**Defining Modules to Control Scope and Privacy**

# Start from the crate root

When compiling a crate, the compiler first looks in the crate root file (usually *src/lib.rs* for a library crate or *src/main.rs* for a binary crate) for code to compile.

# Declaring modules

In the crate root file, you can declare new modules; say, you declare a “garden” module with mod garden;. The compiler will look for the module’s code in these places:

* Inline, within curly brackets that replace the semicolon following mod garden
* In the file *src/garden.rs*
* In the file *src/garden/mod.rs*

# Declaring submodules

In any file other than the crate root, you can declare submodules. For example, you might declare mod vegetables; in *src/garden.rs*. The compiler will look for the submodule’s code within the directory named for the parent module in these places:

* Inline, directly following mod vegetables, within curly brackets instead of the semicolon
* In the file *src/garden/vegetables.rs*
* In the file *src/garden/vegetables/mod.rs*

# Paths to code in modules

Once a module is part of your crate, you can refer to code in that module from anywhere else in that same crate, as long as the privacy rules allow, using the path to the code. For example, an Asparagus type in the garden vegetables module would be found at crate::garden::vegetables::Asparagus.

# Private vs public

Code within a module is private from its parent modules by default. To make a module public, declare it with pub mod instead of mod. To make items within a public module public as well, use pub before their declarations.

# The use keyword

Within a scope, the use keyword creates shortcuts to items to reduce repetition of long paths. In any scope that can refer to crate::garden::vegetables::Asparagus, you can create a shortcut with use crate::garden::vegetables::Asparagus; and from then on you only need to write Asparagus to make use of that type in the scope.

NOTE:

Earlier, we mentioned that *src/main.rs* and *src/lib.rs* are called crate roots. The reason for their name is that the contents of either of these two files form a module named crate at the root of the crate’s module structure, known as the *module tree*.

backyard

├── Cargo.lock

├── Cargo.toml

└── src

├── garden

│   └── vegetables.rs

├── garden.rs

└── main.rs

# Example

$cargo new --lib restaurant

src/lib.rs

mod front\_of\_house {

mod hosting {

fn add\_to\_waitlist() {}

fn seat\_at\_table() {}

}

mod serving {

fn take\_order() {}

fn serve\_order() {}

fn take\_payment() {}

}

}

**The module tree**

crate

└── front\_of\_house

├── hosting

│ ├── add\_to\_waitlist

│ └── seat\_at\_table

└── serving

├── take\_order

├── serve\_order

└── take\_payment

# References

<https://doc.rust-lang.org/book/ch07-02-defining-modules-to-control-scope-and-privacy.html>

# END OF FILE