

---

# RUST

Uvod  
Podešavanje okruženja za rad

---

# RUST?

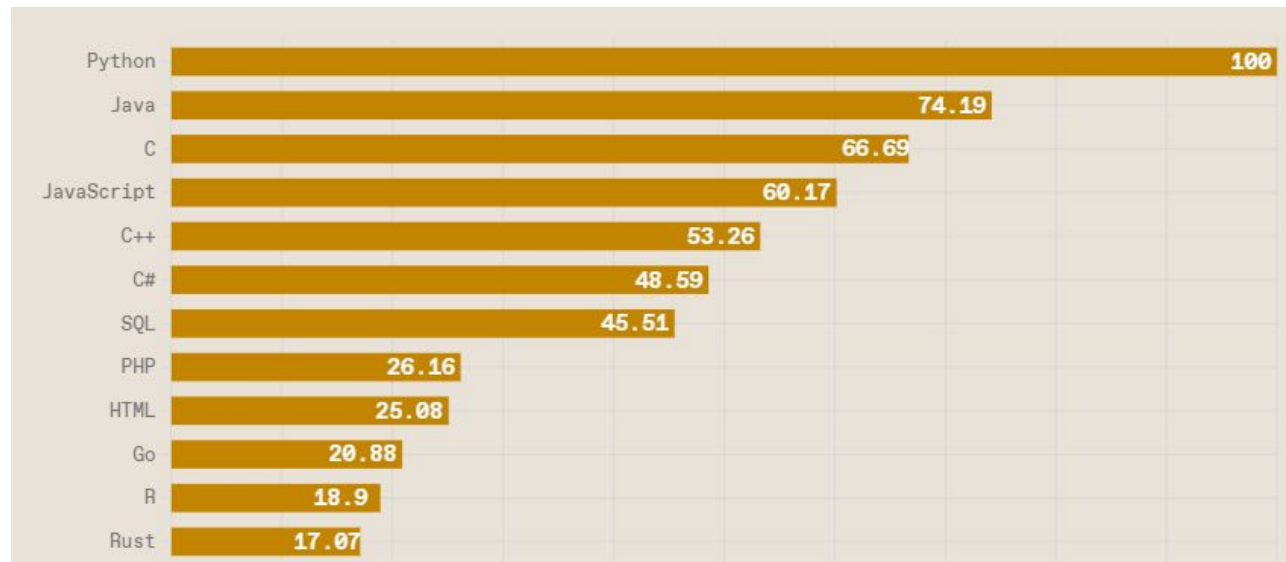
- Sistemski programski jezik koji se izvršava neverovatno brzo, sprečava semantičke greške i garantuje bezbednost niti.
- Dizajniran je tako da omogući:
  - brzinu,
  - bezbednost,
  - konkurentnost.
- Omogućava kontrolu nad korišćenjem memorije i održava blisku vezu između primitivnih operacija jezika i onih mašina na kojima radi → programeri mogu da predvide troškove operacija (vreme i prostor)
- Idealan je za različite grupe ljudi:
  - programeri,
  - studenti,
  - kompanije,
  - ljudi koji cene brzinu i stabilnost.

# Sistemsko programiranje

- Sistemsko programiranje uključuje projektovanje i pisanje računarskih programa koji omogućavaju hardveru da se poveže sa programerom i korisnikom, što dovodi do efektivnog izvršavanja aplikativnog softvera na računarskom sistemu.
- Oblasti sistemskog programiranja:
  - Operativni sistemi,
  - Upraljivački programi (device drivers),
  - Embedded sistemi,
  - Sistemi u realnom vremenu (Real Time Systems),
  - Umrežavanje,
  - Virtuelizacija i kontejnerizacija,
  - Primena računarstva u naučne svrhe,
  - Video igrice.
- C, C++, Ada, D

# Ko koristi RUST?

- Amazon
- Facebook
- Firefox
- Discord
- Dropbox
- Coursera



# Zašto RUST?

- Brzina
- **Bezbednost** - statički tipiziran, kod se proverava u vreme kompajliranja radi bezbednosti memorije
- **Pouzdana paralelnost**

# Rust

```
#include <stdio.h>
```

```
int main() {
```

```
    // printf() displays the string inside quotation
```

```
    printf("Hello, World!");
```

```
    return 0;
```

```
}
```

```
fn main() {
```

```
    println!("Hello, world!");
```

```
}
```

# Instaliranje RUST-a

- Komanda za instaliranje:
  - `$ curl --proto '=https' --tlsv1.3 https://sh.rustup.rs -sSf | sh`
- Komanda za update:
  - `$ rustup update`
- Komanda za proveru da li je uspešno instaliran:
  - `$ rustc --version`

# Provera da li je Rust dobro instaliran?

- Napraviti Rust program koji na konzolu ispisuje

„Ovo je nas prvi kod napisan u Rust-u!”



