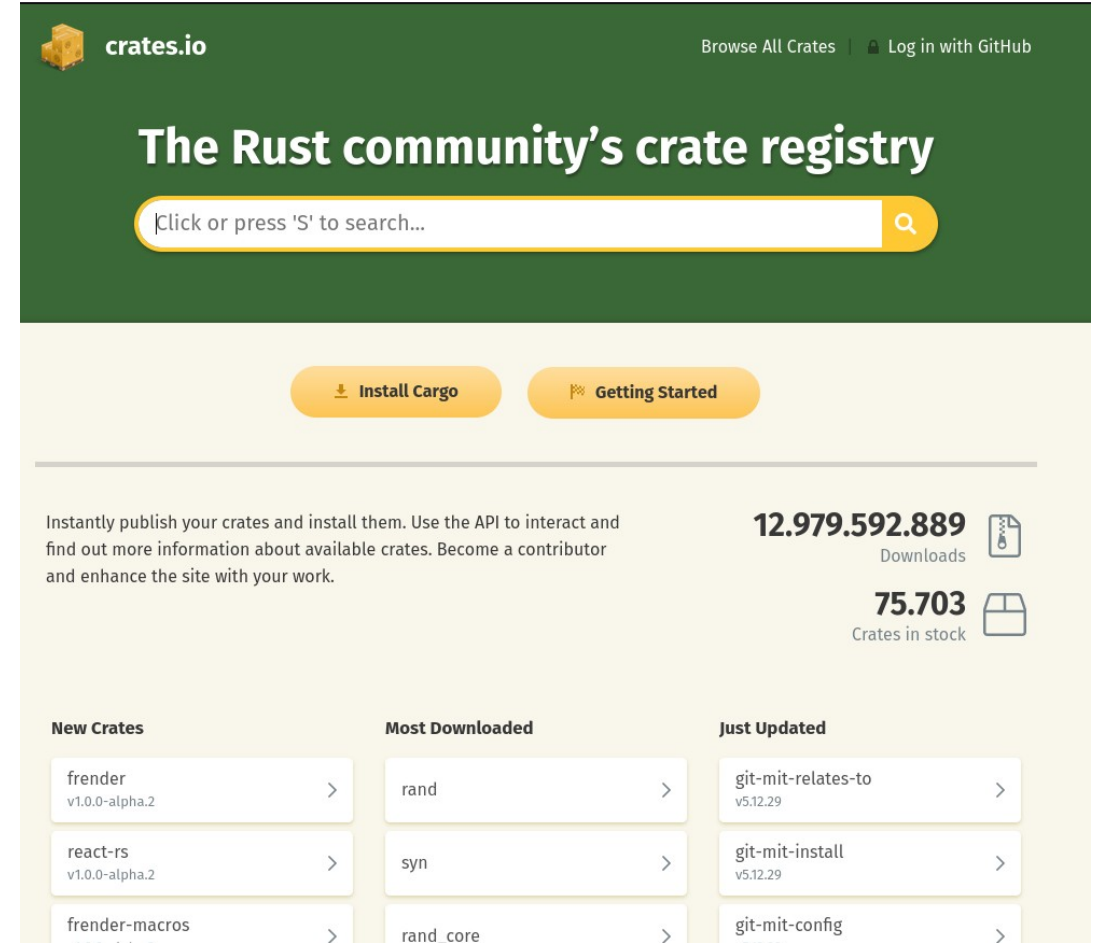
Rust: Crate, Test

Hafta - 3

Crate

Projenize dahil edebileceğiniz Rust kütüphaneleridir.

Crate'lerin bulunduğu adres: crates.io



The screenshot shows the crates.io website, which is the Rust community's crate registry. The header features the crates.io logo and a navigation bar with links to "Browse All Crates" and "Log in with GitHub". The main heading is "The Rust community's crate registry", followed by a search bar with the placeholder text "Click or press 'S' to search...". Below the search bar are two buttons: "Install Cargo" and "Getting Started". The main content area displays statistics: "12.979.592.889 Downloads" and "75.703 Crates in stock". Below the statistics are three columns of featured crates: "New Crates", "Most Downloaded", and "Just Updated". Each column lists three crates with their names and versions, and a right arrow indicating more details.

