BOLDLY GO

THE TEST GENERATION

Or: How to get started testing in Go

May 2, 2023
Golang Amsterdam Meetup

Agenda

- Who am I?
- Why test?
- Write your first test
- A more involved test
- Asserting
- Multiple tests
- For further reading

Who Am I?

M-559

14

23

25

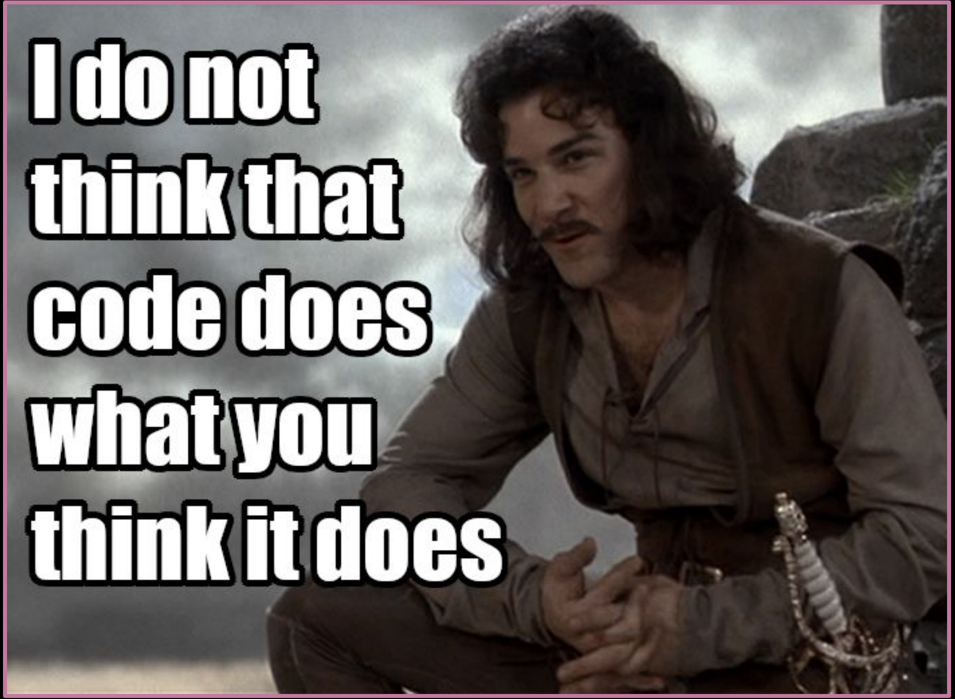
Independent Go Dev, “YouTuber” (I hate that moniker), Podcaster, Blogger, etc

https://boldlygo.tech/
https://youtube.com/@BoldlyGo

Jonathan Hall

Why Test?

- Validate Expectations
- No Unwanted Drift
- Executable Docs
- Many more reasons



Writing your first test

M-559

14

23

25

main.go

```go
package main

import "fmt"

func main() {
    fmt.Println("Hello, World!")
}
```

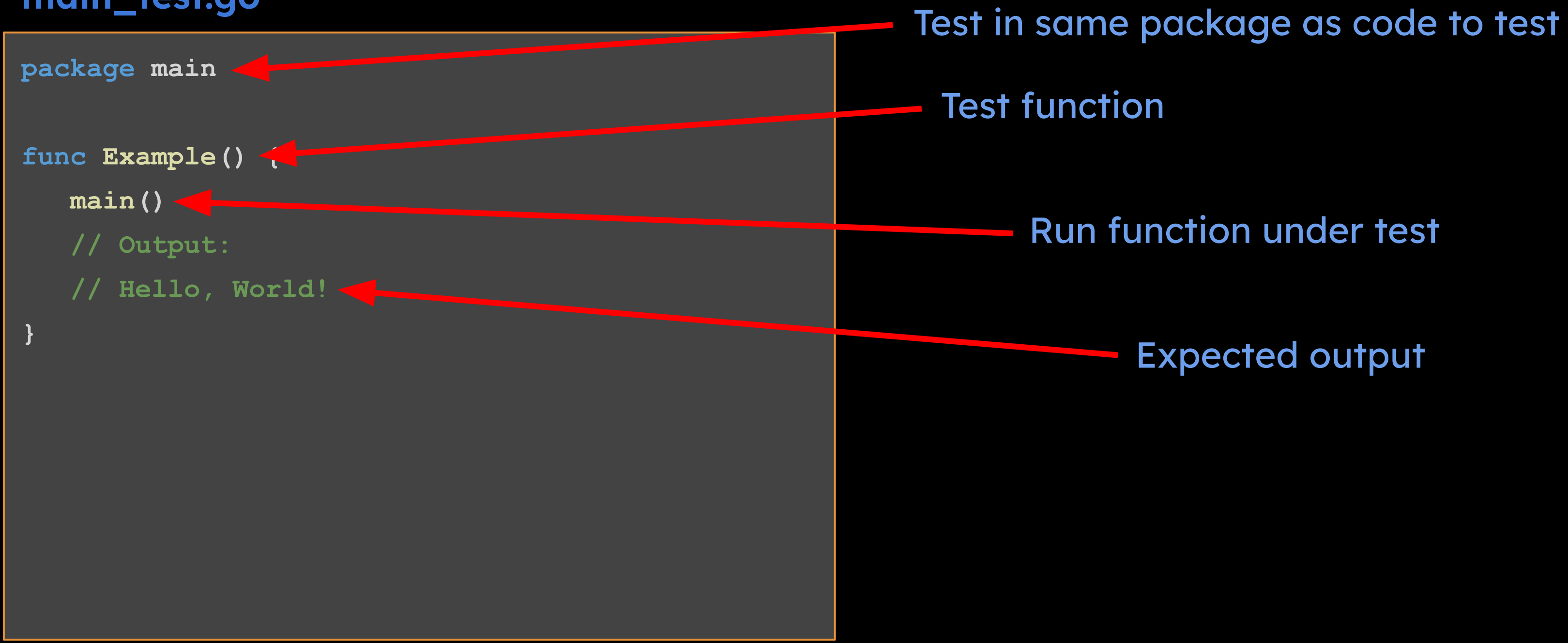
main_test.go

```go
package main

func Example() {
    main()
    // Output:
    // Hello, World!
}
```

```
$ go test -v
=== RUN   Example
--- PASS: Example (0.00s)
PASS
ok      foo     0.001s
```

