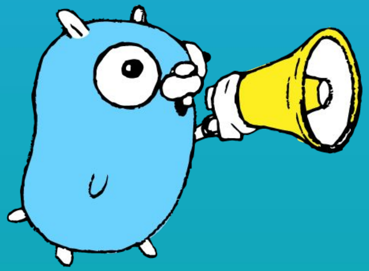
FOR GoTO

INTRO TO GO

Alexander Kallaway

Pilot Interactive, Inc.

Twitter: @ka11away
www.dotheoppo.site



Go Gopher Go!

About Go 01

Why Learn Go 02

Projects To Check Out 03

On Syntax 04

Let’s Get Into Code 05

Surprises 😅🍀 06

SECTION ONE

About Go

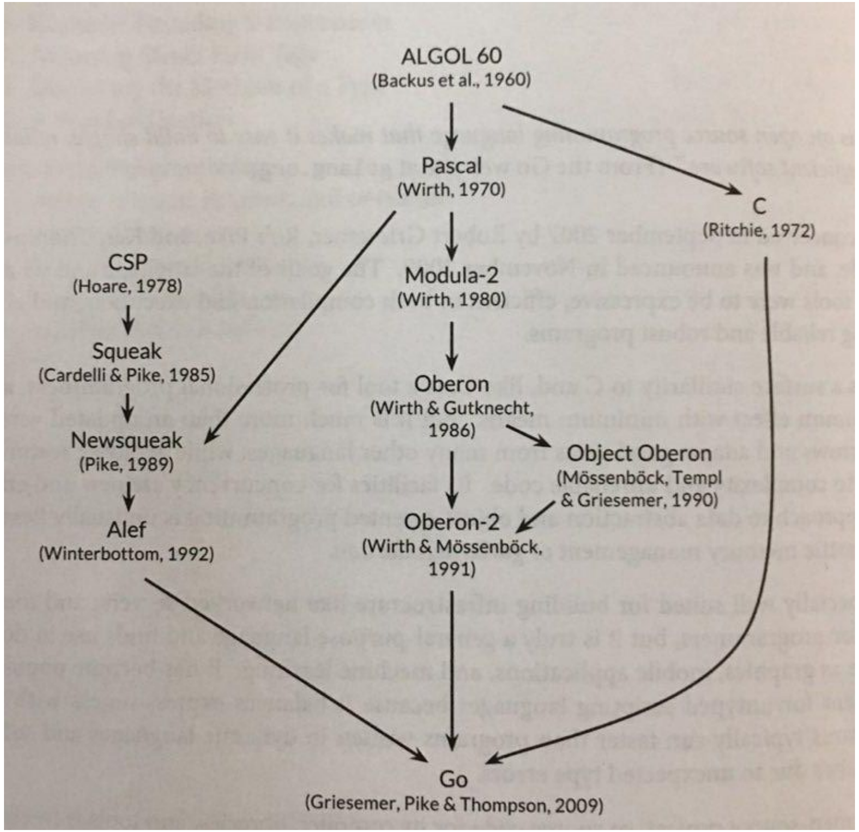
Where Created & Who's Behind Go

- Created at Google
- Released as Open Source Software in 2009
- Creators: Robert Griesemer, Rob Pike, and Ken Thompson
- They also worked on (created/contributed to) C, B, Unix, Unicode, JVM, and others

Go’s Values

- Thoughtful
- Simple
- Efficient
- Reliable
- Productive
- Friendly

Go Family Tree



ALGOL 60
(Backus et al., 1960)

Pascal
(Wirth, 1970)

C
(Ritchie, 1972)

CSP
(Hoare, 1978)

Modula-2
(Wirth, 1980)

Squeak
(Cardelli & Pike, 1985)

Oberon
(Wirth & Gutknecht, 1986)

Newsqueak
(Pike, 1989)

Object Oberon
(Mössenböck, Templ & Griesemer, 1990)

Oberon-2
(Wirth & Mössenböck, 1991)

Alef
(Winterbottom, 1992)

Go
(Griesemer, Pike & Thompson, 2009)

