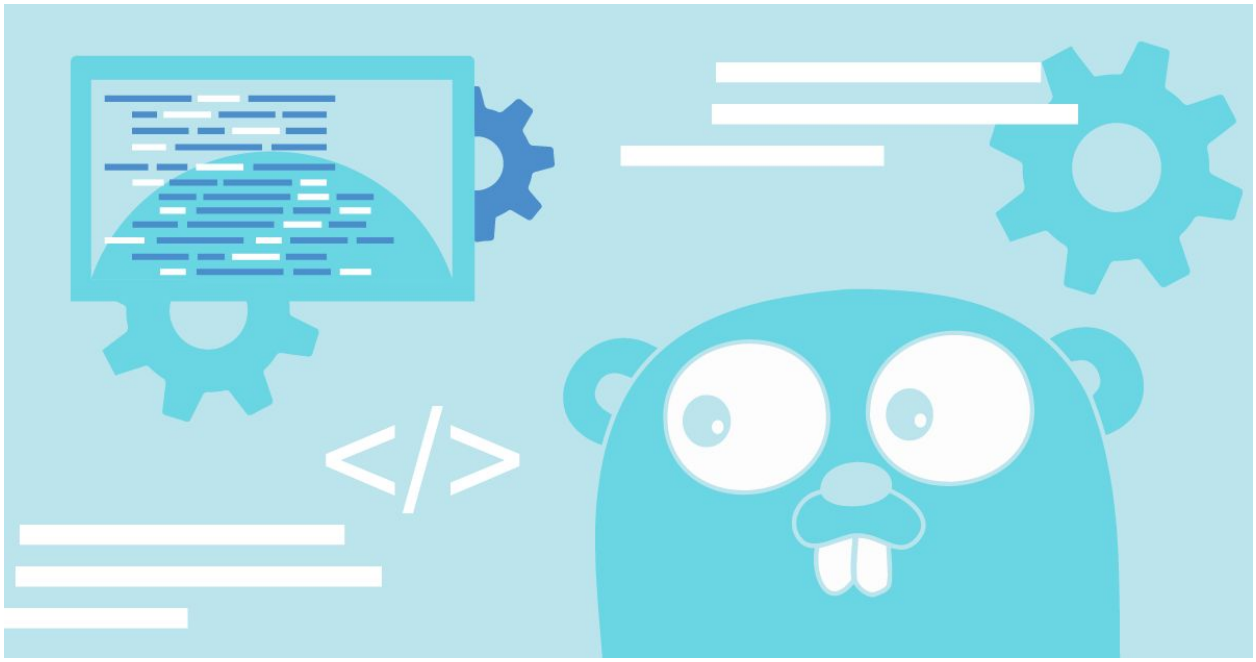


The (Un)official Go(lang) Training Guide

02.10.2020

David Federspiel
Rural Sourcing

Overview

Welcome to Golang Training 101!, your custom-tailored, fast-track guide to becoming a Go developer. This is a mildly aggressive, three week learning track that will guide you through the fundamentals of Go within the context of the web.

In today's tech-world, we need fast and scalable systems to support our clients, and Single Page Application (SPA) style projects are so commonplace it would be silly not to use them; this is awesome news because we've finally found ourselves in a world where frontend and backend systems can be completely disconnected, giving us a clear separation of concerns, and consequently a plethora of ways to implement our designs.

The focus of this course will be on Golang in a headless architecture. Personally, I'm a React kinda fellow, but you can build any frontend you'd like, be it Angular, Vue, Ember, Meteor, Mithril, Polymer.... the list goes on and on, but the great news is that it doesn't really matter.

Gone are the days of tightly bound systems: welcome to the future... **Gospeed**!

Goals of this Course

1. Learning to read Golang syntax
2. Understanding how to structure your Go app
3. Testing in Go
4. Building your own, *awesome*, web service

Tools

We don't need much to get started, just the Go SDK and an editor of your choice, but be aware that not all Go IDE's are alike, and some are downright frustrating. We'll talk about that soon, but before we do that, let's install the Go SDK:

Important

At some point in the *very near future* you will encounter **GOROOT** and **GOPATH**. These are part of the Go environment variables and they can be a little confusing at first. Just keep these two rules in mind in mind:

1. **GOROOT** must point to the SDK you're about to install (or other Go SDK's when/if you need to maintain multiple versions of Go)
2. **GOPATH** is a reference to your project files, or "workspace." Read more about it here as it requires a specific folder structure to work correctly