Writing your first test

main_test.go



```go
package main

func Example() {
    main()
    // Output:
    // Hello, World!
}
```

Test in same package as code to test

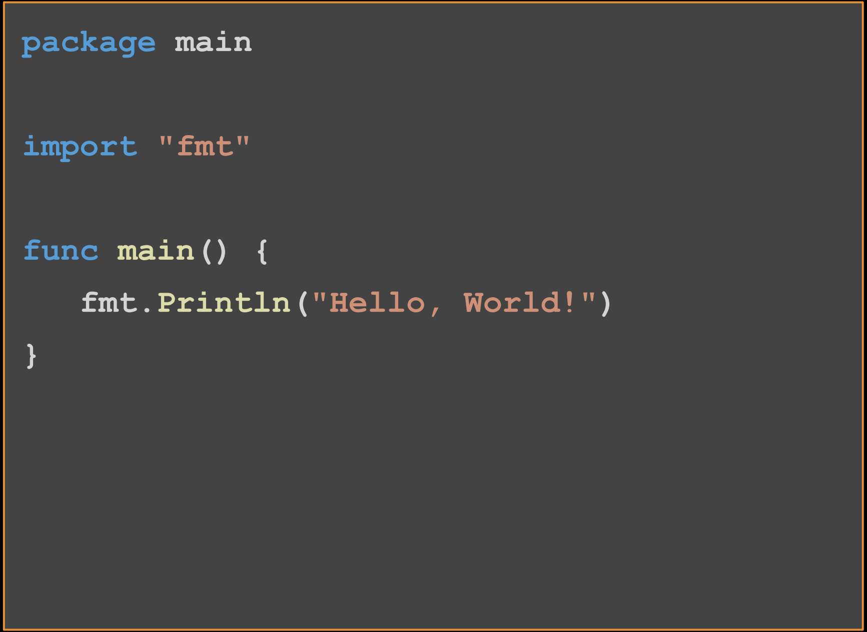
Test function

Run function under test

Expected output

Writing your first test

main.go



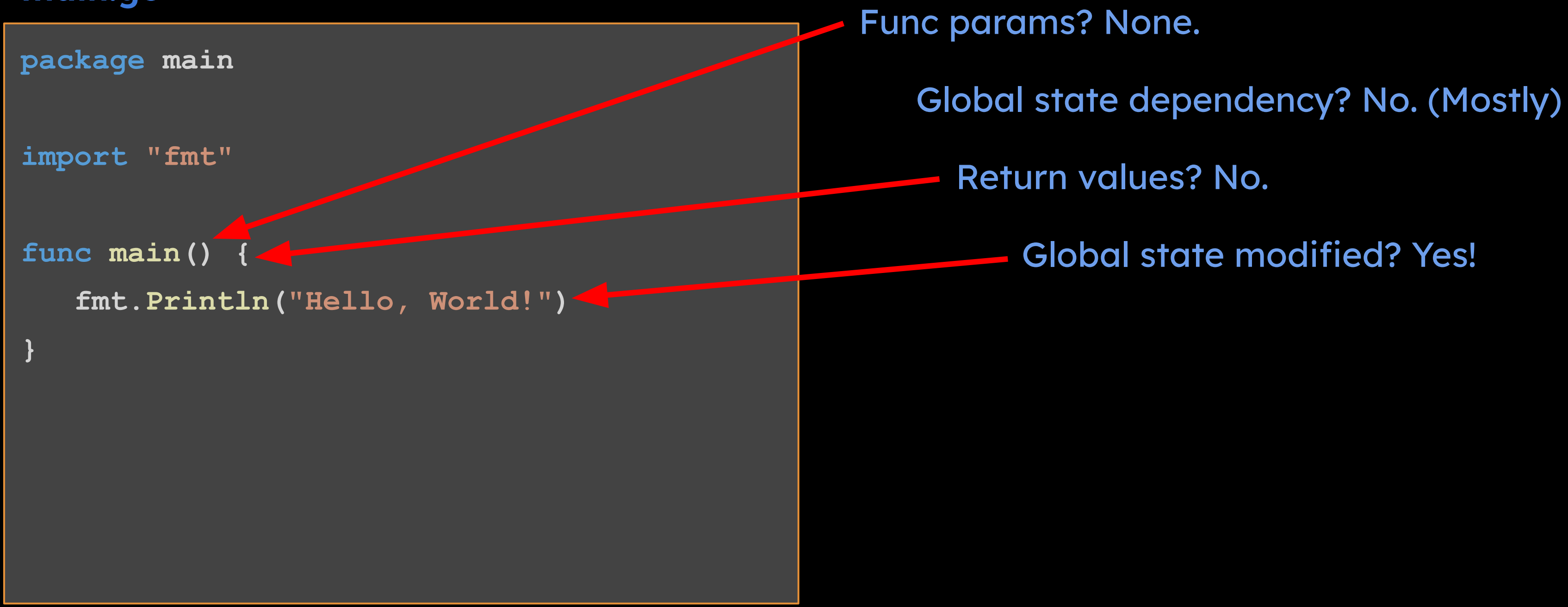
```go
package main

import "fmt"

func main() {
   fmt.Println("Hello, World!")
}
```

What does this code do?