# Koraci za rešavanje zadatka sa prethodnog slajda:

- 1) Napraviti fajl sa ekstenzijom `.rs` u koji ćemo smestiti Rust kod.
- 2) U fajl dodati *main* funkciju i neophodan kod za realizaciju zadatka.
- 3) Kompajlirati napisani kod.

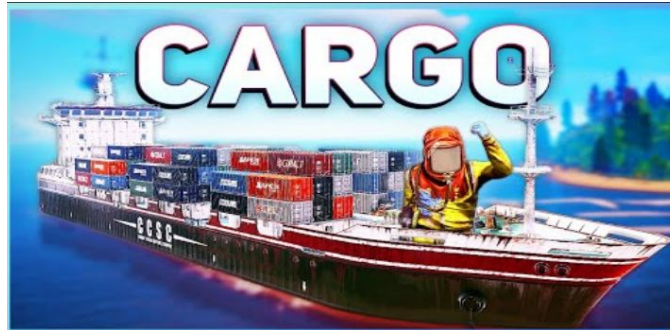
*rustc main.rs*

- 4) Pokrenuti aplikaciju.

*./main*

# CARGO

- Alat za razrešavanje zavisnosti i upravljanje projektom

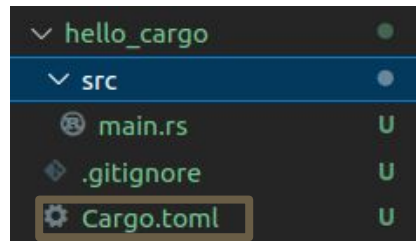


# CARGO

- Kreiranje projekta
  - ***cargo new naziv\_projekta*** (Pr. *cargo new hello\_cargo*)
  - ***cargo new --lib naziv\_projekta***