Installing the Go SDK

- Download windows **zip** file from https://golang.org/dl/
  *** don't download the msi installer as the network will probably block it.*
- Extract files to **C:\Go**
- Add the binary to your path:
  a. Open Windows Start Menu and start typing '**environment**'
  b. Click on '**Edit environment variables *FOR YOUR ACCOUNT***
  c. Click on Path and click the edit button
  d. Click '**new**' and add a reference to **C:\Go\bin**
- Open powershell or another terminal and type **go** and hit enter, it should pull up the go commands.
- If that didn't happen for some reason, reach out in the slack channel **#go_collective**

IDE's

JetBrains GoLand

We strongly recommend using JetBrains GoLand for development as it is a dedicated Golang IDE. At the time of this writing, VSCode support for go is **very** alpha and as the *gopher-guinea-pig group*[1] was going through training, we found that VSCode caused us a lot of extra troubleshooting hours which can be annoying and even spirit-killing as we were learning a new language.

VSCode

If you must use VSCode, it will still work, but just be aware that it can be a little glitchy. Also, get the Go extensions and read up on Go in VSCode here

The Unsolicited Advice

Now that you have your tools set up, if you haven't already, test your installation with a simple guided exercise here to make sure everything is working as expected. In addition, here are some tips to help you in learning:

1. Reading is less fun than doing, but reading is **KEY.** That said**,** go read Effective Go, like right now, and if you're like me, it'll make your head spin. Fear not, the victory will come when you read it again after a little while and your head only spins half as much as it did the first time. And if you can get through it all and haven't gotten dizzy, then look me up and teach me something.
2. Familiarize yourself with GoDoc, it is the defacto place for both the go standard library as well as many other packages that you will most definitely rely on. You're soon to realize the go ecosystem is amazing. In fact, let's take a quick detour and read about those now
3. Try to leave what you know about other languages behind - it's fine enough to contrast language nuance with ones you know, but you'll find that just about everything in Go has its own flavor, its own style, terminology, and you may encounter issues trying to bring other styles into Go.
4. Last but not least, Go, and everything around it, is open source, and once you achieve a level of literacy, nothing can stop you. Don't be afraid to crawl through burrows of source code to see how the community does it.

[1] Thanks to everyone in the FWA center that endured the first training session, and to RSI's own Go developer, Scott Brown for reviewing my code for quality and correctness.

Milestones

Over the next few weeks, you will be familiarizing yourself with Go, learning how to build an application that will scale, and finally putting it all into practice by building your own end-to-end application. It'll be broken down like this:

I. Go Literacy (Week 1)

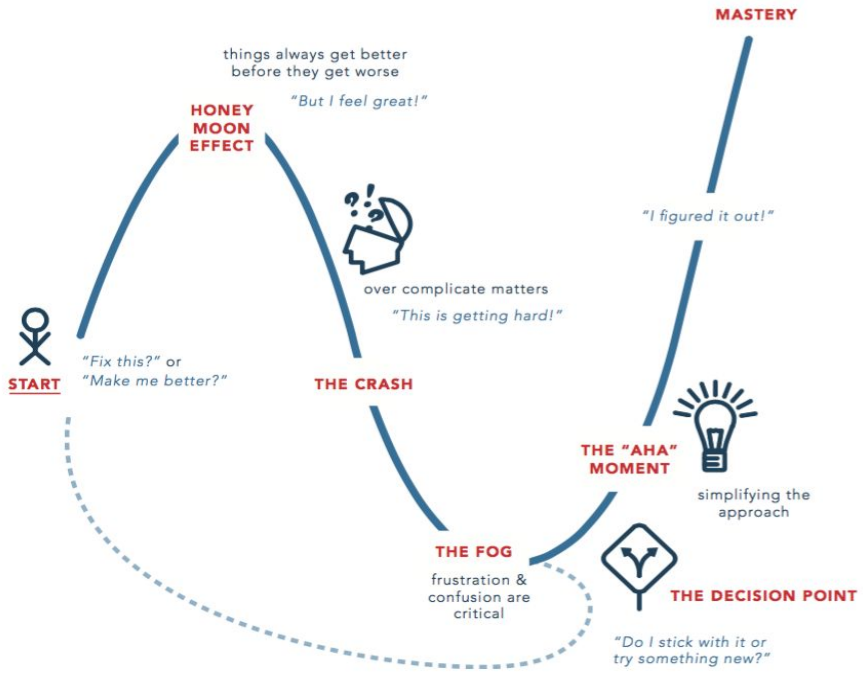
Deep study of syntax, familiarizing yourself with the environment and building simple apps to assert your understanding as you're introduced to the ins and outs of Go.