- What are the inputs?
  - Func params
  - Global state
- What are the outputs?
  - Return values
  - Global state/side effects

Writing your first test

main.go



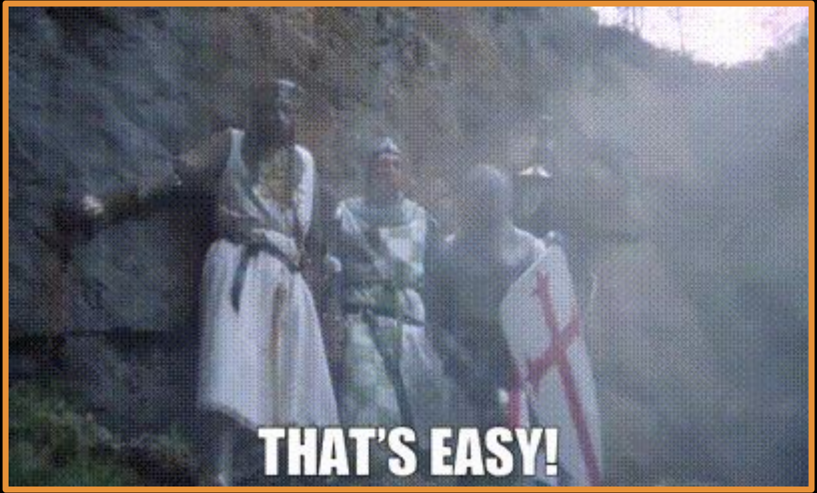
```go
package main

import "fmt"

func main() {
    fmt.Println("Hello, World!")
}
```

Func params? None.

Global state dependency? No. (Mostly)

Return values? No.

Global state modified? Yes!



Most functions aren’t so trivial

Therefore, most tests aren’t so trivial

A more involved example

fib.go

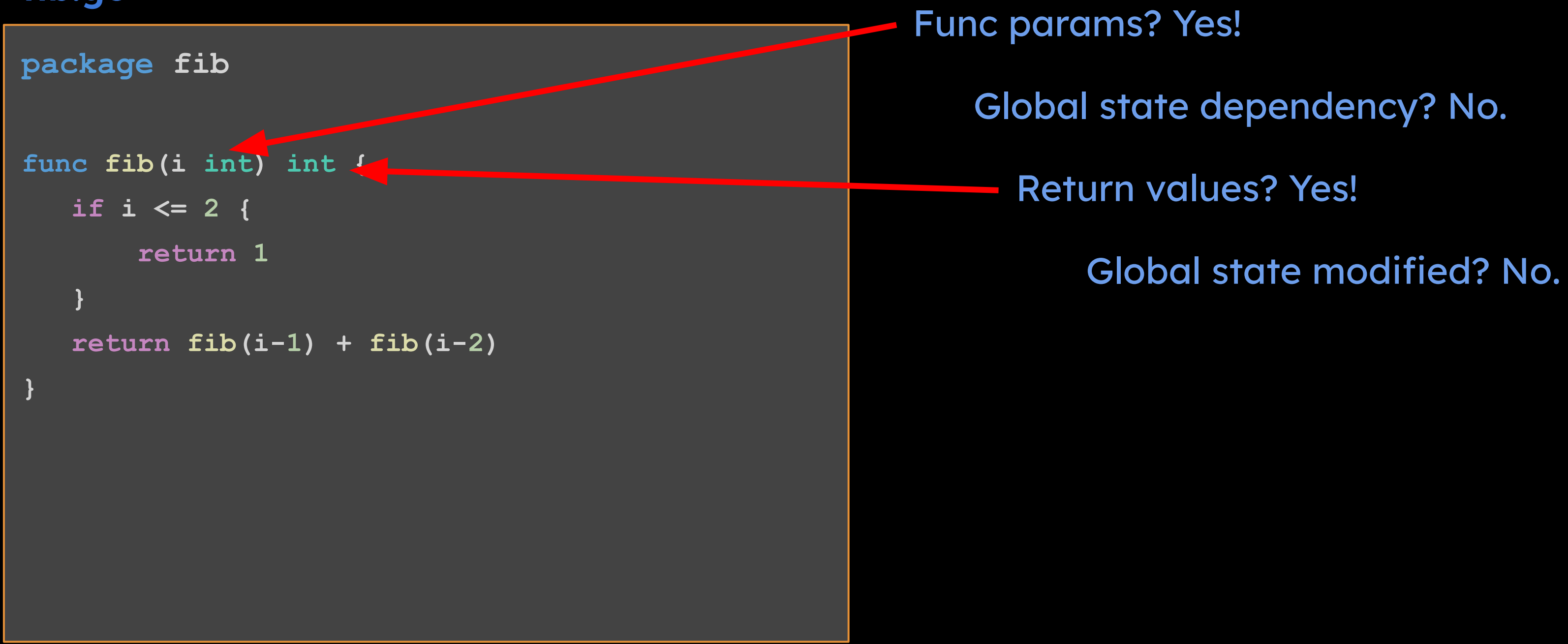
```go
package fib


func fib(i int) int {
   if i <= 2 {
       return 1
   }
   return fib(i-1) + fib(i-2)
}
```

What does this code do?

