Structuring Rust Code

Zachary Bennett
Lead Software Developer

@z_bennett_ | zachbennettcodes.com

A Rust module helps organize structs, enums, traits and functions into cohesive units

Using Modules and Paths

```
// inline private module
mod coffee {

  // Coffee module code here…

}
```

```
// Public module

pub mod coffee;
```

Using Modules and Paths

This declares a public module – if this module is in the root crate, the compiler will look for the code associated with this module in the ”src/coffee.rs” file. If that file doesn’t exist, the compiler will look at the “src/coffee/mod.rs” file.

Modules and “use”

```
pub mod coffee {

  pub struct Coffee {}
  pub struct CoffeeMachine {}
  pub struct CoffeeCup

}
```

```
use crate::coffee::Coffee;

use create::coffee::{Coffee, CoffeeMachine};

use crate::coffee::*;
```

Using ”super” for Relative Paths

```
// Outer module code…

fn make_coffee() {}

// submodule
mod espresso {

  fn start_hybrid_mode() {
    super::make_coffee();
  }

}
```

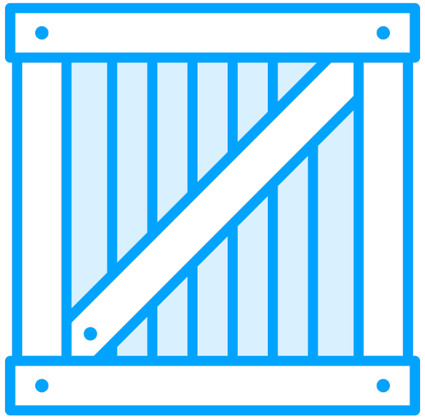
Demo

Modules and the filesystem

Using modules to structure code

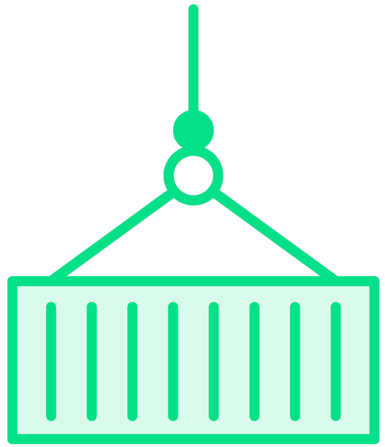
Crates and Packages

Organizing Rust Programs



Crates

A grouping of modules that produces either a library or an executable

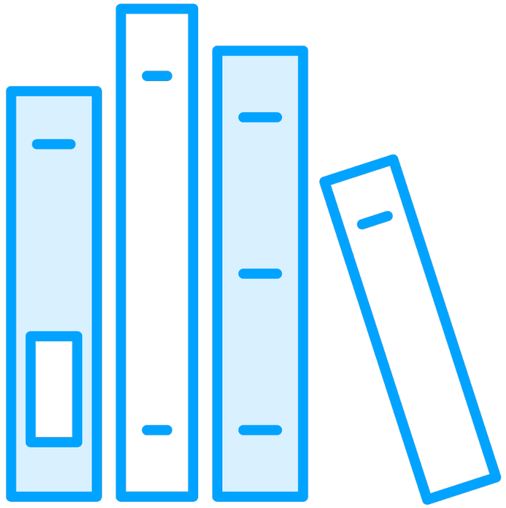


Packages

Used by Cargo, the Rust package manager, helps work with crates

Sometimes packages are referred to as crates!

Two Types of Crates



Library Crate

A crate that contains code meant to be consumed by other Rust projects

Binary Crate

A crate that is compiled to an executable

A package can contain ”x” number of binary crates but only 1 library crate

Example Package Cargo.toml File

```
[package]
name = “test”
version = “0.1.0”
edition = “2021”
```

Third-party Dependencies

```
[package]
name = “test”
version = “0.1.0”
edition = “2021”

[dependencies]
time = “0.1.12”
regex = { git = https://github.com/rust-lang/regex.git }
```

Demo

Crates

- Binary
- Library

Demo

Adding third-party packages to a project

Changing versions of third-party packages