II. Go Architecture (Week 2)

Putting it all together can be a tough step. You feel comfortable with the language, and now it's time to build something real-world. How do you get from literacy to effectiveness?

III. Go Build Something (Week 3)

Now that you have a grasp of the language and some understanding of architecture, let's put it into action. By now you should be in the Honeymoon phase of the learning curve. This is the time to dive in feet first, while not forgetting your friends (**#go_collection #fwa_go**) along the way.



The One-Page Syllabus

Week One

Begin with A Tour of Go, a comprehensive introduction to all things Go. This will help you to familiarize yourself with the language, and covers quite a bit of ground using a handy web-based compiler where you can modify code and see the results in real-time.

Next, let's look at Go Web Programming (PDF), a guide to building a complete web application. Feel free to skim through this book for the good stuff; some of it will be remedial to seasoned web developers, but it will help make the connections between syntax and application design.

Week Two

Unit testing and TDD is all the rage these days, and if your enterprise code isn't testing critical components of your app, then you're leaving yourself vulnerable to headaches as the codebase changes and grows. That said, Learn Go with tests is an excellent re-entry to Go with testing in mind.

Delving further into architecture, the folks at Uber have been kind enough to share the way they implement Go in the following Go Style Guide, showing us the good and bad of writing Go. We all know that objectives can be implemented in a number of ways, and this reference points out a number of common patterns and how to deal with them in an idiomatic way.

At this point you're probably asking yourself where in the heck you're going to organize your files. Thankfully we have a very well documented repository with the shell of a sound Project Layout

Week Three

Now it's time to put all this into action. Assuming you've already started building your own awesome thing in Go, let's make sure you're firing on all cylinders with the following. Try and put all you've learned into practice with the following challenge. There are no wrong answers, just opportunities to learn and teach what you know. So let's get started!

fmt.Println("My %s Project", noun)

```
Go run main.go

> My Pet Project
```

Prerequisite

Just one before you start. Create a new project and do that whole **mod init** thing you learned about before. You *will* lose a percentage point if you call it **Pet.** Just kidding, but no really, now's the time to be original and bring forth the beginnings of your new, *amazing*, project.

Rules and Guidelines

Yep, you guessed it, **provide unit tests**. The cleanliness and readability of tests is just as important as our production code. You'll be writing these anyway, so best to learn now. Think simple. Readability and modularity are better than being clever.

Also think **MVP**, and don't get carried away with how you want to approach your idea. Limit yourself to dealing with just one or two structs so that you can cover the wide ground we need to cover. At most, take no longer than **three days** to hit your goal, from data to api access.

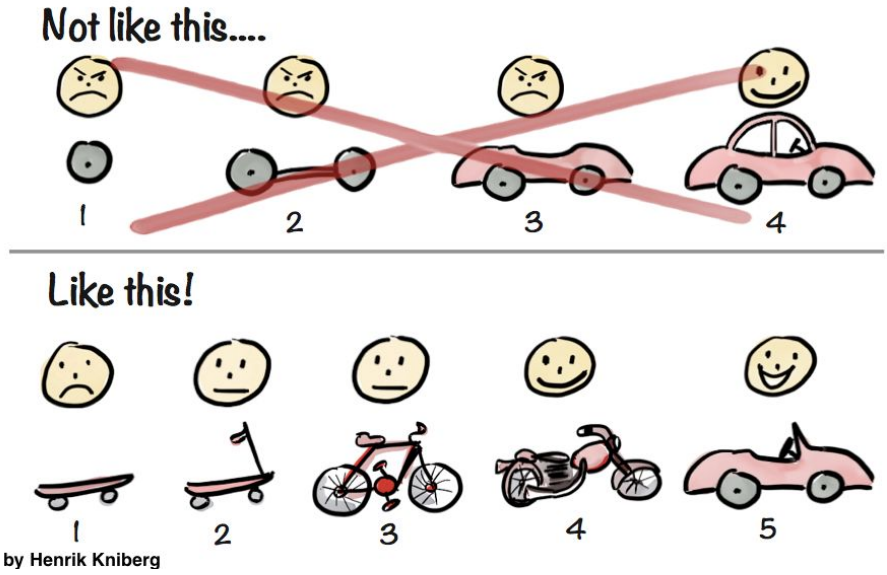
For instance: In my pet project, I created a basic CRUD service, using the most common verbs, POST, GET, PUT, and DELETE, to interact with a simple Pet structure. It was simple in broad terms, and it should be, but it worked well using Postman, persisted to a SqlLite database, and gave decent feedback across all endpoints. (e.g. status codes, response data, etc.)