- What are the inputs?
  - Func params
  - Global state
- What are the outputs?
  - Return values
  - Global state/side effects

A more involved example

fib.go



```go
package fib

func fib(i int) int {
    if i <= 2 {
        return 1
    }
    return fib(i-1) + fib(i-2)
}
```

Func params? Yes!

Global state dependency? No.

Return values? Yes!

Global state modified? No.

A more involved example

fib_test.go

```go
package fib

import "testing"

func Test_fib(t *testing.T) {
    t.Parallel()
    want := 1
    got := fib(1)
    if want != got {
        t.Errorf("Unexpected: %v", got)
    }
}
```

Test in same package as code to test

Import `testing` package

Allow running tests in parallel

Run function under test

Validate result

```
$ go test -v
=== RUN   Test_fib
=== PAUSE Test_fib
=== CONT  Test_fib
--- PASS: Test_fib (0.00s)
PASS
ok      foo     0.001s
```

Don’t Assert

Don’t Assert

“Avoid the use of 'assert' libraries to help your tests.”

- Go Wiki

“… our experience has been that programmers use them as a crutch to avoid thinking about proper error handling and reporting.”

- Go FAQ

If you must…

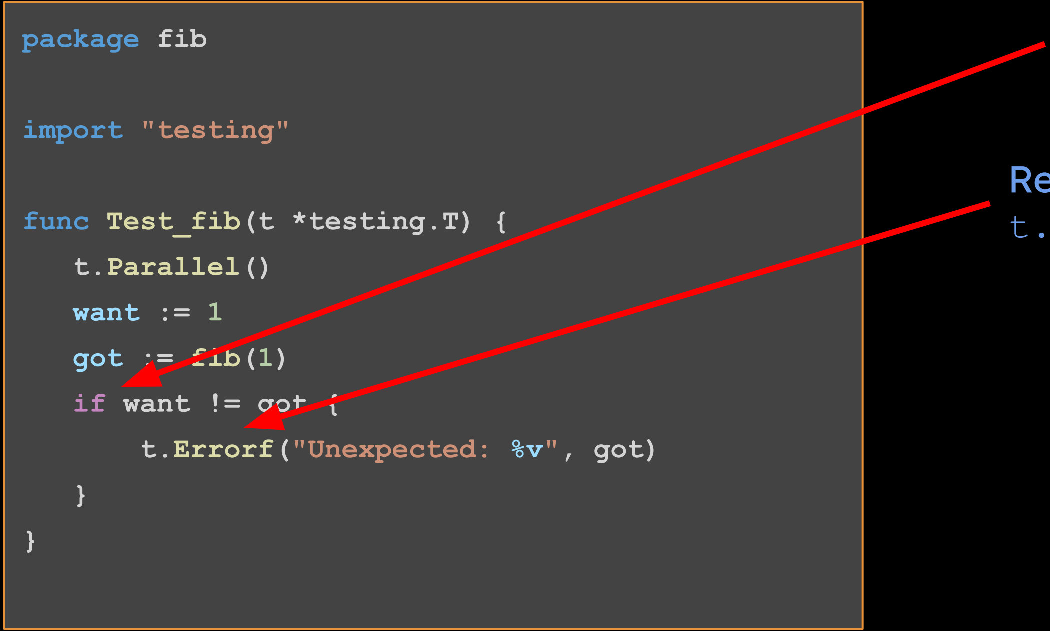
The only Go assertion library you'll ever need
100% ALL NATURAL INGREDIENTS
assert( )
NOW with MORE!

PARODY ALERT

gitlab.com/flimzy/assert

Don’t Assert

fib_test.go



```go
package fib

import "testing"

func Test_fib(t *testing.T) {
    t.Parallel()
    want := 1
    got := fib(1)
    if want != got {
        t.Errorf("Unexpected: %v", got)
    }
}
```

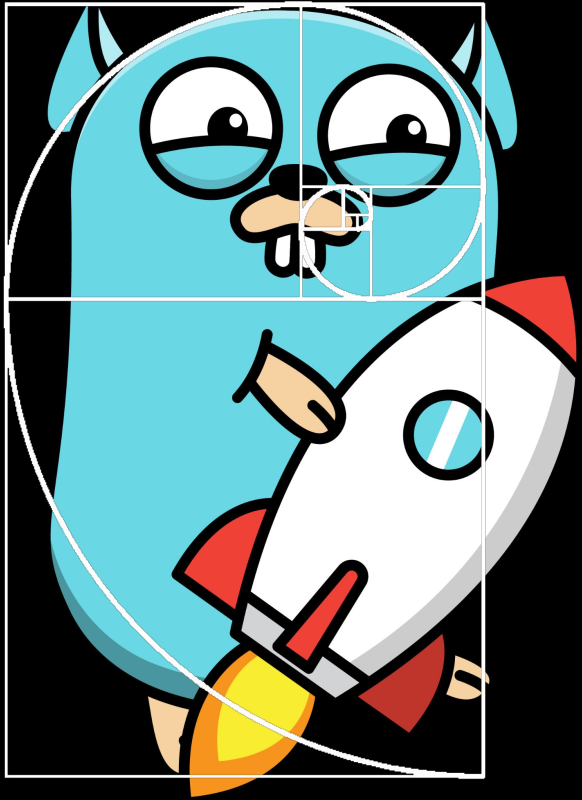
Standard conditional checking

Report failure with `t.Error`, `t.Errorf`, `t.Fail`, etc.

