Table of Contents

Learn Go with Tests

Or learn test-driven development with Go.

This book is hosted on github-pages.

Ebook can be downloaded as pdf, epub and mobi.

Why

- Explore the Go language by writing tests
- **Get a grounding with TDD**. Go is a good language for learning TDD because it is a simple language to learn and testing is built in
- Be confident that you'll be able to start writing robust, well tested systems in Go

Contributing

- *This book is a work in progress* If you would like to contribute, please do get in touch.
- Read CONTRIBUTING.md for guidelines
- Any ideas? Create an issue

Integers

Integers work as you would expect. Let's write an add function to try things out. Create a test file called `adder_test.go` and write this code.

Write the test first

```
package integers

import "testing"

func TestAdder(t *testing.T) {
    sum := Add(2, 2)
    expected := 4

    if sum != expected {
        t.Errorf("expected '%d' but got '%d'", expected, sum)
    }
}
```

Show the complete test

```
{% include "./adder_test.go" %}
```

Contributing

Contributions are very welcome, I hope for this to become a great home for guides of how to learn go by writing tests

What we're looking for

- Teaching Go features (e.g things like `if`, `select`, structs, methods, etc)
- Showcase interesting functionality within the standard library. Show off how easy it is to TDD a HTTP server for instance.
- Show how Go's tooling, like benchmarking, race detectors, etc can help you arrive at great software

If you don't feel confident to submit your own guide, submitting an issue for something you want to learn is still a valuable contribution.

Style guide

- Always reinforce the TDD cycle. Take a look at template.md
- Emphasis on iterating over functionality driven by tests. The Hello, world example works well because we gradually make it more sophisticated and learning new techniques *driven* by the tests. For example:
  - `Hello()` <- how to write functions, return types.
  - `Hello(name string)` <- arguments, constants
  - `Hello(name string)` <- default to "world" using `if`
  - `Hello(name, language string)` <- `switch`
- Try to minimise the surface area of required knowledge.
  - Thinking of examples that showcase what you're trying to teach without confusing the reader with other features is important.
  - For example, you can learn about `struct`s without understanding pointers.
  - Brevity is king.
- Follow the Code Review Comments style guide. It's important for a consistent style across all the sections.
- Your section should have a runnable application at the end (e.g `package main` with a `main` func) so users can see it in action and play with it
- All tests should pass

Publishing this book

Docker is used both for travis and for local builds. You can create the Docker image as follows:

```
docker build -t learn-go-with-tests .
```

NOTE The Docker solution is not using Volumes, its copying the contents into the image, so update files locally without rebuilding the image will not show the updates.

The book can be published to Github Pages with the following command:

```
docker run --rm -it learn-go-with-tests make publish -e GITHUB_TOKEN=xxxx
```