Testing: A Concrete Example

This lesson brings an application to test according to the concepts learned in the last lesson.

Here is a concrete example for you to run and learn to test:

Environment Variables

Key:	Value:
GOROOT	/usr/local/go
GOPATH	//root/usr/local/go/src
PATH	//root/usr/local/go/src/bin:/usr/local/go...

```
package even
import "testing"

func TestEven(t *testing.T) {
    if !Even(10) {
        t.Log(" 10 must be even!")
        t.Fail()
    }
    if Even(7) {
        t.Log(" 7 is not even!")
        t.Fail()
    }
}

func TestOdd(t *testing.T) {
    if !Odd(11) {
        t.Log(" 11 must be odd!")
        t.Fail()
    }
    if Odd(10) {
        t.Log(" 10 is not odd!")
        t.Fail()
    }
}
```

This is our test program: it imports the `even` package at **line 4**, which will contain the functional logic and the tests. Then, we have for-loop at **line 8** controlling **100** iterations. It shows the first 100 integers and whether they are even or not (see **line 9**).

The package `even` in the subfolder `even` contains the logic in the two functions `Even` and `Odd`. The `Even` function returns *true* if an integer `i` passed to it is even (`i%2 == 0`). The `Odd` function returns *true* if an integer `i` passed to it is not even (`i%2 != 0`).

The file **oddeven_test.go** contains the tests. It belongs to the package `even` (**line 1**), so it can use its logic. It needs the specific functionality from the package `testing`, so this is imported at **line 2**. It contains 2 functions `TestEven` and `TestOdd`, both complying with the requirements for a test function, and it must have a parameter `t` of type `*testing.T`, which contains the `Log()` and `Fail()` methods. Then, we simply test out a few examples of which we know the outcome. We use `!` here and use only *failing* examples. For example:

- Look at **line 5**. We are using `!Even()` and testing it for **10**, which would definitely fail because **10** is an *even* number.
- Look at **line 9**. We are using `Even()` and testing it for 7, which would definitely fail because 7 is not an *even* number.
- Look at **line 16**. We are using `!Odd()` and testing it for **11**, which would definitely fail because as **11** is an *odd* number.
- Look at **line 20**. We are using `Odd()` and testing it for **10**, which would definitely fail because as **10** is not an *odd* number.

In a more real and complex case, you should also test the correct outcome and fringe cases.

Because testing uses concrete cases of input and we can never test all cases (most likely there is an infinite number), we must give some thought to the test cases we are going to use.

We should at least include:

- A normal case
- Abnormal cases (wrong input like negative numbers or letters instead of numbers, no input)
- Boundary cases (if a parameter has to have a value in the interval 0-1000, then check 0 and 1000)

We can now test our package with the command `go test -v package1/package1` from within the folder:

```
$GOPATH/src/package1.
Output:      === RUN TestEven

             --- PASS: TestEven (0.00 seconds)
             === RUN TestOdd
             --- PASS: TestOdd (0.00 seconds)
             PASS
             ok even/even 0.217s
```

Because the test-functions we saw in Error-Handling and Panicking in a User-Defined Package do not invoke `t.Log` or `t.Fail`, this gives us as result: **PASS**. In this simple example, everything works fine.

To see output in case of a failure, change the function `TestEven` to:

```go
func TestEven(t *testing.T) {
  if Even(10) {
    t.Log("Everything OK: 10 is even, just a test to see failed output!")
    t.Fail()
  }
}
```

`Even(10)` now invokes `t.Log` and `t.Fail`, giving us as a result:

```
--- FAIL: even.TestEven (0.00 seconds)
Everything OK: 10 is even, just a test to see failed output!
FAIL
```

The file **main.go** gets built with: `go build main.go`.

To run the program, invoke `main` on the command-line, with the output:

```
>main
Is the integer 0 even? true
Is the integer 1 even? false
Is the integer 2 even? true
Is the integer 3 even? false
Is the integer 4 even? True
...
```

For another concrete example of testing and benchmarking, see Chapter 12 where benchmarking is performed on an example with Goroutines. In the next lesson, you'll learn a technique that makes the testing systematic.