# CARGO - konfiguracioni fajl



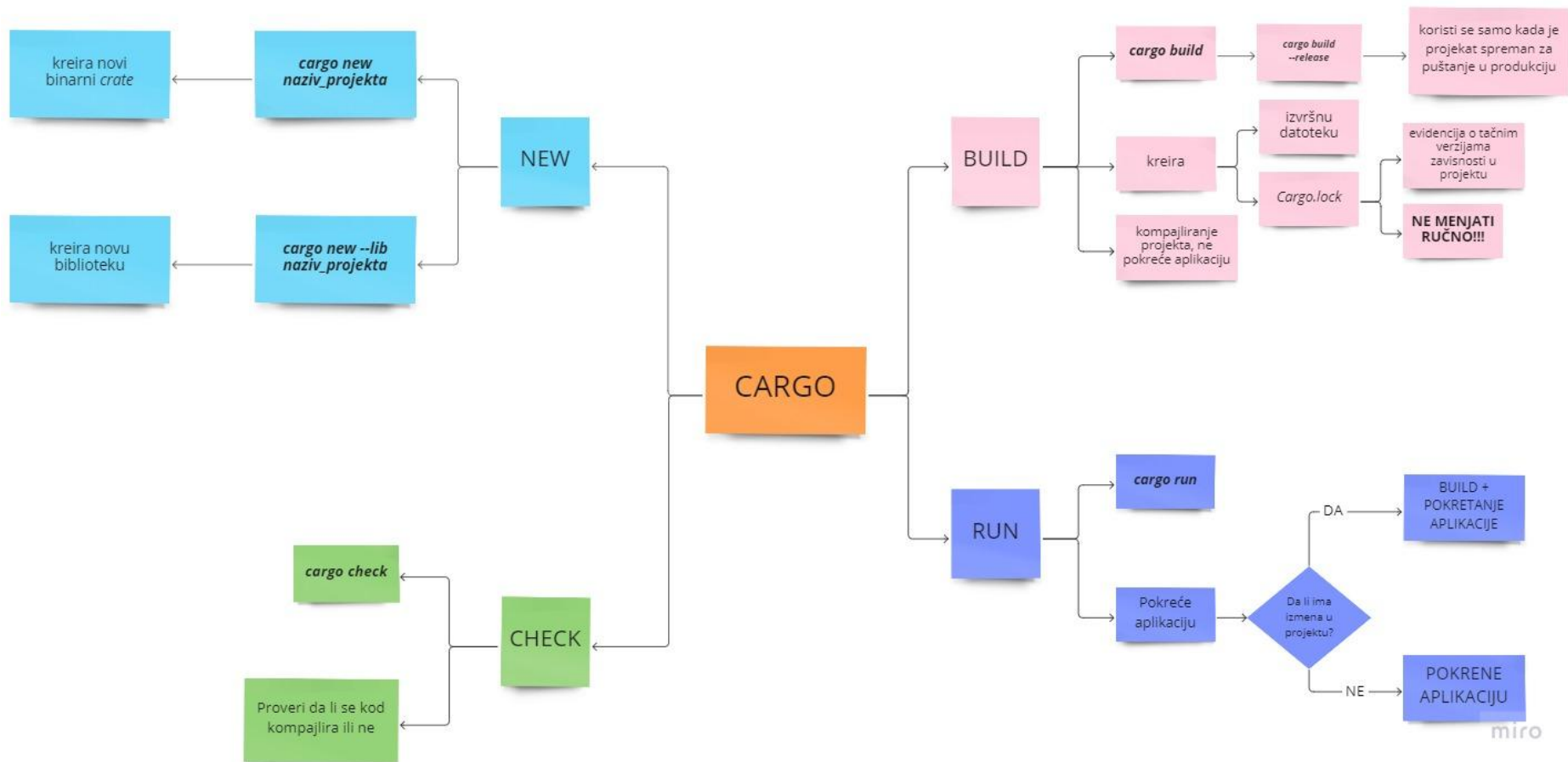
- TOML (*Tom's Obvious, Minimal Language*) - CARGO konfiguracioni jezik

Odeljak koji ukazuje da sledeći *statement-i* definišu paket

Informacije o konfiguraciji koje su neophodne za kompajliranje

Odeljak u kojem će biti navedene sve zavisnosti

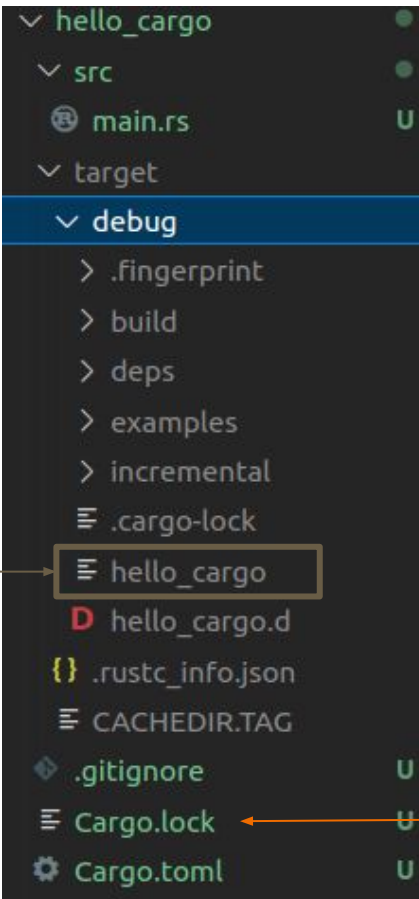
```
1 [package]
2   name = "hello_cargo"
3   version = "0.1.0"
4   edition = "2021"
5
6   # See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html
7
8 [dependencies]
9
```



# CARGO - BUILD

- ***cargo build***
- Kreira izvršnu datoteku, ne pokreće aplikaciju.
- *Cargo.lock* - fajl u kojem se vodi evidencija o tačnim verzijama zavisnosti u projektu. Cargo sam upravlja njegovim sadržajem. **NE MENJATI RUČNO!!!**
- ***cargo build --release***

```
jovana@jovana:~/Desktop/rust/hello_cargo$ cargo build
Compiling hello_cargo v0.1.0 (/home/jovana/Desktop/rust/hello_cargo)
Finished dev [unoptimized + debuginfo] target(s) in 1.20s
```