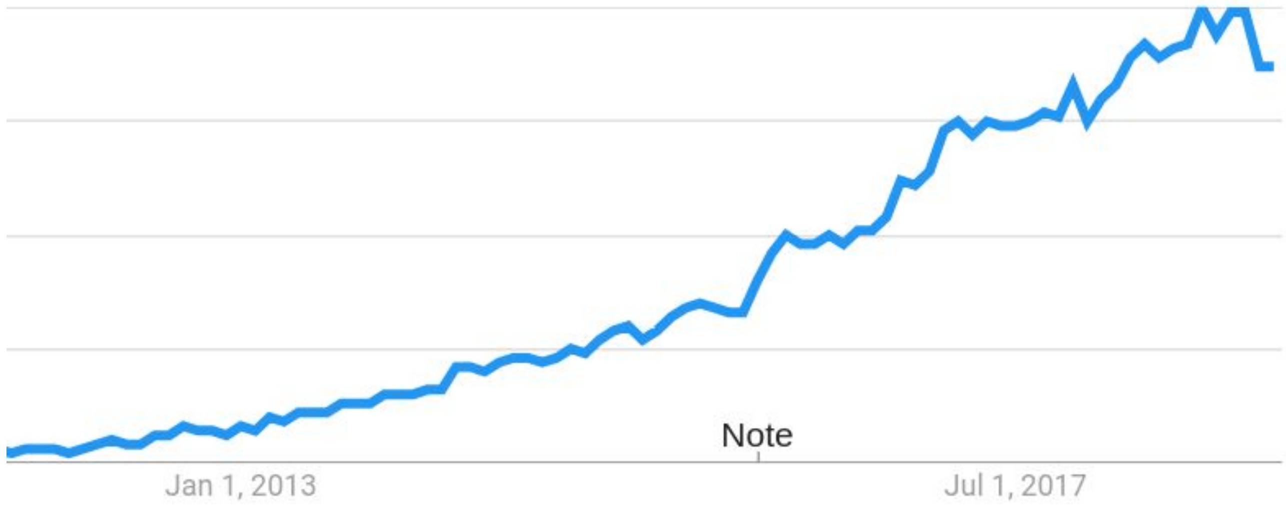
Go’s Main Features

- Go is a modern, general purpose language
- Strongly typed, compiled
- Automatic garbage collection
- Concurrency as a core language feature
- Unique approach to error handling
- No classes, structs and interfaces instead
- Simple, consistent language design

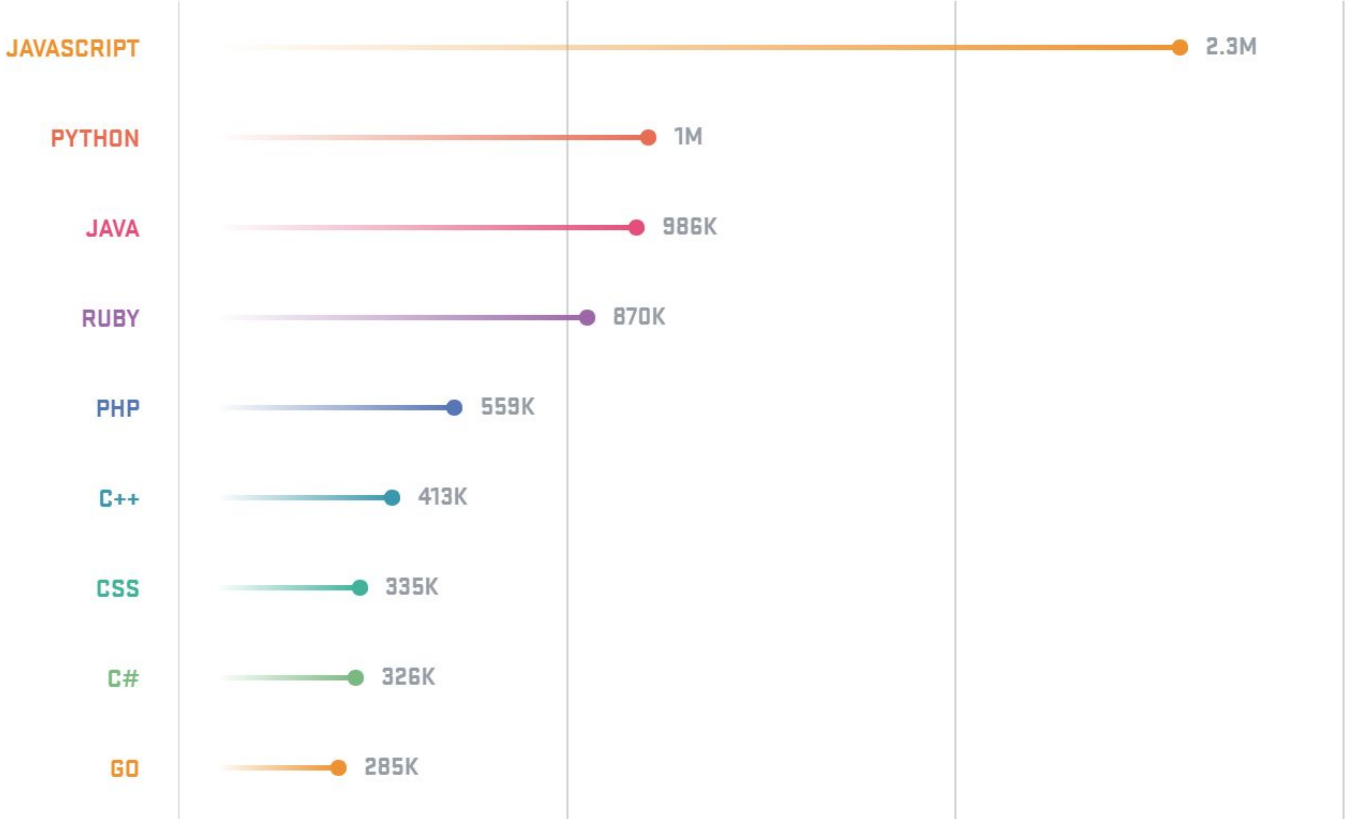
SECTION TWO

Why Learn Go?

