Getting Started with Go

An Introduction

21 March 2015

Jacob Walker
Gopher, Wrecking Ball Media Group

Background

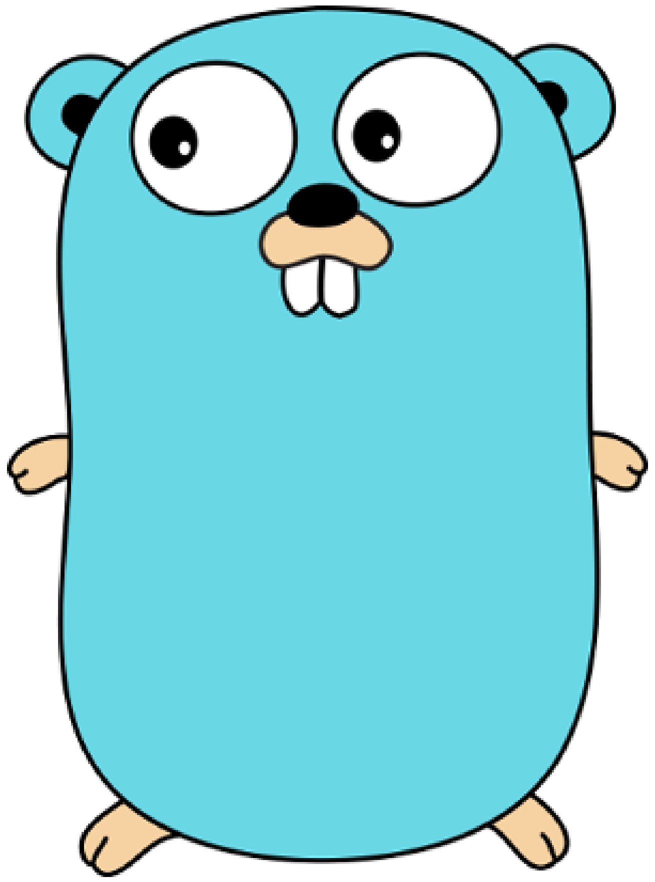
Me

The majority of my experience is in PHP. I ventured into Ruby, Python, and Node.js. Dabbled with C++.

Found Go and felt at home.

I'm not affiliated with Google, just a passionate ʕ◔ϖ◔ʔ

Enter The Gopher



Gopher by Renée French (http://www.reneefrench.com)

Go

Originally developed at Google by Ken Thompson, Rob Pike, and Robert Griesemer.

Relatively young: started in 2007 and announced in 2009.

Compiled, Statically Typed, Concurrent, Imperative language.

Originally developed in response to pain points in C and C++, Go has become
very popular among developers coming from dynamic general purpose languages.

Overview

My intended audience is someone who is interested in starting to use Go.

Any other gophers here?

- How to get started
- Notable Features of Go
- How to be successful

How To Get Started

Take the tour

https://tour.golang.org (https://tour.golang.org)

Install Go

- Recommended: Download from golang.org (https://golang.org/doc/install) and unpack

```
sudo tar -C /usr/local -xzf go1.4.linux-amd64.tar.gz
export PATH=$PATH:/usr/local/go/bin                    # Add to profile
```

- Use package manager?
- Build from source?

Test Installation

```
package main

import "fmt"

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

```
$ go run hello.go
Hello, World
```

How to Write Go Code

https://golang.org/doc/code.html (https://golang.org/doc/code.html)

Configure Workspace

```
export GOPATH=$HOME/go          # Add to profile
export PATH=$PATH:$GOPATH/bin  # Add to profile

$HOME/go
├── bin
├── pkg
└── src
    ├── bitbucket.com
    │   └── jcbwlkr
    │       └── hello
    └── github.com
        ├── hitjim
        │   └── coollib
        └── jcbwlkr
            ├── coolapp
            └── logger

12 directories, 0 files
```

Our First App

Put hello world in `$GOPATH/src/bitbucket.com/jcbwlkr/hello/main.go`

```
package main

import "fmt"

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

Build it!

```
$ pwd
~/go/src/bitbucket.com/jcbwlkr/hello
$ go build
$ ./hello
Hello, World
```

Take a look

```
$ ls -lh
total 1.8M
-rwxr-xr-x 1 jwalker jwalker 1.8M Mar 20 11:04 hello*
-rw-rw-r-- 1 jwalker jwalker   73 Mar 20 11:04 main.go
```

Single statically linked binary. Build, copy, run, profit!

Compilation

- Compiles fast (really fast).
- Easy to compile

```
go build
```

- Easy to cross compile

```
GOOS=linux GOARCH=amd64 go build
```

- Feels like a scripted language.

```
go run hello.go
```

- Can compile Go code that calls C code with cgo

Our First Library

`$GOPATH/src/github.com/jcbwlkr/strings/strings.go`

```go
package strings

func Reverse(s string) string {
    chars := []rune(s)
    rev := make([]rune, len(chars))

    for i, j := len(chars)-1, 0; i >= 0; i, j = i-1, j+1 {
        rev[j] = chars[i]
    }

    return string(rev)
}
```

Code from Stack Overflow (http://stackoverflow.com/questions/1752414/how-to-reverse-a-string-in-go)

Our First Test

`$GOPATH/src/github.com/jcbwlkr/strings/strings_test.go`

```
package strings_test

import (
    "testing"

    "github.com/jcbwlkr/strings"
)

func TestReverse(t *testing.T) {
    in := "Jacob 助步车"
    want := "车步助 bocaJ"
    got := strings.Reverse(in)

    if got != want {
        t.Errorf("Reverse(%q) = %q; want %q", in, got, want)
    }
}
```

Testing Continued

```
$ go test
PASS
ok      github.com/jcbwlkr/strings      0.004s
```