Izvršna datoteka  
koju je neophodno  
pokrenuti

Pojaviće se kada se prvi  
put pokrene *build*. Vodi  
računa o verzijama  
zavisnosti u projektu. **NE  
MENJATI RUČNO!!!**

# CARGO - RUN

- ***cargo run***
- Pokreće aplikaciju:
  - Prvo proveriti da li ima izmena u projektu
    - Ima izmena - uradi *cargo build* i pokrene aplikaciju
    - Nema izmena - samo pokrene izvršni fajl



# CARGO - CHECK

- ***cargo check***
- Proverava da li se kod kompajlira, *ne kreira izvršni fajl!*

# CRATE

- Najmanja količina koda koju Rust kompajler istovremeno razmatra.
  - Može da bude i jedan `.rs` fajl ako ga kompajlirate komandom `rustc`
- Oblici:
  - *Binarni* - programi koje možete da prevedete u izvršnu datoteku, koju možete da pokrenete. Obavezno sadrži *main* funkciju koja definiše šta se dešava kada se izvršni fajl pokrene.
  - *Biblioteka* - programi koji se ne kompajliraju u izvršni i nemaju *main* funkciju.

# PAKET

- Skup jednog ili više *crate-ova* koji pruža skup funkcionalnosti.
- Sadrži *Cargo.toml* datoteku
- Po konvenciji ako paket sadrži:
  - *src/main.rs* je koren *binarnog crate-a* sa istim imenom kao paket,
  - *src/lib.rs* je paket *biblioteka* sa istim imenom kao paket.
- Može da sadrži više binarnih *crate-ova*, ali samo jednu biblioteku.
- Cargo prosleđuje korenske fajlove *crate-a* u *rustc* da bi napravio biblioteku ili binarni fajl
  - Ako paket sadrži i *main.rs* i *lib.rs* onda će se kreirati dva *crate-a*, binarni i biblioteka, oba sa istim imenom kao i paket.
  - U jednom paketu može da postoji više binarnih *crate-ova*, svaki fajl će da bude zaseban *binarni crate*.

# CARGO - DODATNE FUNKCIONALNOSTI

- Prilagodite verziju kroz profile izdanja
- Objavite *crate-a* na *crates.io*
- Organizujete velike projekte koristeći radne prostore
- Instalirate binarne datoteke na *crates.io*
- Proširite Cargo koristeći prilagođene komande

# 1. PROFILI IZDANJA

- Profili izdanja - unapred definisani i prilagodljivi profili sa različitim konfiguracijama koje omogućavaju programeru da ima veću kontrolu nad različitim opcijama kompajliranja koda.
- Svaki profil se nezavisno konfiguriše.
- Glavni profili:
  - *dev*
    - *cargo build*
    - Podrazumevane vrednosti za razvoj
  - *release*
    - *cargo build --release*
    - Podrazumevane vrednosti za verzije izdanja

```
$ cargo build
  Finished dev [unoptimized + debuginfo] target(s) in 0.0s
$ cargo build --release
  Finished release [optimized] target(s) in 0.0s
```

# 1. PROFILI IZDANJA

- Postoji podrazumevano podešavanje za svaki od profila u sekciji *[profile.\*]* u fajlu *Cargo.toml*

```
[profile.dev]
opt-level = 0

[profile.release]
opt-level = 3
```

- *opt-level* - broj optimizacija koje će Rust da primeni na kod
  - Primena više optimizacija produžava vreme kompajliranja

## 2. POSTAVLJANJE *CRATE*-A NA *CRATES.IO*

- *Crate*-ovi koje koristimo u projektima se povlače sa *crates.io*, ali možemo i da postavljamo i *crate*-ove koje mi razvijemo
- Funkcije koje ljudima olakšavaju pronalaženje i korišćenje vašeg objavljenog paketa
- Potrebno je:
  - Napraviti korisnu dokumentaciju
  - Kreirati API upotrebom *pub use*
  - Podesiti naloga na *Crates.io*
  - Dodati metapodatke u *crate*
  - Postaviti *crate*