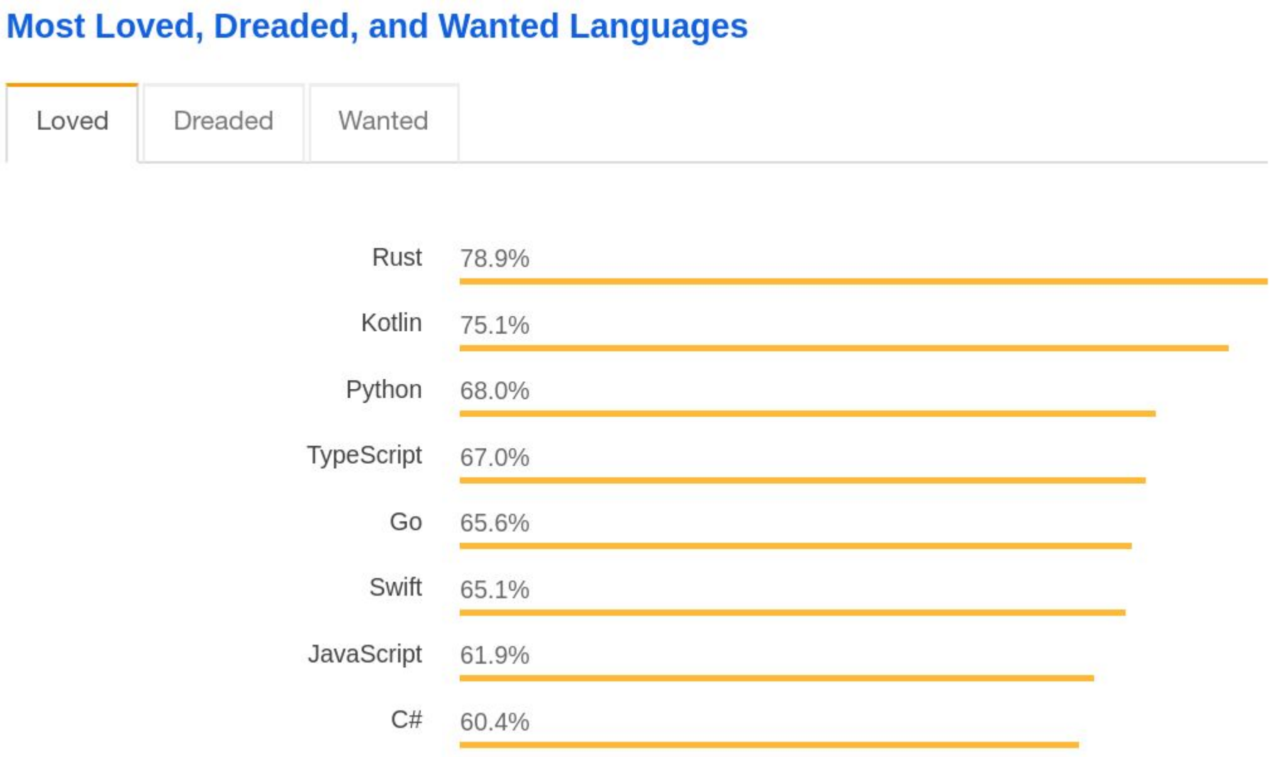
Google Trends - Golang - Interest over time