New Crates	Most Downloaded	Just Updated
frender v1.0.0-alpha.2	rand	git-mit-relates-to v5.12.29
react-rs v1.0.0-alpha.2	syn	git-mit-install v5.12.29
frender-macros	rand_core	git-mit-config

Örnek Kütüphane

kutuphane.rs:

```
pub fn public_fonksiyon() {  
    println!("Kütüphanemin public_fonksiyon'u çalıştı");  
}
```

```
$ rustc --crate-type=lib kutuphane.rs
```

Kütüphanemiz "**libkutuphane.rlib**" ismiyle oluşturuldu.

```
$ ls lib*  
libkutuphane.rlib
```

Örnek Kütüphane

main.rs:

```
fn main() {  
    kutuphane::public_fonksiyon();  
}
```

```
$ rustc main.rs --extern kutuphane=libkutuphane.rlib
```

Kütüphanemiz "libkutuphane.rlib" ismiyle oluşturuldu.

```
$ ./main
```

Kütüphanemin public_fonksiyon'u çalıştı

Cargo ile Crate oluşturma

```
$ cargo init --lib
```

```
$ tree
```

```
.
├── Cargo.toml
└── src
    └── lib.rs
```

```
1 directory, 2 files
```

Cargo ile Crate oluşturma

```
$ tree
```

```
├── Cargo.toml
└── src
    └── lib.rs
```

```
----> lib.rs:
```

```
pub fn public_fonksiyon() {
    println!("Kütüphanemin public_fonksiyon'u çalıştı");
}

fn private_fonksiyon() {
    println!("Kütüphanemin private_fonksiyon'u çalıştı");
}
```

```
$ cargo build --release
```

```
$ ls ./target/release/lib*
libcrateornek.d
libcrateornek.rlib
```

Cargo ile Crate oluşturma

Kütüphanemizin projesini, programımızın projesi içine taşıyalım:

```
$ tree
```

```
├── Cargo.toml
├── crateornek
│   ├── Cargo.toml
│   └── src
│       └── lib.rs
└── src
    └── main.rs
```

```
3 directories, 4 files
```

Cargo ile Crate oluşturma

```
$ tree
```

```
.
├── Cargo.toml
├── crateornek
│   ├── Cargo.toml
│   └── src
│       └── lib.rs
└── src
    └── main.rs
```

--->

Cargo.toml:

```
[package]
name = "crate-kullanan-ornek"
version = "0.1.0"
edition = "2021"

[dependencies]
crateornek = { path = "./crateornek" } // crateornek'i projemize yerel diziniyle ekledik.
```

```
$ tree
```

```
.
├── Cargo.toml
├── crateornek
│   ├── Cargo.toml
│   └── src
│       └── lib.rs
└── src
    └── main.rs
```

--->

Cargo.toml:

```
[package]
name = "crateornek"
version = "0.1.0"
edition = "2021"

[dependencies]
```


Cargo ile Crate oluşturma

main.rs:

```
use crateornek;  
  
fn main() {  
    crateornek::public_fonksiyon();  
}
```

\$ cargo run

```
Compiling crateornek v0.1.0      (/home/ef/Rust/crate-kullanan-ornek/crateornek)  
Compiling crate-kullanan-ornek v0.1.0 (/home/ef/Rust/crate-kullanan-ornek)
```

```
Finished dev [unoptimized + debuginfo] target(s) in 0.93s  
Running `target/debug/crate-kullanan-ornek`
```

Kütüphanemin public_fonksiyon'u çalıştı

Cargo ile Crate oluşturma

```
$ tree
```

```
.
├── Cargo.toml
├── crateornek
│   ├── Cargo.toml
│   └── src
│       └── lib.rs
└── src
    └── main.rs
```

