Level 2–3

Underneath the Tracks

Working With Errors

ON TRACK with GOLANG

Don’t Wake the Gophers Up!

This program should not be allowed to run before 7AM. In case this happens, it should terminate immediately and return an **error code**.

AM 8:00

```
$ go run main.go
```

```
Good Morning
```

PM 2:00

```
$ go run main.go
```

```
Good Afternoon
```

PM 7:00

```
$ go run main.go
```

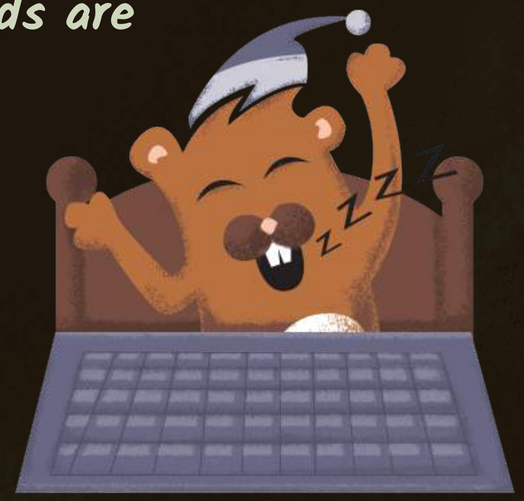
```
Good Evening
```

AM 6:00

```
$ go run main.go
```

```
Too early for greetings!
```

Our gopher friends are still asleep...



Declaring Multiple Return Values

In Go, we communicate errors via a **separate return value**. Let's update our function signature to return **two different values**: a string and an error.

src/hello/main.go

```
...

func getGreeting(hour int) (string, error) {

  if hour < 7 {


  }

  ...
}
```

Two values will now be returned from this function.

If it's earlier than 7AM, this block will be executed.

Returning With Error

If invoked before 7AM, the getGreeting() function will return an empty string **and a new error.**

```
...
import (
  "errors"
  ...
)

func getGreeting(hour int) (string, error) {
  var message string

  if hour < 7 {
    err := errors.New("Too early for greetings!")
    return message, err
  }
  ...
}
```

Import package from standard library.

Manually declaring the variable and data type.

Assigning a new error and returning it alongside an empty string message

Has not been assigned a value at this point, so defaults to zero value of empty string

Zero Values

A zero value in Go is the **default value** assigned to variables declared without an explicit initial value.

```
var message string
fmt.Print(message)
```

→ `""`

```
var age int
fmt.Print(age)
```

→ `0`

```
var isAdmin bool
fmt.Print(isAdmin)
```

→ `false`

http://go.codeschool.com/go-zero-value

Type	Zero Value
float	`0.0`
byte	`0`
function	`nil`
etc...	

Every primitive data type has an associated zero value to it.

Assigning a Message

We determine the appropriate greeting and assign it to the previously declared message variable.

```
...

func getGreeting(hour int) (string, error) {


   ...
   if hour < 7 { ... }

   if hour < 12 {
      message = "Good Morning"
   } else if hour < 18 {
      message = "Good Afternoon"
   } else {
      message = "Good Evening"
   }

}
```

Using = to assign a value because variable was manually declared previously

Returning With No Error

We use an **explicit nil** as the second return value. This indicates the function ran with no errors.

```
...

func getGreeting(hour int) (string, error) {
  ...
  if hour < 7 { ... }

  if hour < 12 {
    message = "Good Morning"
  } else if hour < 18 {
    message = "Good Afternoon"
  } else {
    message = "Good Evening"
  }
  return message, nil
}
```

A nil value for error tells the caller this function has no error.

Reading Multiple Values From a Function

We can assign multiple values at once by separating the new variables using a comma.

```
...

func main() {
  hourOfDay := time.Now().Hour()
  greeting, err := getGreeting(hourOfDay)

  fmt.Print(greeting)
}

func getGreeting(hour int) (string, error) {
  ...
}
```

Two values are now being returned from `getGreeting()`.

Checking for Errors

It is a common practice in Go to always check if an error exists before proceeding.

```
...

func main() {
  hourOfDay := time.Now().Hour()
  greeting, err := getGreeting(hourOfDay)
  if err != nil {



  }
  fmt.Print(greeting)
}

func getGreeting(hour int) (string, error) {
  ...
}
```

If err is NOT nil, then some error must have occurred!

Exiting With Error

The exit code 1 is a **POSIX standard** indicating the program has finished, but **errors have occurred.**

```
import (
  "os"
  ...
)

func main() {
  hourOfDay := time.Now().Hour()
  greeting, err := getGreeting(hourOfDay)
  if err != nil {
    fmt.Println(err)
    os.Exit(1)
  }
  fmt.Print(greeting)
}
...
```

We import our old friend, the os package.

Prints error to the console.

The Exit function takes an exit code of type int as argument and causes the program to terminate immediately.

Running the Complete Code

If we run the code now, it still prints all messages just like before and also an error message if it's before 7AM.

AM 8:00

```
$ go run main.go
```

```
Good Morning
```

PM 2:00

```
$ go run main.go
```

```
Good Afternoon
```

PM 7:00

```
$ go run main.go
```

```
Good Evening
```

AM 6:00

```
$ go run main.go
```

```
Too early for greetings!
```

Our gopher friends can now sleep in peace.

Where is the **exit code**?

Exit codes are used by **other programs** so they know whether or not an error occurred.

(Remember systems programming?)