# Kreiranje dokumentacije

- Dokumentacije je namenjena da programerima objasni kako da koriste *crate*, a ne kako je on implementiran
- Dokumentacioni komentar - `///`
  - Generisaće HTML dokumentaciju
  - Treba da se nađu iznad stavke koju dokumentuje
- Komanda za kreiranje dokumentacije
  - `cargo doc`
  - `cargo doc --open` - odmah otvori dokumentaciju u pretraživaču



# Kreiranje dokumentacije

```
/// Adds one to the number given.
///
/// # Examples
///
/// ```
/// let arg = 5;
/// let answer = my_crate::add_one(arg);
/// assert_eq!(6, answer);
/// ```
pub fn add_one(x: i32) -> i32 {
    x + 1
}
```

my\_crate

Functions

add\_one

Crates

my\_crate



Click or press 'S' to search, '?' for more options...



Function my\_crate::add\_one

[–][src]

```
pub fn add_one(x: i32) -> i32
```

[–] Adds one to the number given.

Examples

```
let arg = 5;
let answer = my_crate::add_one(arg);

assert_eq!(6, answer);
```

# Kreiranje dokumentacije

- Sekcija se definiše sa #
- Sekcije koje se najčešće koriste:
  - Panics
    - Koristi se za funkcije koje mogu da se „uspaniče” (funkcije koje dovedu vaš program u stanje koje ne može da se obradi)
  - Errors
    - Ako funkcija vrati neki rezultat, opisivanje vrste grešaka koje se mogu pojaviti i koji uslovi mogu uzrokovati vraćanje tih grešaka
  - Safety
    - Ako funkcija nije bezbedna za pozivanje treba da postoji odeljak koji objašnjava zašto funkcija nije bezbedna

# Dokumentacijski komentari kao testovi

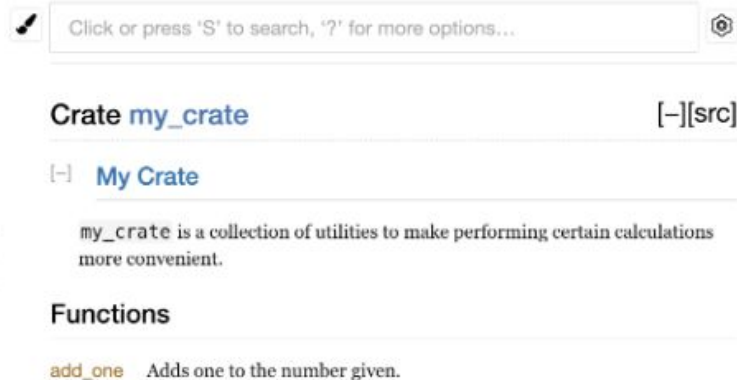
- U komentare možete da dodate i primere koda → pokretanje kreiranja dokumentacije pokrenuće i izvršavanje primera koje ste napisali

# Komentarisanje sadržanih stavki

- `//!` - dodaje dokumentaciju na stavku koja sadrži komentar, a ne na stavke koje slede nakon komentara
- Koriste se unutar korenske datoteke *crate-a* ili unutar modula, da dokumentuju *crate* ili modul u celini
- Posle ovog tipa komentara nema koda
- *cargo doc - open* → prikazaće se na naslovnoj strani dokumentacije iznad liste javno dostupnih stavki *crate-a*

```
//! # My Crate
//!
//! `my_crate` is a collection of utilities to make performing certain
//! calculations more convenient.

/// Adds one to the number given.
// --snip--
```



# Izvoz javnog API-a upotrebom *pub use*

- Prilikom kreiranja *crate-a* sami kreirate njegovu strukturu, ali ta struktura možda neće biti zgodna za vaše korisnike
- Dobra stvar:
  - Možete da izvezete javnu strukturu koja se razlikuje od vaše privatne → *pub use*
  - Ponovni izvoz uzima javnu stavku na jednom mestu i čini je javnom na drugom mestu, kao da je umesto toga definisana na drugom mestu

```

//! # Art
//!
//! A library for modeling artistic concepts.

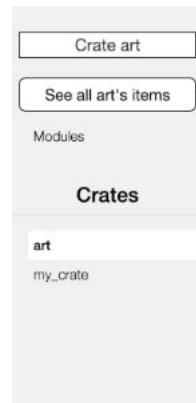
pub mod kinds {
    /// The primary colors according to the RYB color model.
    pub enum PrimaryColor {
        Red,
        Yellow,
        Blue,
    }

    /// The secondary colors according to the RYB color model.
    pub enum SecondaryColor {
        Orange,
        Green,
        Purple,
    }
}

pub mod utils {
    use crate::kinds::*;

    /// Combines two primary colors in equal amounts to create
    /// a secondary color.
    pub fn mix(c1: PrimaryColor, c2: PrimaryColor) -> SecondaryColor {
        // --snip--
    }
}