----> lib.rs:

```
pub mod ara_modul {
    pub fn public_fonksiyon() {
        println!("Kütüphanemin public_fonksiyon'u çalıştı");
    }

    fn private_fonksiyon() {
        println!("Kütüphanemin private_fonksiyon'u çalıştı");
    }
}
```

```
$ tree
```

```
.
├── Cargo.toml
├── crateornek
│   ├── Cargo.toml
│   └── src
│       ├── lib.rs
│       └── main.rs
└── src
    └── main.rs
```

----> main.rs:

```
use crateornek::ara_modul;

fn main() {
    ara_modul::public_fonksiyon();
}
```

```
$ cargo run
```

```
Kütüphanemin public_fonksiyon'u çalıştı
```

Örnek crate: "serde"



serde 1.0.136

A generic serialization/deserialization framework

[#serde](#) [#serialization](#) [#no_std](#)

Metadata

3 days ago

MIT OR Apache-2.0

76.2 kB

Install

Add the following line to your Cargo.toml file:

```
serde = "1.0.136"
```

Homepage

serde.rs

Documentation

docs.serde.rs/serde

Repository

github.com/serde-rs/serde

Örnek crate: "serde"

Serde in action

```
use serde::{Serialize, Deserialize};

#[derive(Serialize, Deserialize, Debug)]
struct Point {
    x: i32,
    y: i32,
}

fn main() {
    let point = Point { x: 1, y: 2 };

    // Convert the Point to a JSON string.
    let serialized = serde_json::to_string(&point).unwrap();

    // Prints serialized = {"x":1,"y":2}
    println!("serialized = {}", serialized);

    // Convert the JSON string back to a Point.
    let deserialized: Point = serde_json::from_str(&serialized).unwrap()

    // Prints deserialized = Point { x: 1, y: 2 }
    println!("deserialized = {:?}", deserialized);
}
```

Tests

Test'ler yazdığımız programın doğru çalışıp çalışmadığını test ettiğimiz kodlardır.

```
$ tree
```

```
.
├── Cargo.toml
├── src
│   └── lib.rs
└── tests
    └── testim.rs
```

2 directories, 4 files

Cargo.toml

```
[package]
name = "test-ornek"
version = "0.1.0"
edition = "2021"
```

```
[dependencies]
```

Tests

\$ tree

```
.
├── Cargo.toml
├── src
│   └── lib.rs
└── tests
    └── testim.rs
```

----> lib.rs:

```
pub mod modul {
    pub fn sayi() -> i32 {
        42
    }
}
```

\$ tree

```
.
├── Cargo.toml
├── src
│   └── lib.rs
└── tests
    └── testim.rs
```

----> testim.rs:

```
#[cfg(test)]
mod tests {
    use test_ornek::modul;

    #[test]
    fn test_42() {
        assert_eq!(modul::sayi(), 42);
    }
}
```

Tests

```
$ tree
```

```
.
├── Cargo.toml
├── src
│   └── lib.rs
└── tests
    └── testim.rs --->
```

```
testim.rs:
```

```
#[cfg(test)]
mod tests {
    use test_ornek::modul;

    #[test]
    fn test_42() {
        assert_eq!(modul::sayi(), 42);
    }
}
```

```
$ cargo test
```

```
Running tests/testim.rs (target/debug/deps/testim-4cd276d703205a1f)
```

```
running 1 test
test tests::test_42 ... ok
```

Homeworks

1. Make Person { name:String, age:u8, gender:Gender::Male } struct transformable to JSON string with serde & serde_json crate:
 - Example: {"name":"Emin","age":24,"gender":"Male"}
2. Write 3 test cases for first task.
3. Write jsonified output of the Person struct to person.json file.
 - Examples: <https://doc.rust-lang.org/std/fs/struct.File.html>