More options

```
$ go test -bench=.
$ go test -race
$ go test -cover
```

Fetch remote packages

```
go get github.com/russross/blackfriday
```

```
package main

import (
    "fmt"

    "github.com/russross/blackfriday"
)

func main() {
    in := `
# Title
## Subtitle

This is my content
`
    out := blackfriday.MarkdownCommon([]byte(in))

    fmt.Println(string(out))
}
```

Run

Notable Features

Summary

- Simplicity
- Tooling
- Concurrency

Simple syntax

- Only 25 keywords.
- Opinionated. Often only one way to do something.
- Only one looping construct `for` that can handle `for`, `foreach`, `while` styles of looping.
- Few needs for semicolons. Few need for parens.
- Mandatory braces. Have to be on same line.
- Double quotes for strings. Single quotes for single runes.
- Consistent, Predictable, Orthoganal

gofmt

Core tool that enforces consistent coding style.

No more bike-shedding over style.

Tabs vs Spaces? Non-issue.

Expansive standard library

http://golang.org/pkg/ (http://golang.org/pkg/)

- `net/http`: Build web servers and clients
- `text/template`, `html/template`: Templating tools
- `sync`: Tools supporting concurrency
- `encoding/json`: JSON Marshalling and Unmarshalling

Multiple return values

```go
package main

import "fmt"

func main() {
    n := 16
    d := 3
    q, r := divide(n, d)

    fmt.Printf("%d divided by %d is %d with remainder %d", n, d, q, r)
}

// divide accepts a number and divisor and returns the quotient and remainder
func divide(num, div int) (int, int) {
    quot := int(num / div)
    rem := num % div

    return quot, rem
}
```

Run

No exceptions

Intentionally omitted. Return and check errors instead.

```go
package main

import (
    "fmt"
    "log"
    "regexp"
)

func main() {
    input := "The rain in Spain falls mainly on the plains"

    re, err := regexp.Compile("[rm[ain")
    if err != nil {
        log.Fatalln(err)
    }

    matches := re.FindAllString(input, -1)

    fmt.Println(matches)
}
```

Run

Error Handling

Don't ignore errors!

Handling errors can be repetitive but being explicit you know how they are handled. You have the full power of the language at your disposal to respond to errors.

Use `Must` versions when you know it won't error such as `MustCompile`.

Combine error handling when appropriate.

`panic` only when absolutely necessary

Blog post by Rob Pike: Errors Are Values (https://blog.golang.org/errors-are-values)

Concurrency With "go"

```go
package main

import (
    "fmt"
    "time"
)

func main() {
    go printEvery("In goroutine", 500*time.Millisecond)
    printEvery("In main", 250*time.Millisecond)
}

func printEvery(s string, d time.Duration) {
    for i := 0; i < 10; i++ {
        fmt.Println(s)
        time.Sleep(d)
    }
}
```

Run

Synchronized Concurrency

```go
package main

import (
    "fmt"
    "sync"
    "time"
)

func main() {
    wg := &sync.WaitGroup{}
    wg.Add(2)
    go printEvery("Foo", 500*time.Millisecond, wg)
    go printEvery("Bar", 250*time.Millisecond, wg)
    wg.Wait()
}

func printEvery(s string, d time.Duration, wg *sync.WaitGroup) {
    defer wg.Done()

    for i := 0; i < 10; i++ {
        fmt.Println(s)
        time.Sleep(d)
    }
}
```

Run

Concurrency Continued

- Channels: Typed conduits for communicating between goroutines. Use for message passing and synchronization.
- Package sync provides Mutexes, WaitGroups and other synchronization tools.

More Notable Features

- Statically Typed
- Automatic garbage collection
- OOP Composition over Inheritance (No direct inheritance)
- Implicit interfaces
- First class unicode Support
- No generics

How to Be Successful

Read!

- Effective Go (https://golang.org/doc/effective_go.html)
- Code Review Comments (https://github.com/golang/go/wiki/CodeReviewComments)
- The spec (https://golang.org/ref/spec)
- The FAQ (https://golang.org/doc/faq)
- The Go Blog (https://blog.golang.org/index)
- Read the standard library source

Use the tools

- `gofmt` (or `goimports`) on save
- `golint` look for style errors
- `go vet` look for suspicious / dangerous code
- godoc browse Go Doc of all locally installed packages

Editor Love

Editor Plugins

- Vim-Go (https://github.com/fatih/vim-go)
- Emacs go-mode (https://github.com/dominikh/go-mode.el)
- Atom.io go-plus (https://atom.io/packages/go-plus)
- GoSublime (https://github.com/DisposaBoy/GoSublime)

IDE

- LiteIDE (https://github.com/visualfc/liteide)

Misc & Best Practices

- Write Idiomatic Go. Don't try to write Python/Ruby/Java.
- Vendor your dependencies
- Use channels to communicate between goroutines.

```
Do not communicate by sharing memory; instead, share memory by communicating.
```

- Go, not Golang

Community

- Gopher Slack (http://bit.ly/go-slack-signup)
- `#go-nuts` on `irc.freenode.net`
- Exercism.io (http://exercism.io/) Go exercises with community sourced feedback
- Monthly Go Challenge (http://golang-challenge.com/)

Other Resources

- Awesome Go (https://github.com/avelino/awesome-go) (curated list of great community packages)
- Go By Example (https://gobyexample.com/) Go tutorials
- Golang weekly (http://golangweekly.com/) free weekly newsletter
- RSS Feed of StackOverflow questions tagged go

Thank you

Jacob Walker
Gopher, Wrecking Ball Media Group
http://jacob-walker.com (http://jacob-walker.com)
@jcbwlkr (http://twitter.com/jcbwlkr)