Multiple Tests

Three approaches:

- Top-level func per check
- Many checks per func
- Sub-tests
  - Table-driven tests



Multiple Checks Per Func

fib_test.go

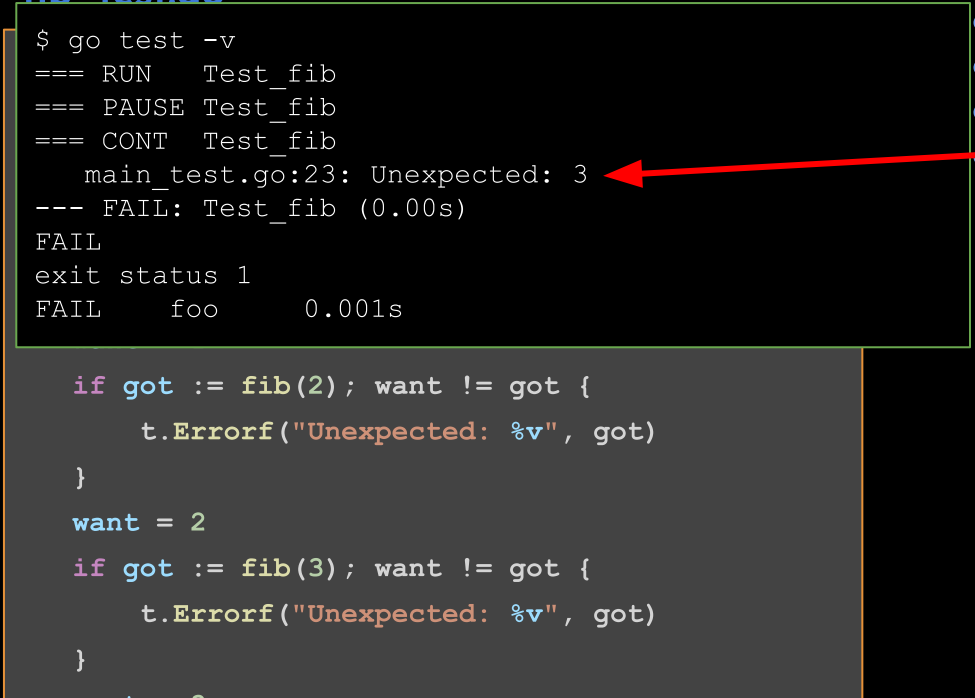
```go
func Test_fib(t *testing.T) {
    t.Parallel()
    want := 1
    if got := fib(1); want != got {
        t.Errorf("Unexpected: %v", got)
    }
    want = 1
    if got := fib(2); want != got {
        t.Errorf("Unexpected: %v", got)
    }
    want = 2
    if got := fib(3); want != got {
        t.Errorf("Unexpected: %v", got)
    }
```

Multiple checks in same func

```
$ go test -v
=== RUN   Test_fib
=== PAUSE Test_fib
=== CONT  Test_fib
--- PASS: Test_fib (0.00s)
PASS
ok      foo     0.001s
```

Multiple Checks Per Func

fib_test.go



```
$ go test -v
=== RUN   Test_fib
=== PAUSE Test_fib
=== CONT  Test_fib
   main_test.go:23: Unexpected: 3
--- FAIL: Test_fib (0.00s)
FAIL
exit status 1
FAIL    foo     0.001s
```

```go
	if got := fib(2); want != got {
		t.Errorf("Unexpected: %v", got)
	}
	want = 2
	if got := fib(3); want != got {
		t.Errorf("Unexpected: %v", got)
	}
```

- Hard to write & modify
- Hard to read
- No parallel execution
- Very difficult to debug