9th out of 15 Most Popular Languages on Github (by Pull Requests) as of 2017



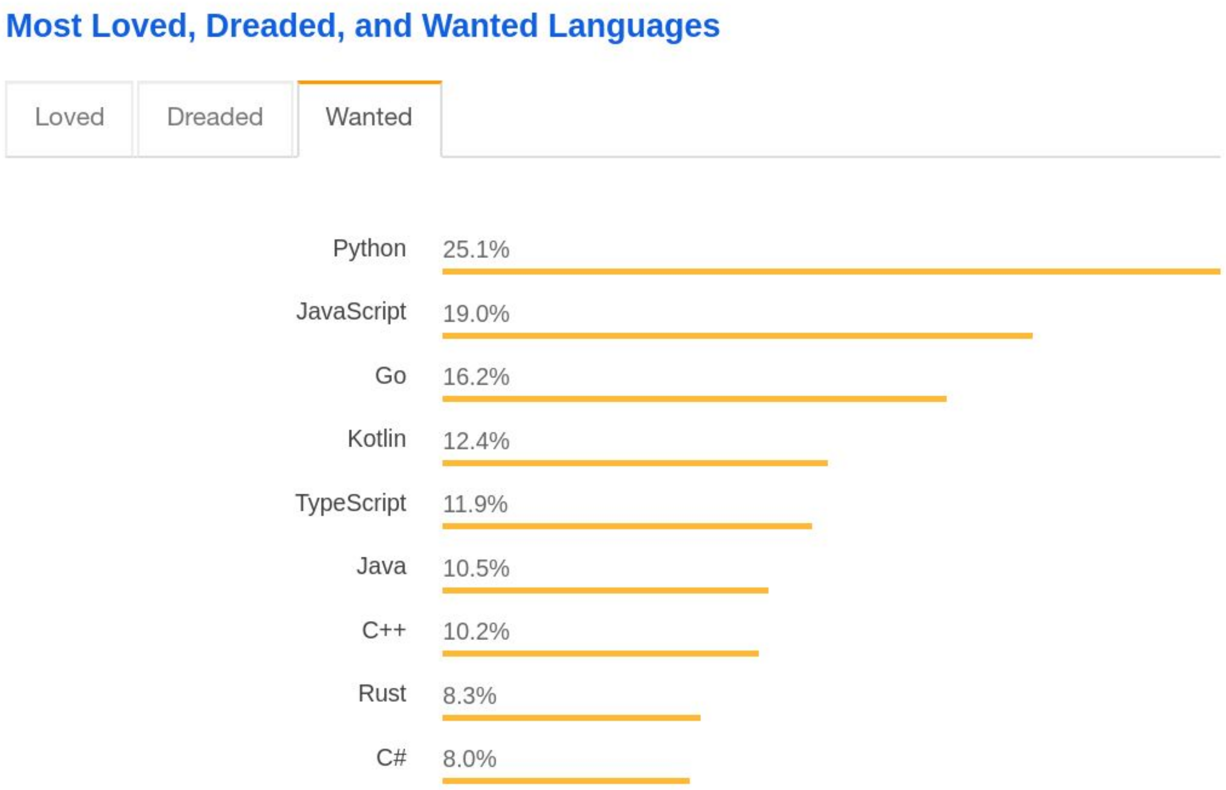
Why Go?



Most Loved, Dreaded, and Wanted Languages

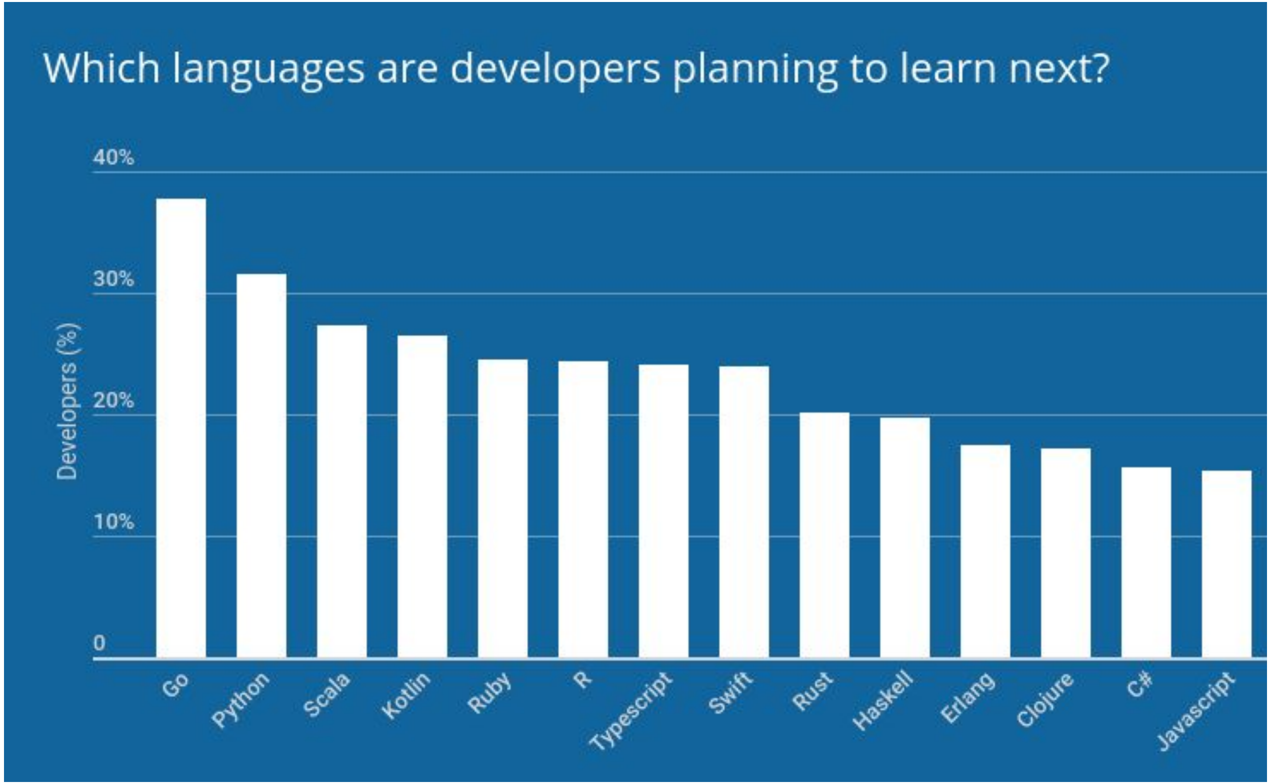
Rust	78.9%
Kotlin	75.1%
Python	68.0%
TypeScript	67.0%
Go	65.6%
Swift	65.1%
JavaScript	61.9%
C#	60.4%

Why Go?

