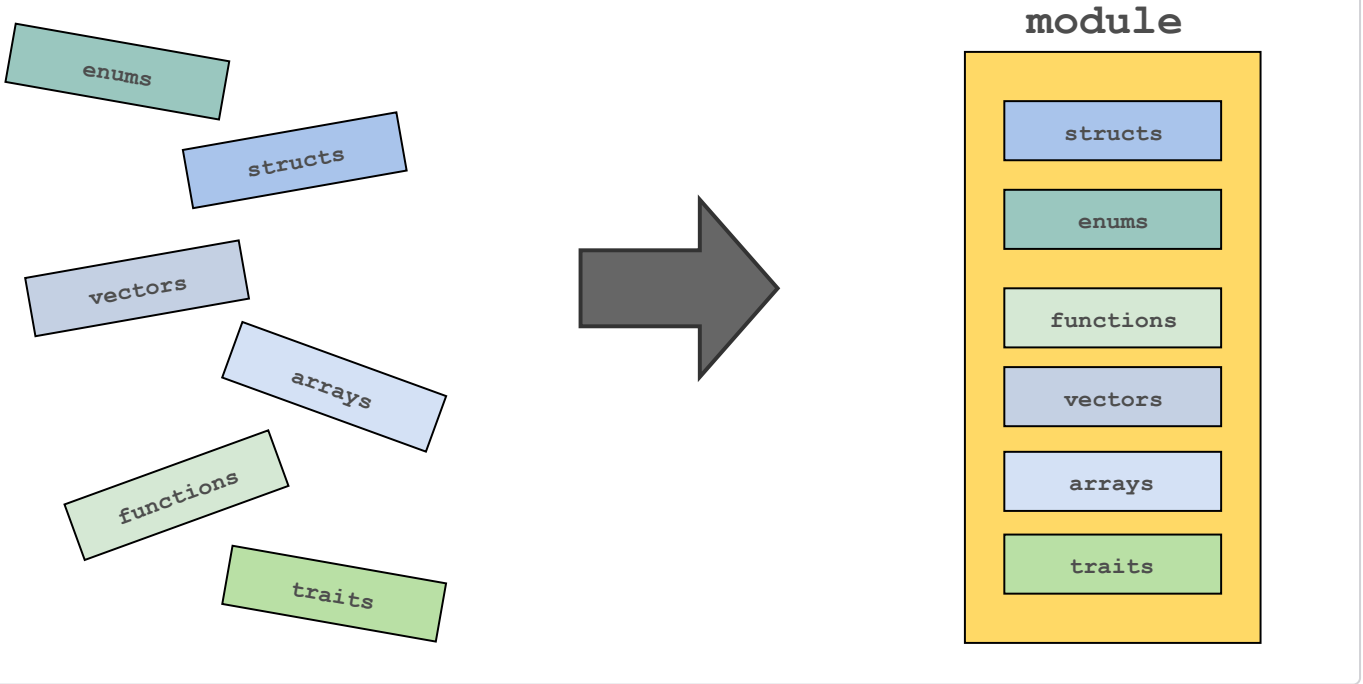
Introduction to Modules

This lesson will get you acquainted with organization of code through Modules in rust.

We'll cover the following

What Are Modules?

Modules are a collection of items that can contain structs, functions, enums, vectors, arrays, etc.



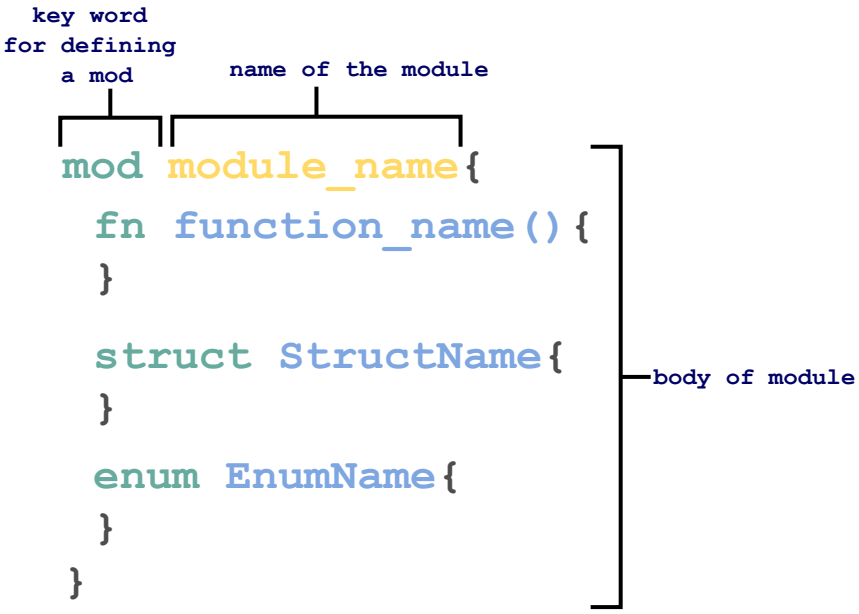
Why make a module?

As a result of making modules,

- the program *code becomes organized*.
- you can use the *same name for things like a struct*. For example, you can use the name Configuration in different modules and code it differently. Otherwise, you would have different clumsy names for the struct like EngineConfiguration, ConsoleConfiguration etc.

Declare a Module

To declare a module in Rust use the `mod` keyword followed by the name of the module and the body of the module within curly braces `{ }`.

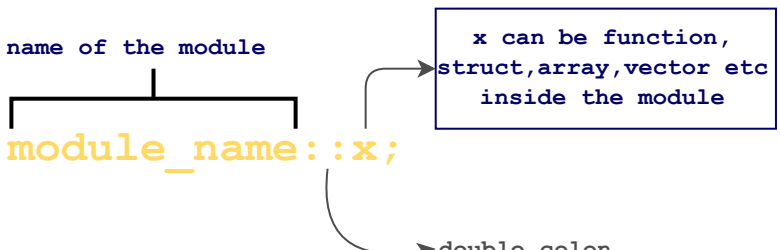


Define a module

> **Naming Convention**
>
> Name of the module should be written in **snake_case**.

Invoking a Module

The module can be invoked from anywhere within the program code.



Invoke a module

Keywords for Modules

The following keywords are used for declaring modules:

- `mod` - declares a new module
- `pub` - makes a public module
- `use` - imports the module in the local scope

> **Note:** Modules are declared by the `mod` keyword and are private by default.

Example

The following example makes use of the `mod` keyword to declare a module named `r`, and defines a function `print_statement` within the module:

```
// declare a module
mod r {
  fn print_statement(){
    println!("Hi, this a function of module r");
  }
}
// main function
fn main() {
  // invoke a module 'r'
   r::print_statement();
}
```

> The above code generates an **error**, ✕, because the function `print_statement` is private

---

Let's look at how `pub` can solve this issue in the next lesson!