Separating Modules into Different Files
So far, all the examples in this chapter defined multiple modules in one file.
When modules get large, you might want to move their definitions to a separate
file to make the code easier to navigate.
For example, let’s start from the code in Listing {{#ref use-keyword}} that had multiple
restaurant modules. We’ll extract modules into files instead of having all the
modules defined in the crate root file. In this case, the crate root file is
src/lib.cairo.
First, we’ll extract the `front_of_house` module to its own file. Remove the
code inside the curly brackets for the `front_of_house` module, leaving only
the `mod front_of_house;` declaration, so that _src/lib.cairo_ contains the code
shown in Listing {{#ref front-extraction}}. Note that this won’t compile until we create the
_src/front_of_house.cairo_ file.
<span class="filename">Filename: src/lib.cairo</span>
```cairo,noplayground
{{#include ../listings/ch07-managing-cairo-projects-with-packages-crates-and-modules/
listing_13_front_extraction/src/lib.cairo:front-extraction}}
```
{{#label front-extraction}}
<span class="caption">Listing {{#ref front-extraction}}: Declaring the `front_of_house`
module whose body will be in _src/front_of_house.cairo_</span>
Next, place the code that was in the curly brackets into a new file named
_src/front_of_house.cairo_, as shown in Listing {{#ref module-foh}}. The compiler
knows to look
in this file because it came across the module declaration in the crate root
with the name `front_of_house`.
<span class="filename">Filename: src/front_of_house.cairo</span>
```cairo,noplayground
{{#include ../listings/ch07-managing-cairo-projects-with-packages-crates-and-modules/
listing_14_front_definition/src/lib.cairo}}
```
{{#label module-foh}}
<span class="caption">Listing {{#ref module-foh}}: Definitions inside the
`front_of_house` module in _src/front_of_house.cairo_</span>
Note that you only need to load a file using a `mod` declaration _once_ in your
module tree. Once the compiler knows the file is part of the project (and knows
where in the module tree the code resides because of where you’ve put the `mod`
statement), other files in your project should refer to the loaded file’s code
using a path to where it was declared, as covered in the ["Paths for Referring to an Item
in the Module Tree"][path] chapter.
In other words, `mod` is _not_ an “include” operation that you may have seen in other
programming languages.
Next, we’ll extract the `hosting` module to its own file. The process is a bit
different because `hosting` is a child module of `front_of_house`, not of the
root module. We’ll place the file for `hosting` in a new directory that will be
named for its ancestors in the module tree, in this case _src/front_of_house/_.

To start moving `hosting`, we change _src/front_of_house.cairo_ to contain only the declaration of the `hosting` module:
<span class="filename">Filename: src/front_of_house.cairo</span>
```cairo,noplayground
pub mod hosting;
```
Then we create a _src/front_of_house_ directory and a file _hosting.cairo_ to contain the definitions made in the `hosting` module:
<span class="filename">Filename: src/front_of_house/hosting.cairo</span>
```cairo,noplayground
pub fn add_to_waitlist() {}
```
If we instead put _hosting.cairo_ in the _src_ directory, the compiler would expect the _hosting.cairo_ code to be in a `hosting` module declared in the crate root, and not declared as a child of the `front_of_house` module. The compiler's rules for which files to check for which modules' code means the directories and files more closely match the module tree.
We've moved each module's code to a separate file, and the module tree remains the same. The function calls in `eat_at_restaurant` will work without any modification, even though the definitions live in different files. This technique lets you move modules to new files as they grow in size.
Note that the `use crate::front_of_house::hosting;` statement in _src/lib.cairo_ also hasn't changed, nor does `use` have any impact on what files are compiled as part of the crate. The `mod` keyword declares modules, and Cairo looks in a file with the same name as the module for the code that goes into that module.
[path]: ./ch07-03-paths-for-referring-to-an-item-in-the-module-tree.md
Summary
Cairo lets you split a package into multiple crates and a crate into modules so you can refer to items defined in one module from another module. You can do this by specifying absolute or relative paths. These paths can be brought into scope with a `use` statement so you can use a shorter path for multiple uses of the item in that scope. Module code is **private** by default.
{{#quiz ../quizzes/ch07-05-separate-modules.toml}}