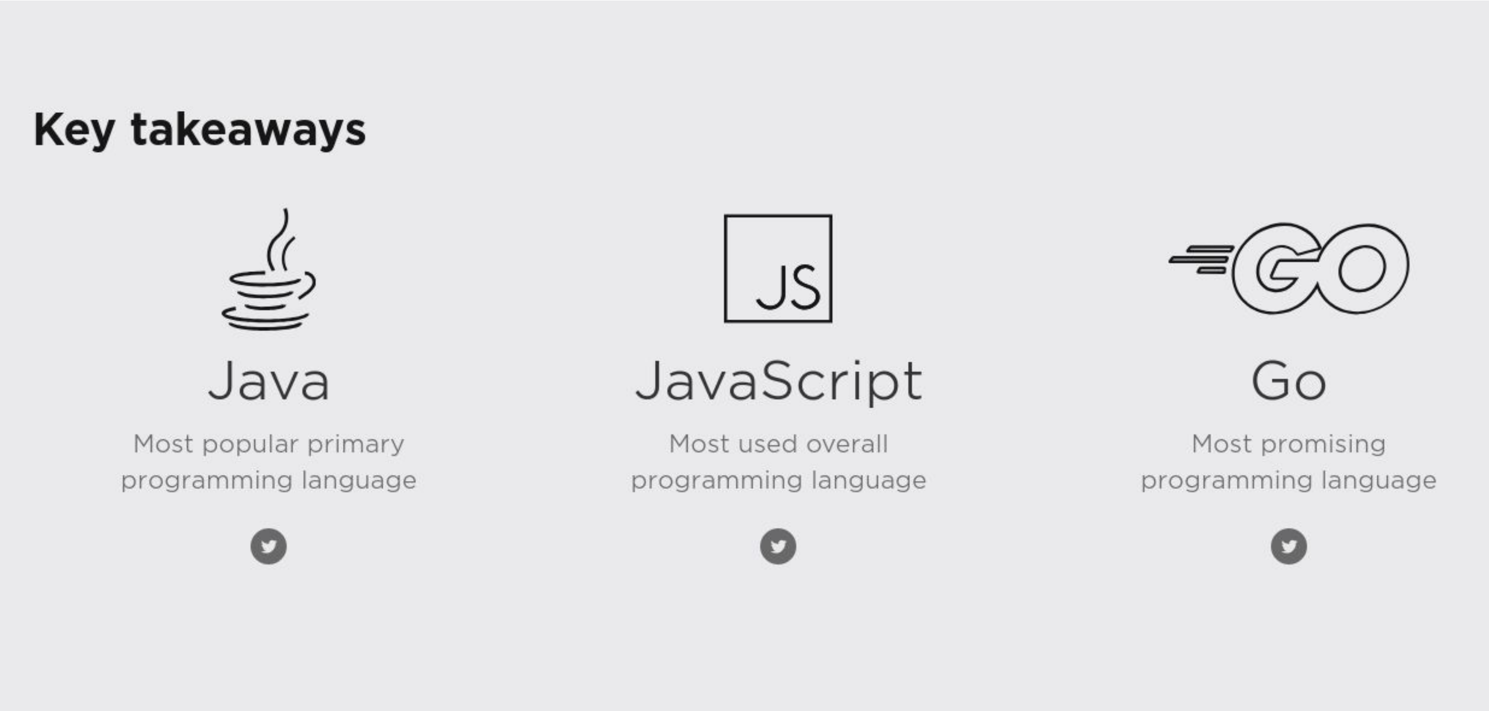


Most Loved, Dreaded, and Wanted Languages

From HackerRank’s 2018 Developer Survey



From JetBrains’s 2018 Developer Survey



Key takeaways

Java

Most popular primary programming language

JavaScript

Most used overall programming language

Go

Most promising programming language

Companies/Projects Using Go

- Google
- YouTube
- Intel
- Dropbox
- Uber
- BBC
- The Economist
- The New York Times
- IBM
- Twitter
- Facebook
- Tumblr
- 500px
- Wattpad
- Hootsuite ;)
- Koho
- Tencent
- CircleCI
- Honeywell
- Netflix
- Pinterest
- Slack
- thoughtbot
- Reddit

More info here: https://github.com/golang/go/wiki/GoUsers

Why Learn Go?

- It’s insanely fast
- Concurrency features built into the core
- Rich standard library (+ Networking packages are available in standard library)
- Simple, consistent design (think Unix)
- Moore’s Law is failing

Why Learn Go?

