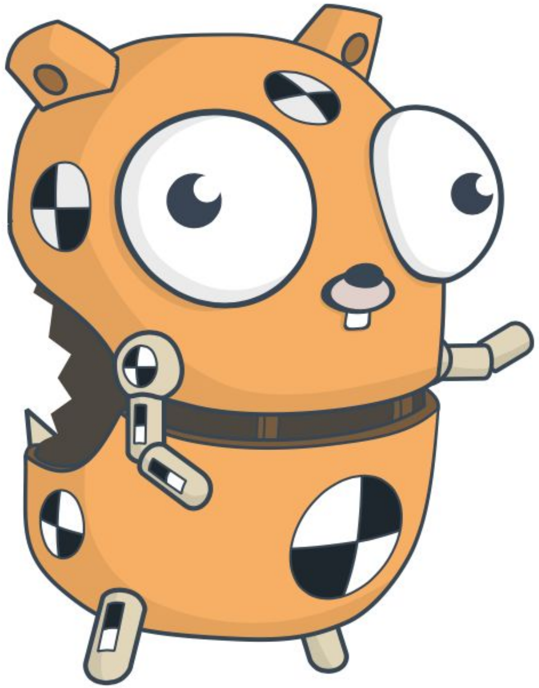
Testing in Go



Who am I?

Tyler Lugger

Backend Software Engineer at Bitly

Engineer: 5 years

Dog owner: 4 years

Writer of Go: 3 years

Coloradoan: 4ever

Why test your code?

Just don’t make bugs, right?

Importance of testing

- Not all software is perfect
- We frequently change and add features to very large established codebases
- Find problems before your users/customers
- It will be tested at some point
- Encourages good software design

Types of Software testing

- Manual vs Automated
- Functional vs Non-functional
    - Testing expected vs actual output of a function for a given input (functional)
    - Testing for how software operates rather than specific behaviors (non-functional)
- Unit vs Integration testing
    - Testing a single function/module
    - Testing one or more across a single user interaction

- Our focus: automated, functional unit and integration tests github.com/tlugger/testing-workshop

tylerkno.ws/testing

Testing frameworks

- Frequently built-in library in most languages
- Execute tests against your application and report results
- Defines the format to set expectations for function under test
    - Typically using code logic from the programming language
    - Common to use an assertion library for simplicity

```
if actual != expected {
    t.Fatal(“test case failed”)
}

assert.Equal(t, actual, expected, “test case failed”)
```

Testing in Go

package viewcounter

- “testing” package allows us to write tests in Go
- Test files must end with *_test.go* and all test functions must start with `Test`
- Test functions take a struct (usually `*testing.T`) to hold test state and format results

- Go supports table driven tests through subtests!
  - Tests are frequently set up with an array of parameters
  - Subtests then loop through and `Run` each test

- Run tests with the `go test` command

Test coverage

package pwdvalidator

- Measure of the percentage of source code tested
- Helps us find code that may be untested
- Ensures our tests cover all possible return points of a function

- Go measures coverage with built in tooling
- `go test -cover`

Mocking function calls in tests

package mapiss

- Unit tests are meant to test isolated behavior from our functions under test
- Those functions can have dependencies on other functions/packages/APIs
- Mocking allows us to avoid testing those dependencies

- Interface substitution is a common technique to achieve this
  - “Accept interfaces, return structs”
- Struct provides returned implementation by package
- Interface defines consumers expected implementation

Questions?

Keep in touch!

- Bitly Email: tyler.lugger@bit.ly
- Personal Email: notnottyler@gmail.com
- LinkedIn: https://tylerkno.ws/linkedin
- Github: https://github.com/tlugger