```




## Crate [art](#)

[--][src]

[-] [Art](#)

A library for modeling artistic concepts.

## Modules

[kinds](#)  
[utils](#)

# Izvoz javnog API-a upotrebom *pub use*

- Da bi programer koristio bilo šta iz biblioteke prikazane na prethodnom slajdu, morao bi da bude upoznat sa njenom unutrašnjom strukturom i da zna gde šta u okviru nje da traži

```
use art::kinds::PrimaryColor;
use art::utils::mix;

fn main() {
    let red = PrimaryColor::Red;
    let yellow = PrimaryColor::Yellow;
    mix(red, yellow);
}
```

# Izvoz javnog API-a upotrebom *pub use*

- Da bismo uspešno rešili problem sa prethodnog slajda, neophodno je da upotrebimo *pub use*

```
//! # Art
//!
//! A library for modeling artistic concepts.
```

```
pub use self::kinds::PrimaryColor;
pub use self::kinds::SecondaryColor;
pub use self::utils::mix;
```

```
pub mod kinds {
    // --snip--
}
```

```
pub mod utils {
    // --snip--
}
```

The screenshot shows the Rust documentation for the 'art' crate. On the left, there is a sidebar with a search bar and buttons for 'Crate art' and 'See all art's items'. Below these are links for 'Re-exports' and 'Modules'. The main content area shows the crate name 'Crate art' with a source link '[--][src]'. Below that is the description 'A library for modeling artistic concepts.' and a section titled 'Re-exports' which contains the following code:

```
pub use self::kinds::PrimaryColor;
pub use self::kinds::SecondaryColor;
pub use self::utils::mix;
```

Below the 'Re-exports' section is a 'Modules' section listing 'kinds' and 'utils'.




# Izvoz javnog API-a upotrebom *pub use*

```
use art::kinds::PrimaryColor;
use art::utils::mix;

fn main() {
    let red = PrimaryColor::Red;
    let yellow = PrimaryColor::Yellow;
    mix(red, yellow);
}
```

*pub use*



```
use art::mix;
use art::PrimaryColor;

fn main() {
    // --snip--
}
```

# Izvoz javnog API-a upotrebom *pub use*

- Možete da napraviti značajnu razliku u iskustvu ljudi koji koriste biblioteku
- Kreiranje korisne javne strukture API-a je više umetnost nego nauka
- Odabir *pub use* pruža vam fleksibilnost u načinu na koji interno struktuirate svoju biblioteku i razdvaja unutrašnju strukturu od onoga što predstavljate drugim korisnicima

# Podešavanje naloga na *Crates.io*

- Pre objavljivanja *crate-a* prvo morate da napravite nalog na *crates.io* i dobijete API token.
  - <https://crates.io/> - prijavite se pomoću github naloga
  - Odeti na podešavanja vašeg naloga i preuzmete API token
  - Pokrenuti prijavu na *crates.io* sa dobijenim tokenom

```
$ cargo login abcdefghijklmnopqrstuvwxyz012345
```

# Dodavanje metapodataka

- Metapodaci se dodaju u sekciju *package* u *Cargo.toml* fajl
- *Crate* mora da ima definisano:
  - Jedinstveno ime
    - Ako je koristite u lokalu onda možete da je nazovete kako god želite, ali na *crates.io* se imena dodeljuju po principu „prvi dođe, prvi uslužen“
    - Pre nego što pokušate da dodelite ime *crate-u*, potražite ime koje želite da koristite
  - Opis
    - Kako bi korisnici znali šta vaš *crate* radi
    - Sastoji se od jedne ili dve rečenice
    - Pojaviće se u rezultatima pretrage
  - Licenca
    - Kako bi korisnici znali pod kojim uslovima mogu da koriste *crate*
    - Vrednost identifikatora licence
    - *The Linux Foundation's Software Package Data Exchange (SPDX)* - lista licenci koje mogu da se koriste
    - Možete koristiti i svoju licencu, tako što ćete napraviti poseban fajl u kojem ćete definisati svoju licencu