- Developers love it - it’s pleasant to write code in it
- It’s rapidly gaining popularity
- Supports Unicode by default
- Incredible quality of documentation: https://golang.org/pkg/

Go is Cross-Platform

- It is used for different platforms, including Windows, Linux, Unix and BSD versions and mobile devices (starting from 2015). In addition, it compiles well on many OS’s.

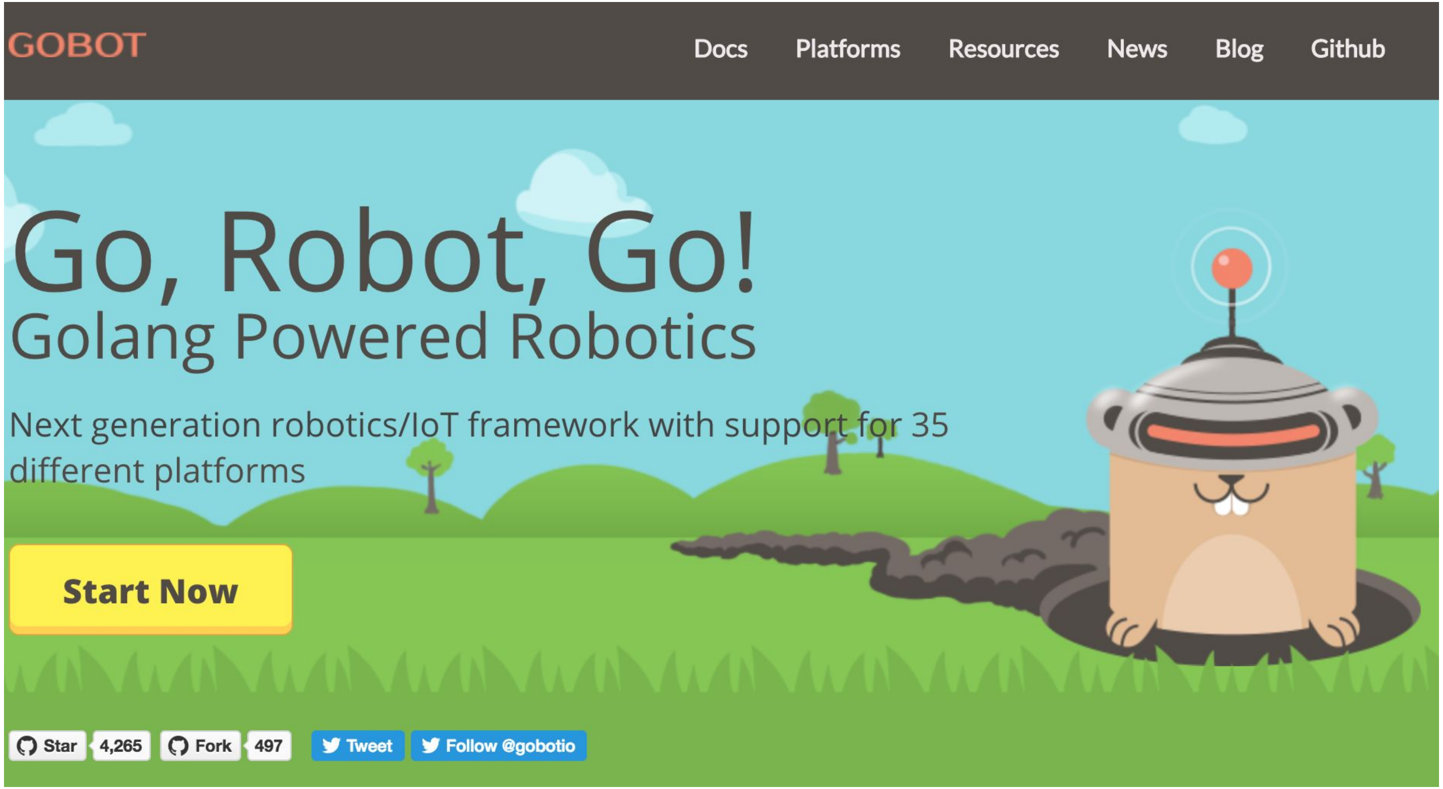
Why Learn Go?

- Go empowered the creation of new bold open source projects like Docker, Kubernetes and Ethereum
- Go’s ability to handle large amounts of load with less memory and CPU cycles translates into savings in servers and hardware costs. There are stories of organizations going from 30 servers to just 2 handling the same load by migrating to Go.

SECTION THREE

Projects to check out

Gobot.io



GOBOT

Docs Platforms Resources News Blog Github

Go, Robot, Go!

