# Return and handle an error

Handling errors is an essential feature of solid code. In this section, you'll add a bit of code to return an error from the greetings module, then handle it in the caller.

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

1. In greetings/greetings.go, add the code highlighted below.

   There's no sense sending a greeting back if you don't know who to greet. Return an error to the caller if the name is empty. Copy the following code into greetings.go and save the file.

```
package greetings

import (
    "errors"
    "fmt"
)

// Hello returns a greeting for the named person.
func Hello(name string) (string, error) {
    // If no name was given, return an error with a message.
    if name == "" {
        return "", errors.New("empty name")
    }

    // If a name was received, return a value that embeds the name
    // in a greeting message.
    message := fmt.Sprintf("Hi, %v. Welcome!", name)
    return message, nil
}
```

   In this code, you:

   - Change the function so that it returns two values: a `string` and an `error`. Your caller will check the second value to see if an error occurred. (Any Go function can return multiple values. For more, see Effective Go.)
   - Import the Go standard library `errors` package so you can use its `errors.New` function.
   - Add an `if` statement to check for an invalid request (an empty string where the name should be) and return an error if the request is invalid. The `errors.New` function returns an `error` with your message inside.
   - Add `nil` (meaning no error) as a second value in the successful return. That way, the caller can see that the function succeeded.

2. In your hello/hello.go file, handle the error now returned by the `Hello` function, along with the non-error value.

   Paste the following code into hello.go.

```
package main

import (
    "fmt"
    "log"

    "example.com/greetings"
)

func main() {
    // Set properties of the predefined Logger, including
    // the log entry prefix and a flag to disable printing
    // the time, source file, and line number.
    log.SetPrefix("greetings: ")
    log.SetFlags(0)

    // Request a greeting message.
    message, err := greetings.Hello("")
    // If an error was returned, print it to the console and
    // exit the program.
    if err != nil {
        log.Fatal(err)
    }

    // If no error was returned, print the returned message
    // to the console.
    fmt.Println(message)
}
```

   In this code, you:

   - Configure the `log` package to print the command name ("greetings: ") at the start of its log messages, without a time stamp or source file information.
   - Assign both of the `Hello` return values, including the `error`, to variables.
   - Change the `Hello` argument from Gladys's name to an empty string, so you can try out your error-handling code.
   - Look for a non-nil `error` value. There's no sense continuing in this case.
   - Use the functions in the standard library's `log` package to output error information. If you get an error, you use the `log` package's `Fatal` function to print the error and stop the program.

3. At the command line in the `hello` directory, run hello.go to confirm that the code works.

   Now that you're passing in an empty name, you'll get an error.

```
$ go run .
greetings: empty name
exit status 1
```

That's common error handling in Go: Return an error as a value so the caller can check for it.

Next, you'll use a Go slice to return a randomly-selected greeting.

**< Call your code from another module** **Return a random greeting >**