# Objavljivanje *crate-a*

- *cargo publish*
- ***Vodite računa, objavljivanje je trajno - verzija se nikad ne može prepisati, a kod se ne može obrisati!!!***
- *Crates.io* - stalna arhiva koda tako da će nadogradnja svih projekata koji zavise od *crate-a* u *crates.io* nastaviti da rade. [brisanje → narušava ovaj cilj]
- Ne postoji ograničenje na broj verzija *crate-a* koje možete da objavite.

# Postavljanje nove verzije *crate-a*

- Potrebno je samo da na ispravan način izmenite verziju *crate-a* i ponovo odradite komandu za objavljivanje *crate-a*

## „Brisanje” zastarele verzije *crate-a*

- Rekli smo da ne može da se izbriše *crate* koji se postavi, ali zato možete da sprečite da ga neko koristi
- Poništavanje verzije sprečava da novi projekti zavise od te verzije, dok se dozvoljava da se svi postojeći projekti koji zavise od njega nastave
- ***cargo yank --vers verzija\_crate-a***
- Ako hoćete da dozvolite da projekti opet mogu da koriste povučenu verziju možete da iskoristite komandu *undo*
  - ***cargo yank --vers verzija\_crate-a -undo***
- **ZAPAMTITE – OVO NE RADI BRISANJE CRATE-A!!!**

### 3. CARGO WORKSPACE

- Radni prostor pomaže u upravljanju više povezanih paketa (projekata) koji su razvijeni zajedno, tj pomaže nam da grupišemo više sličnih projekata.
- Skup paketa koji dele isti *Cargo.lock* i izlazni fajl.
  - Sve to zato što svi projekti koji se nađu u jednom radnom prostoru mogu da zavise jedan od drugog.
    - Veze između projekata morate eksplicitno da navedete → dodati zavisnost u *Cargo.toml* fajl
  - Jedan *Cargo.lock* osigurava da svi projekti u radnom prostoru koriste iste verzije zavisnih *crate-ova* i da projekti međusobno budu kompatibilni
  - *Jedan projekat ne može da koristi crate iz nekog drugog projekta!*



## 4. CARGO WORKSPACES - testovi

- Testovi su jako korisni kada imate više projekata koji zavise jedan od drugog.
- Komanda za pokretanje testova:
  - ***cargo test***
  - Pokreću se svi testovi napisani u svim projektima iz radnog prostora.

```
#[cfg(test)]
▶ Run Tests | Debug
mod tests {
    use super::*;

    #[test]
    ▶ Run Test | Debug
    fn it_works(){
        assert_eq!(5, add_one(4));
    }
}
```

```
jovana@jovana:~/Desktop/rust/add$ cargo test
Compiling add_one v0.1.0 (/home/jovana/Desktop/rust/add/add_one)
Compiling adder v0.1.0 (/home/jovana/Desktop/rust/add/adder)
Finished test [unoptimized + debuginfo] target(s) in 0.42s
Running unittests src/lib.rs (target/debug/deps/add_one-0eca324a9bfcd10f)

running 1 test
test tests::it_works ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s

Running unittests src/main.rs (target/debug/deps/adder-7edd375e414949bb)

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s

Doc-tests add_one

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

## 5. INSTALIRANJE BINARNIH FAJLOVA

- *cargo install naziv\_binarnog\_fajla*
- Nije namenjen zameni sistemskih paketa, služi isključivo za instaliranje paketa (projekata) koji mogu da se pokrenu.
- Svi binarni fajlovi koji se instaliraju čuvaju se u bin folderu gde ste instalirali *cargo*

## 6. PROŠIRIVANJE CARGO-A PRILAGOĐENIM KOMANDAMA

- *Cargo* je dizajniran tako da ga možete proširiti novim komandama.
- Komande oblika:
  - *naziv\_binarne\_datoteke cargo*