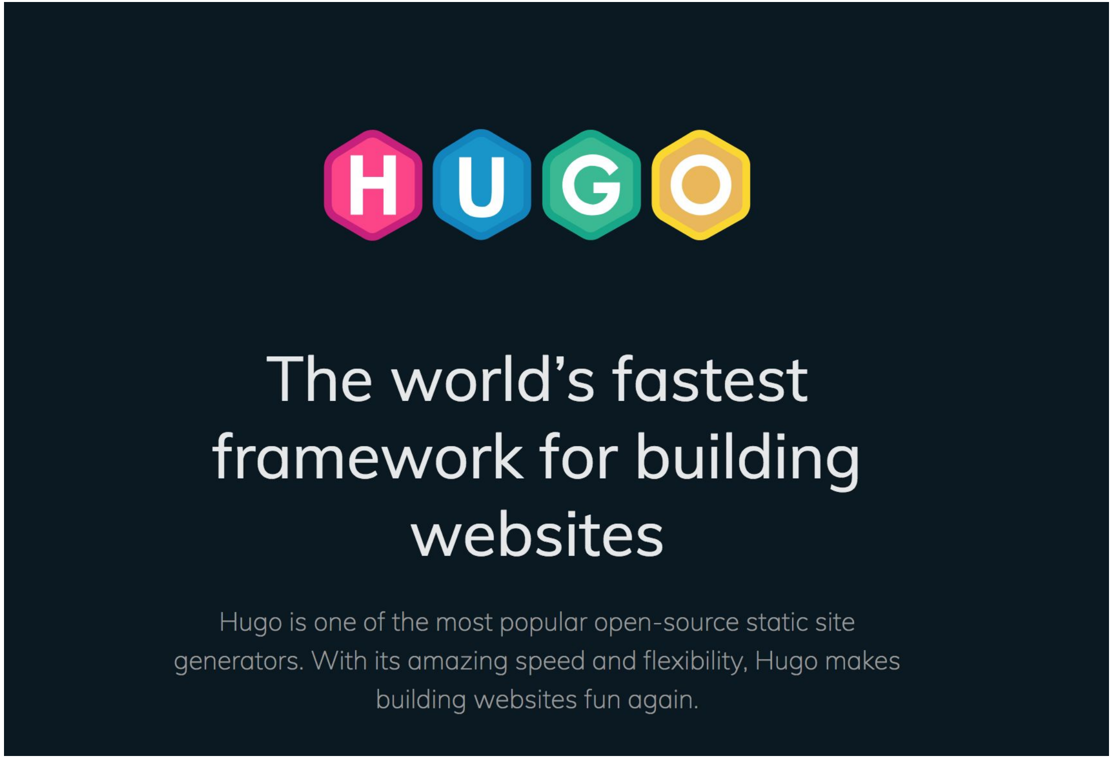
Golang Powered Robotics

Next generation robotics/IoT framework with support for 35 different platforms

Start Now

Star 4,265 Fork 497 Tweet Follow @gobotio

Hugo - Static Site Generator (gohugo.io)

