

# Testing

**01**

Setup

**02**

Examples

**03**

Test Tables

# Testing

- | Important to test software to prevent regressions and ensure it meets specifications
  - | Unit testing - test individual functions
  - | Integration testing - test functions/modules working together
- | Go makes no distinction between the two
  - | Same process to create both Unit & Integration test

# Setup

- | Tests are written in separate files, sharing the name of the file they are testing
  - | `importantPkg.go` → `importantPkg_test.go`
- | Unit tests should be in the same package
- | The **testing** package is used to create tests and must be imported in each test file

# Example: Function

| sample.go

```
package main

import "regexp"

func IsValidEmail(addr string) bool {
    re, ok := regexp.Compile(`.+@.+\..+`)
    if ok != nil {
        panic("failed to compile regex")
    } else {
        return re.Match([]byte(addr))
    }
}
```



# Example: Setup

| sample\_test.go

```
package main
```

```
import "testing"
```

```
func TestIsValidEmail(t *testing.T) {  
    data := "email@example.com"  
    if !IsValidEmail(data) {  
        t.Errorf("IsValidEmail(%v)=false, want true", data)  
    }  
}
```

# Example: Run Tests

- | Execute `go test` to run your tests

```
> go test
```

```
PASS
```

```
ok
```

```
coursecontent
```

```
0.001s
```

# Available Testing Functions

- | Many testing functions available in the **testing** package
- | **Fail()** - Mark the test as failed
  - | **Errorf(string)** - Fail & add a message
- | **FailNow()** - Mark the test as failed, abort current test
  - | **Fatalf(string)** - Fail, abort, and add a message
- | **Logf()** - Equivalent to **Printf**, but only when test fails

# Test Tables

- | Usually need to test more than one set of input data
- | **Test tables** can be used to accomplish this
  - | Similar to parameterized testing



# Test Tables: Example

```
func TestIsValidEmailTable(t *testing.T) {  
    table := []struct {  
        email string  
        want  bool  
    }{  
        {"email@example.com", true},  
        {"missing@tld", false},  
        // ...  
    }  
  
    for _, data := range table {  
        result := IsValidEmail(data.email)  
        if result != data.want {  
            t.Errorf("%v: %t, want: %t", data.email, result, data.want)  
        }  
    }  
}
```

# Test Tables: Running

```
> go test
--- FAIL: TestIsValidEmailTable (0.00s)
    lecture_test.go:33: contains spaces@example.com: true, want: false
    lecture_test.go:33: false@alarm.ok: true, want: false
FAIL
exit status 1
FAIL    coursecontent    0.001s
```

# Recap

- | Test files end with `_test`
- | Use the `testing` package to perform tests
  - | Tests are ran with the command `go test`
- | `Test tables` can be used to test multiple pieces of data
- | Use `t.Errorf` to report an error
- | Use `t.Fatalf` to report an error, and also abort the test case
- | Use `t.Logf` to display debug or test messages