**Paths for Referring to an Item in the Module Tree**

# Path

A path can take two forms:

**Absolute path** is the full path starting from a crate root; for code from an external crate, the absolute path begins with the crate name, and for code from the current crate, it starts with the literal crate.

**Relative path** starts from the current module and uses self, super, or an identifier in the current module.

Example

src/lib.rs

mod front\_of\_house {

mod hosting {

fn add\_to\_waitlist() {}

}

}

pub fn eat\_at\_restaurant() {

// Absolute path

crate::front\_of\_house::hosting::add\_to\_waitlist();

// Relative path

front\_of\_house::hosting::add\_to\_waitlist();

}

// this will fail to build because hosting and add\_to\_waitlist() are private

Our preference in general is to specify absolute paths because it’s more likely we’ll want to move code definitions and item calls independently of each other.

In Rust, all items (functions, methods, structs, enums, modules, and constants) are private to parent modules by default. If you want to make an item like a function or struct private, you put it in a module.

Items in a parent module can’t use the private items inside child modules, but items in child modules can use the items in their ancestor modules. This is because child modules wrap and hide their implementation details, but the child modules can see the context in which they’re defined.

# Exposing Paths with the pub Keyword

Rust does give you the option to expose inner parts of child modules’ code to outer ancestor modules by using the pub keyword to make an item public.

Example

src/lib.rs

mod front\_of\_house {

pub mod hosting {

pub fn add\_to\_waitlist() {}

}

}

pub fn eat\_at\_restaurant() {

// Absolute path

crate::front\_of\_house::hosting::add\_to\_waitlist();

// Relative path

front\_of\_house::hosting::add\_to\_waitlist();

}

// this will be build successfully

While front\_of\_house is not public, because the eat\_at\_restaurant function is defined in the same module as front\_of\_house (that is, eat\_at\_restaurant and front\_of\_house are siblings), we can refer to front\_of\_house from eat\_at\_restaurant.

# Starting Relative Paths with super

We can construct relative paths that begin in the parent module, rather than the current module or the crate root, by using **super** at the start of the path. This is like starting a filesystem path with the .. syntax.

Example

src/lib.rs

fn deliver\_order() {}

mod back\_of\_house {

fn fix\_incorrect\_order() {

cook\_order();

super::deliver\_order();

}

fn cook\_order() {}

}

# Making Structs and Enums Public

We can also use pub to designate structs and enums as public, but there are a few details extra to the usage of pub with structs and enums. If we use pub before a struct definition, we make the struct public, but the struct’s fields will still be private. We can make each field public or not on a case-by-case basis.

Example

src/lib.rs

mod back\_of\_house {

pub struct Breakfast {

pub toast: String,

seasonal\_fruit: String,

}

impl Breakfast {

pub fn summer(toast: &str) -> Breakfast {

Breakfast {

toast: String::from(toast),

seasonal\_fruit: String::from("peaches"),

}

}

}

}

pub fn eat\_at\_restaurant() {

// Order a breakfast in the summer with Rye toast

let mut meal = back\_of\_house::Breakfast::summer("Rye");

// Change our mind about what bread we'd like

meal.toast = String::from("Wheat");

println!("I'd like {} toast please", meal.toast);

// The next line won't compile if we uncomment it; we're not allowed

// to see or modify the seasonal fruit that comes with the meal

// meal.seasonal\_fruit = String::from("blueberries");

}

In contrast, if we make an enum public, all of its variants are then public. We only need the pub before the enum keyword,

src/lib.rs

mod back\_of\_house {

pub enum Appetizer {

Soup,

Salad,

}

}

pub fn eat\_at\_restaurant() {

let order1 = back\_of\_house::Appetizer::Soup;

let order2 = back\_of\_house::Appetizer::Salad;

}

# References

<https://doc.rust-lang.org/book/ch07-03-paths-for-referring-to-an-item-in-the-module-tree.html>

# END OF FILE