Why Go Get Started **Docs** Packages Play Blog

Call your code from another module

In the previous section, you created a greetings module. In this section, you'll write code to make calls to the Hello function in the module you just wrote. You'll write code you can execute as an application, and which calls code in the greetings module.

Note: This topic is part of a multi-part tutorial that begins with Create a Go module.

1. Create a hello directory for your Go module source code. This is where you'll write your caller.

After you create this directory, you should have both a hello and a greetings directory at the same level in the hierarchy, like so:

```
<home>/
|-- greetings/
|-- hello/
```

For example, if your command prompt is in the greetings directory, you could use the following commands:

```
cd ..
mkdir hello
cd hello
```

2. Enable dependency tracking for the code you're about to write.

To enable dependency tracking for your code, run the go mod init command, giving it the name of the module your code will be in.

For the purposes of this tutorial, use example.com/hello for the module path.

```
$ go mod init example.com/hello
go: creating new go.mod: module example.com/hello
```

- 3. In your text editor, in the hello directory, create a file in which to write your code and call it hello.go.
- 4. Write code to call the Hello function, then print the function's return value.

To do that, paste the following code into hello.go.

```
package main

import (
    "fmt"

    "example.com/greetings"
)

func main() {
    // Get a greeting message and print it.
    message := greetings.Hello("Gladys")
    fmt.Println(message)
}
```

In this code, you:

- Declare a main package. In Go, code executed as an application must be in a main package.
- o Import two packages: example.com/greetings and the fmt package. This gives your code access to functions in those packages. Importing example.com/greetings (the package contained in the module you created earlier) gives you access to the Hello function. You also import fmt, with functions for handling input and output text (such as printing text to the console).
- Get a greeting by calling the greetings package's Hello function.
- 5. Edit the example.com/hello module to use your local example.com/greetings module.

For production use, you'd publish the example.com/greetings module from its repository (with a module path that reflected its published location), where Go tools could find it to download it. For now, because you haven't published the module yet, you need to adapt the example.com/hello module so it can find the example.com/greetings code on your local file system.

To do that, use the go mod edit command to edit the example.com/hello module to redirect Go tools from its module path (where the module isn't) to the local directory (where it is).

1. From the command prompt in the hello directory, run the following command:

```
$ go mod edit -replace example.com/greetings=../greetings
```

The command specifies that example.com/greetings should be replaced with ../greetings for the purpose of locating the dependency. After you run the command, the go.mod file in the hello directory should include a replace directive:

```
module example.com/hello
go 1.16
replace example.com/greetings => ../greetings
```

2. From the command prompt in the hello directory, run the go mod tidy command to synchronize the example.com/hello module's dependencies, adding those required by the code, but not yet tracked in the module.

```
$ go mod tidy
go: found example.com/greetings in example.com/greetings v0.0.0-00010101000000-000000000000
```

After the command completes, the example.com/hello module's go.mod file should look like this:

The command found the local code in the greetings directory, then added a require directive to specify that example.com/hello requires example.com/greetings. You created this dependency when you imported the greetings package in hello.go.

The number following the module path is a *pseudo-version number* -- a generated number used in place of a semantic version number (which the module doesn't have yet).

To reference a *published* module, a go.mod file would typically omit the replace directive and use a require directive with a tagged version number at the end.

require example.com/greetings v1.1.0

For more on version numbers, see Module version numbering.

6. At the command prompt in the hello directory, run your code to confirm that it works.

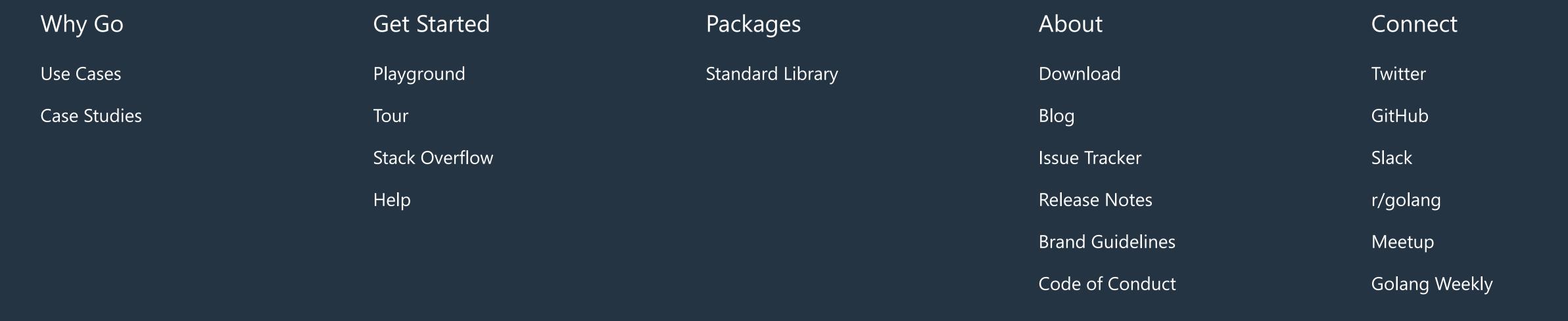
```
Hi, Gladys. Welcome!
```

\$ go run .

Congrats! You've written two functioning modules.

In the next topic, you'll add some error handling.

< Create a Go module</p>
Return and handle an error >





Copyright