Use whatever tools you're comfortable with, and feel free to explore the package ecosystem for code you can use to make things easier for you. Open source is here to support us, and who knows, you might even teach us a thing or two. (probably)

Once you complete your project, we'll give you a chance to show it off, talk about your experience overall, and celebrate your success. *Go Team Go Dev Go!*

Step 1 - Build a system capable of reading data

Assuming that we are building our application data-first, create a command-line app that implements a repository, one that will read from a file system or a database. It makes no difference which so long as it can read data. Thinking forward, we'll eventually need this system to support a full CRUD implementation, but for now let's focus on a read-only system first. Remember, MVP...



Inputs

You have many choices for data storage, including file systems and databases. I/O with plain text in a file system is a bit cumbersome compared to the more friendly data stores like JSON and databases. You're free to use delimited files if your heart desires, but some find it easier to store data in a lightweight DB or by mapping .json files to structs. Go with what you're familiar with for now, and keep it limited to a single structure.

- Structure should contain a DateCreated and DateModified field
- Structure should also contain an int, bool, and string type

Try to keep a clear line of separation between your I/O logic and what you'll be using to display the data.

Outputs

Output should be logged to the console using tests. Doing so should lead to a well-structured data layer that will transition nicely into Step 2.

Output 1 – List to the console all {noun}'s, sorted in ascending order

Output 2 – List one {noun} by parameter

Step 2 - Build a RESTful API to access your system

Tests for this section are required as well. Implement a read-only api that can leverage the logic you created in Step 1 by creating two http handlers capable of (1) listing all records in your data store, sorted and (2) retrieving a single record based on a parameterized input.

Inputs

You'll need to start a server and register a couple new routes for this to work. Handlers should interact with your code from Step 1, preferably through a layer of abstraction. (e.g. service layer)

`GET/{nouns}` - get all records, sorted

`GET /{noun}/:param` - get a single record by parameter

Outputs

It's your choice how you render the output from these endpoints as long as it is well structured data. These endpoints should return JSON.

Output 1 – List all {noun}'s, sorted in ascending order

Output 2 – List one {noun} by parameter

Step 3 - Modify your system so that it is capable of persisting data

It's time to hit our next MVP and add write capabilities so our API consumers don't start complaining about stale content. Building on our read-only implementation, add support for the remaining verbs, POST, PUT, and DELETE so that users can help crowd source data for you. By now you should have a pretty well structured app with a pattern you can follow for adding the remaining functionality.

Inputs

Create a few more routes now, similar as before, only we're going to need to write to our data store this time. Add the following handlers, and the implied logic to support the CUD part of our CRUD service.

`POST/{nouns}` - create a new record, sending json payload in the request body

`PUT /{noun}/:param` - update a record by parameter. Include payload in request body

`DELETE/{noun}/:param` - delete a record by parameter

Outputs

As before, it's your choice how you render the output from these endpoints as long as it is well structured and in JSON format.

Output 1 - `POST/{noun}` - returns records sorted by birth date

Output 2 - `PUT/{noun}/:param` - returns records sorted by name

Output 3 - `DELETE/{noun}/:param` - returns records sorted by name

Closing Notes

Three whole days isn't a lot of time to build an app, but I'm guessing a percentage of you will have already looked ahead and found this. If so, don't hesitate to dig in from day one and begin creating a project that grows with your learning. The above application represents a minimum viable app within a maximum timeframe, but in this agile world of ours we also understand that value changes over time and with greater understanding. That said, feel free to Go crazy, just be sure to timebox as necessary so that you can emerge a well-rounded Gopher.

Additional Resources

- Design Patterns for the Go pro (Use with caution, *i.e.* don't overuse)
- Tired of cyclical build issues? Read this Domain Driven Design doc, marvel at its elegance awhile, then dial it back just a bit and go refactor your code.
- Dependency Injection and Interface Indirection. Level-up by making your code more flexible through interfaces Testing with Interfaces and Mocks
- Another great resource on Go web design Building Web Applications with Go
- In addition to all these, there are numerous resources on Pluralsight and Udemy that can only help as you Go along.

Suggested Implementation

We recommend that any group selected for Go training **meet daily** over the three week course to discuss any challenges or breakthroughs they are having with the curriculum. Additionally, a **group leader** should be appointed to check in twice a week as a group to help steer the class and find opportunities for one-on-one study. Everyone learns at a different pace, and people will spin off as they feel comfortable, but it's important to keep everyone together throughout the training.

At the end of three weeks, everyone in the group will be given 1-3 days to complete their project assessment, followed by a code-review with an experienced Go developer. Assuming the code review goes well, any developer that completes training can be considered for Analyst I level Go development skills. (*with a focus in web*)