Sub-Tests

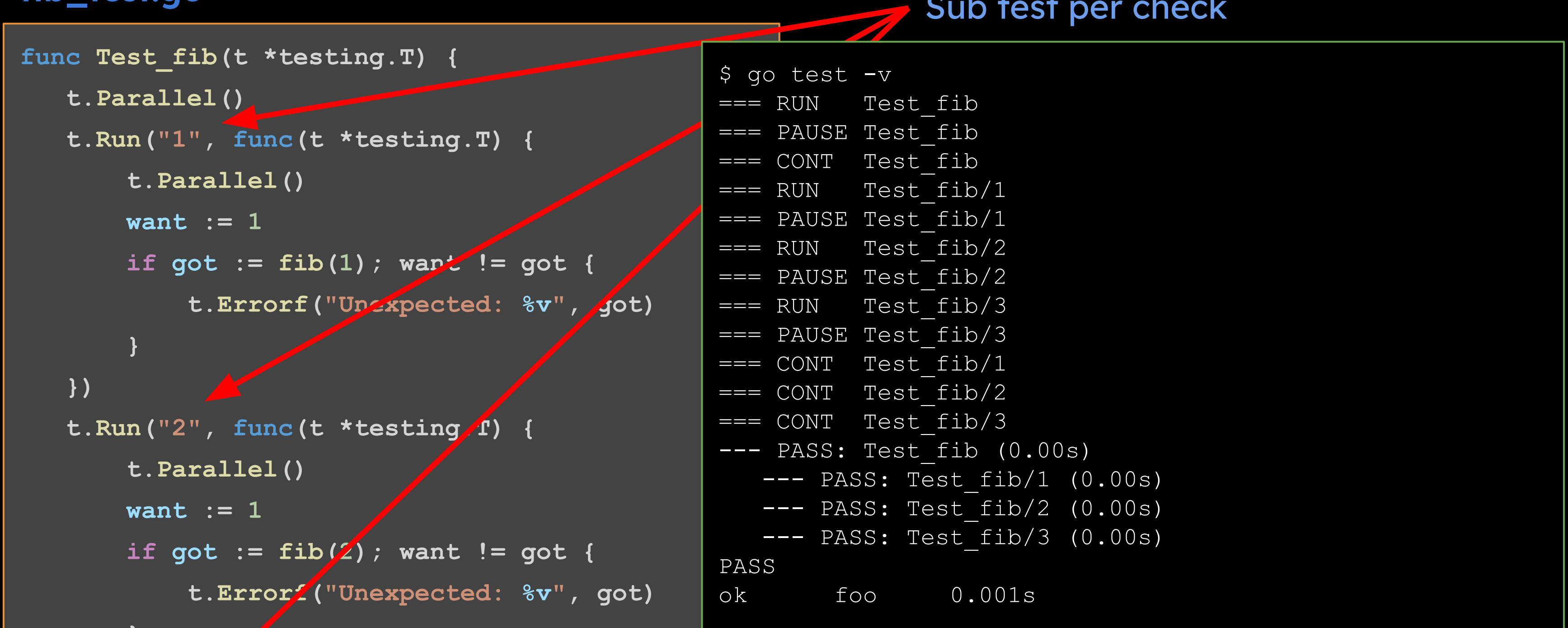
M-559

14

23

25

fib_test.go



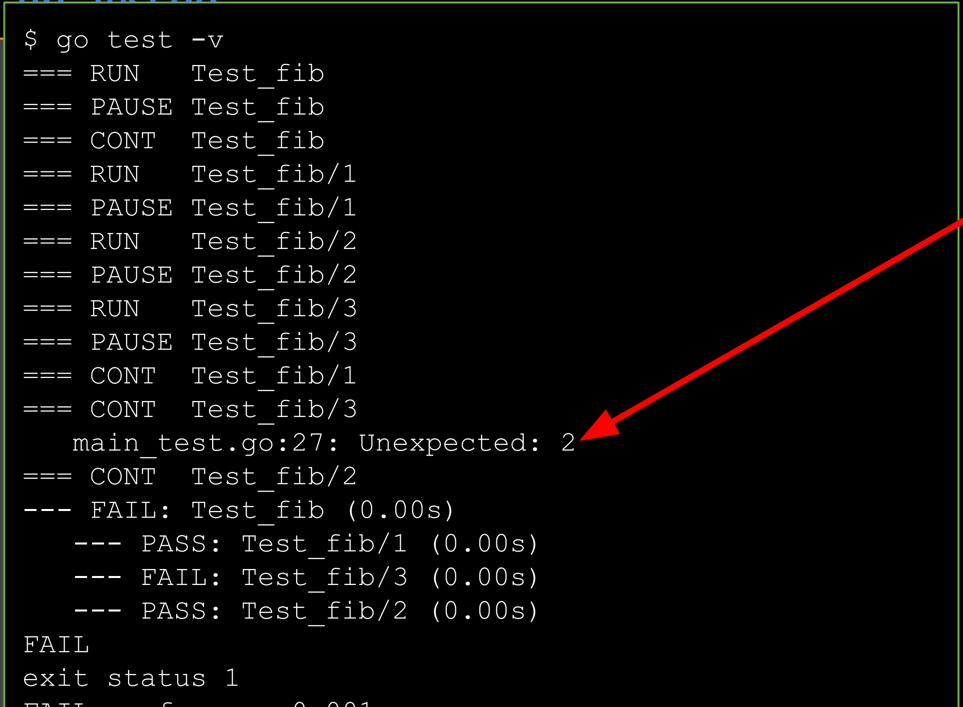
Sub test per check

```go
func Test_fib(t *testing.T) {
    t.Parallel()
    t.Run("1", func(t *testing.T) {
        t.Parallel()
        want := 1
        if got := fib(1); want != got {
            t.Errorf("Unexpected: %v", got)
        }
    })
    t.Run("2", func(t *testing.T) {
        t.Parallel()
        want := 1
        if got := fib(2); want != got {
            t.Errorf("Unexpected: %v", got)
```

```
$ go test -v
=== RUN   Test_fib
=== PAUSE Test_fib
=== CONT  Test_fib
=== RUN   Test_fib/1
=== PAUSE Test_fib/1
=== RUN   Test_fib/2
=== PAUSE Test_fib/2
=== RUN   Test_fib/3
=== PAUSE Test_fib/3
=== CONT  Test_fib/1
=== CONT  Test_fib/2
=== CONT  Test_fib/3
--- PASS: Test_fib (0.00s)
    --- PASS: Test_fib/1 (0.00s)
    --- PASS: Test_fib/2 (0.00s)
    --- PASS: Test_fib/3 (0.00s)
PASS
ok      foo     0.001s
```

Sub-Tests

fib_test.go



```
$ go test -v
=== RUN   Test_fib
=== PAUSE Test_fib
=== CONT  Test_fib
=== RUN   Test_fib/1
=== PAUSE Test_fib/1
=== RUN   Test_fib/2
=== PAUSE Test_fib/2
=== RUN   Test_fib/3
=== PAUSE Test_fib/3
=== CONT  Test_fib/1
=== CONT  Test_fib/3
   main_test.go:27: Unexpected: 2
=== CONT  Test_fib/2
--- FAIL: Test_fib (0.00s)
   --- PASS: Test_fib/1 (0.00s)
   --- FAIL: Test_fib/3 (0.00s)
   --- PASS: Test_fib/2 (0.00s)
FAIL
exit status 1
```

- More verbose/more typing
- Easy to modify/add new cases
- Easier to read
- Parallel execution
- Easier to debug

Table-Driven Tests

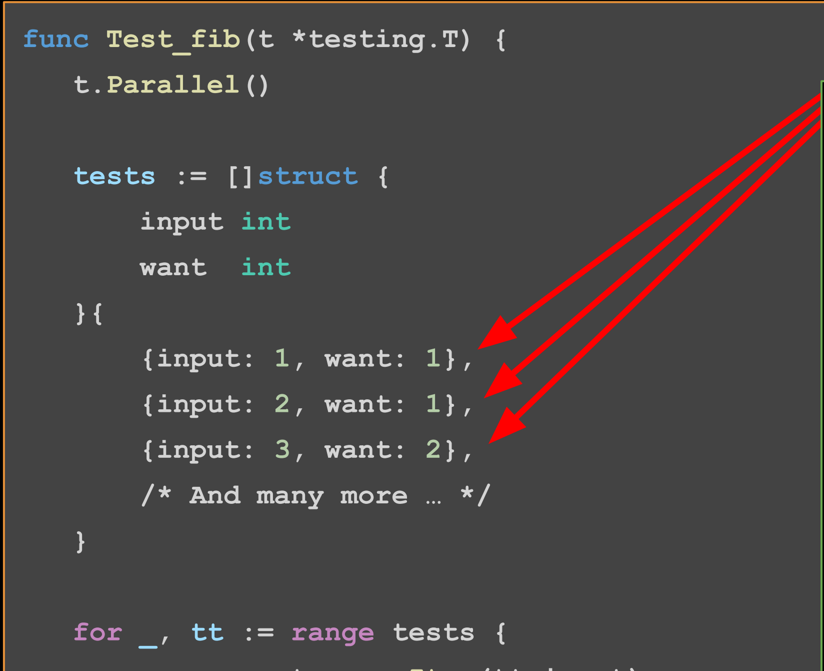
fib_test.go

```go
func Test_fib(t *testing.T) {
   /* snip */
   for _, tt := range tests {
       name := strconv.Itoa(tt.input)
       t.Run(name, func(t *testing.T) {
           t.Parallel()
           got := fib(tt.input)
           if got != tt.want {
               t.Errorf("Unexpected: %v", got)
           }
       })
   }
}
```

A single function that executes multiple test cases from the slice `tests`.

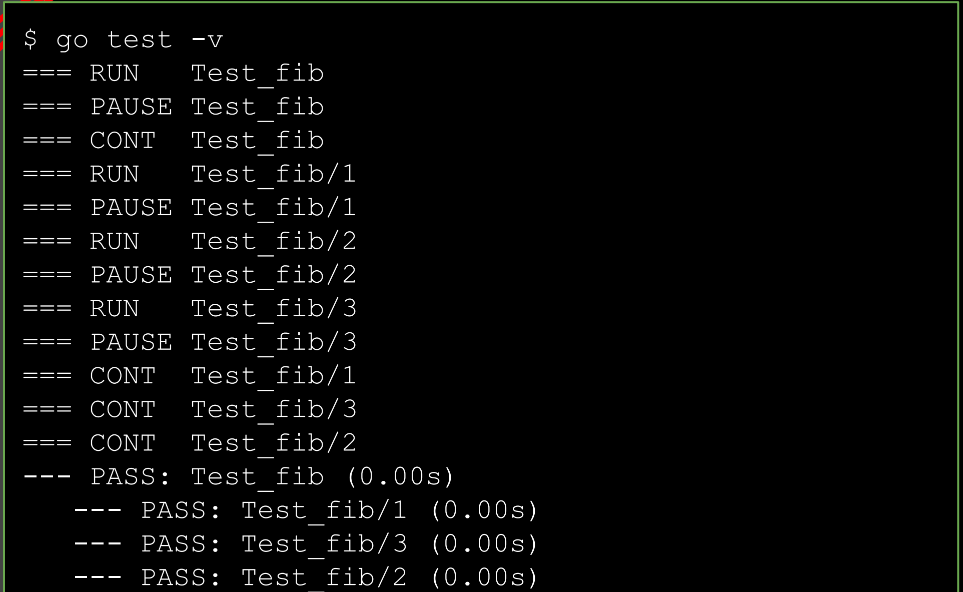
Table-Driven Tests

fib_test.go



```go
func Test_fib(t *testing.T) {
   t.Parallel()

   tests := []struct {
       input int
       want  int
   }{
       {input: 1, want: 1},
       {input: 2, want: 1},
       {input: 3, want: 2},
       /* And many more … */
   }

   for _, tt := range tests {
```

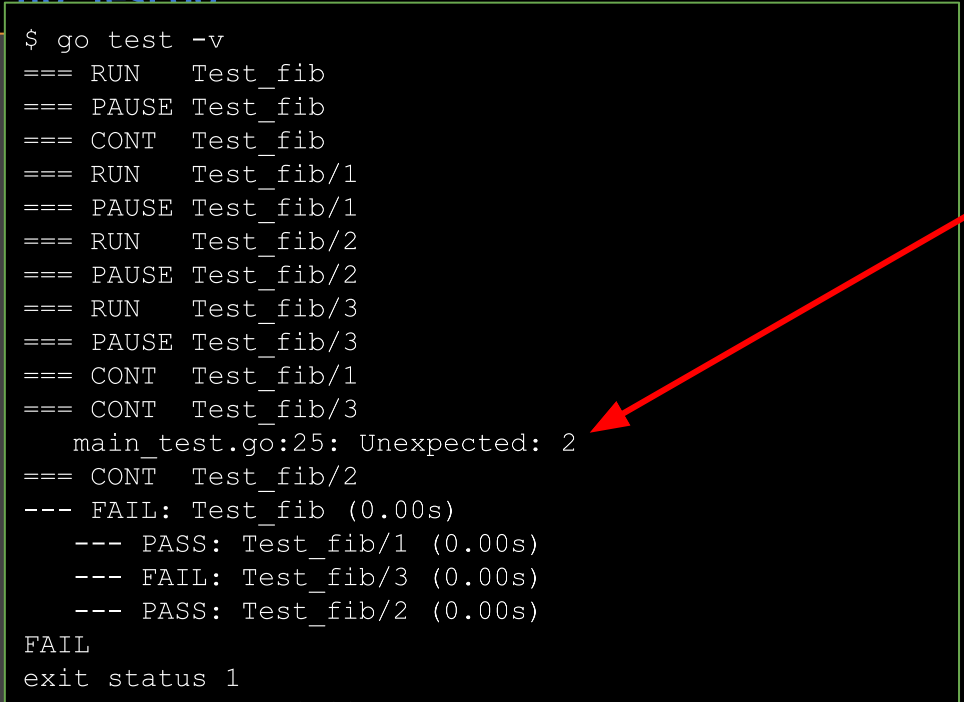
Tests slice can contain an arbitrary number of test cases.



```
$ go test -v
=== RUN   Test_fib
=== PAUSE Test_fib
=== CONT  Test_fib
=== RUN   Test_fib/1
=== PAUSE Test_fib/1
=== RUN   Test_fib/2
=== PAUSE Test_fib/2
=== RUN   Test_fib/3
=== PAUSE Test_fib/3
=== CONT  Test_fib/1
=== CONT  Test_fib/3
=== CONT  Test_fib/2
--- PASS: Test_fib (0.00s)
    --- PASS: Test_fib/1 (0.00s)
    --- PASS: Test_fib/3 (0.00s)
    --- PASS: Test_fib/2 (0.00s)
```

Table-Driven Tests

fib_test.go



```
$ go test -v
=== RUN   Test_fib
=== PAUSE Test_fib
=== CONT  Test_fib
=== RUN   Test_fib/1
=== PAUSE Test_fib/1
=== RUN   Test_fib/2
=== PAUSE Test_fib/2
=== RUN   Test_fib/3
=== PAUSE Test_fib/3
=== CONT  Test_fib/1
=== CONT  Test_fib/3
   main_test.go:25: Unexpected: 2
=== CONT  Test_fib/2
--- FAIL: Test_fib (0.00s)
   --- PASS: Test_fib/1 (0.00s)
   --- FAIL: Test_fib/3 (0.00s)
   --- PASS: Test_fib/2 (0.00s)
FAIL
exit status 1
```

- Less typing than alternatives
- Easy to modify/add new cases
- Easy to read
- Parallel execution
- Easy to debug

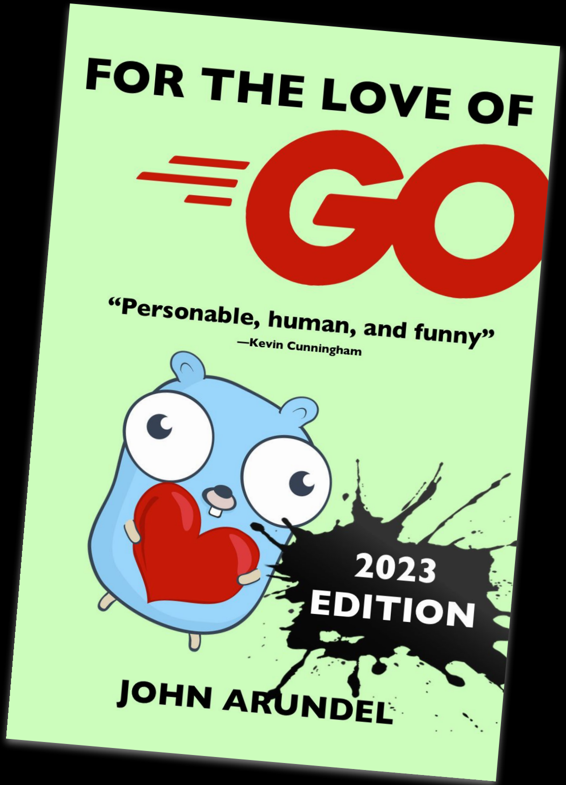
For Further Reading

For the Love of Go by John Arundel

(Read/Watch my review)

My upcoming course on testing in Go

https://boldlygo.tech/courses/



Thank You

Code Long and Prosper!

Questions?

BOLDLY GO

https://youtube.com/@BoldlyGo

https://boldlygo.tech/daily

https://cupogo.dev/