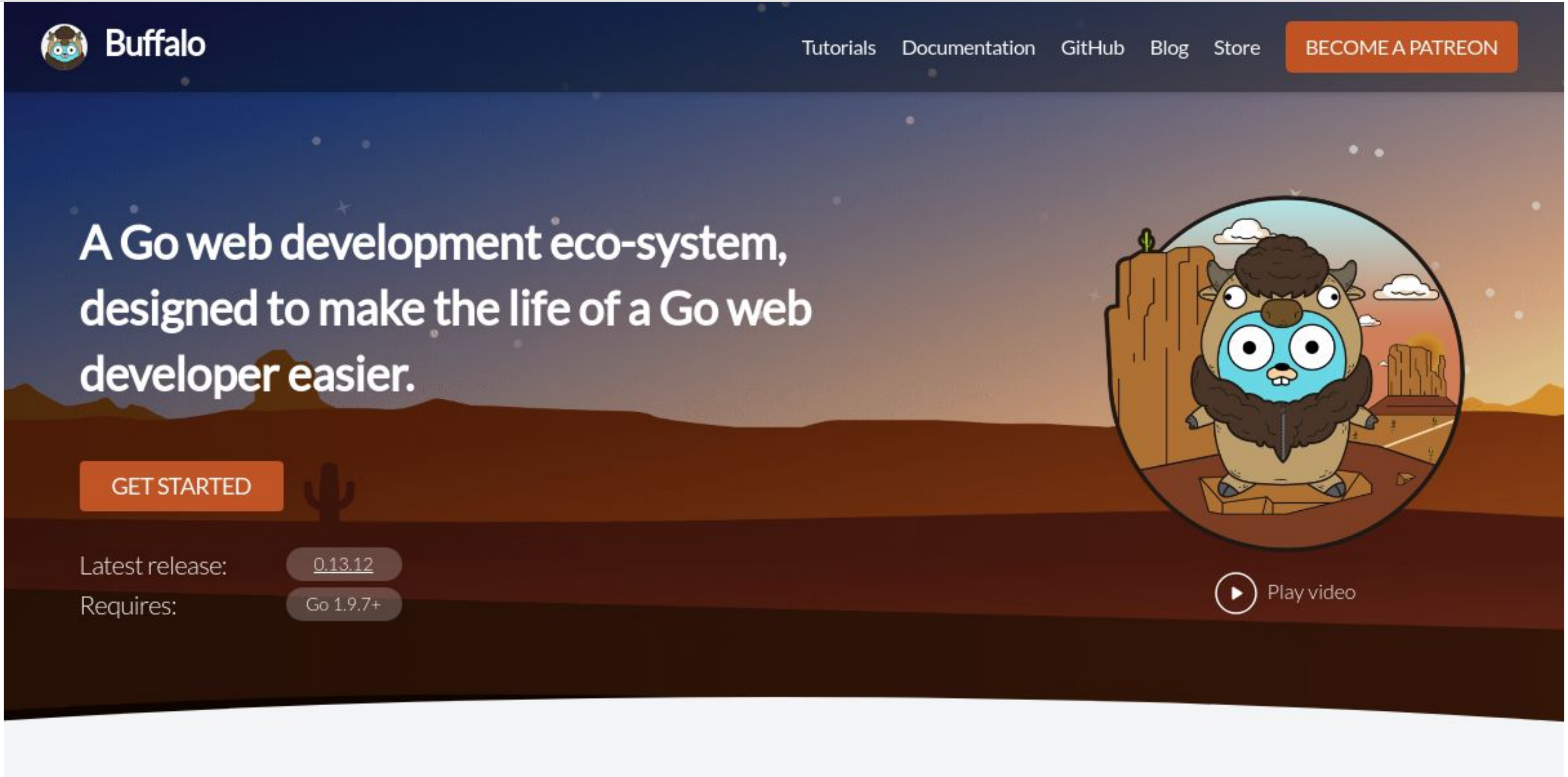


HUGO

The world’s fastest framework for building websites

Hugo is one of the most popular open-source static site generators. With its amazing speed and flexibility, Hugo makes building websites fun again.

Go Buffalo - Rapid Web Development with Go

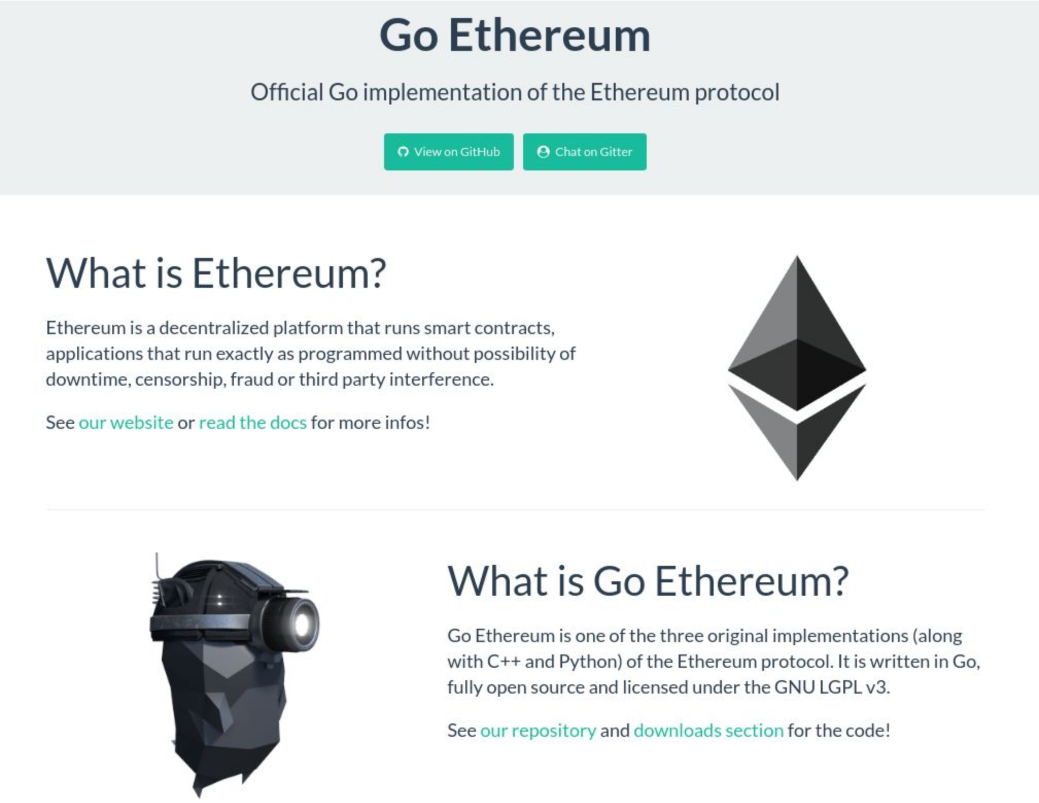


A Go web development eco-system, designed to make the life of a Go web developer easier.

Latest release: 0.13.12

Requires: Go 1.9.7+

Go Ethereum



Go Ethereum

Official Go implementation of the Ethereum protocol

What is Ethereum?

Ethereum is a decentralized platform that runs smart contracts, applications that run exactly as programmed without possibility of downtime, censorship, fraud or third party interference.

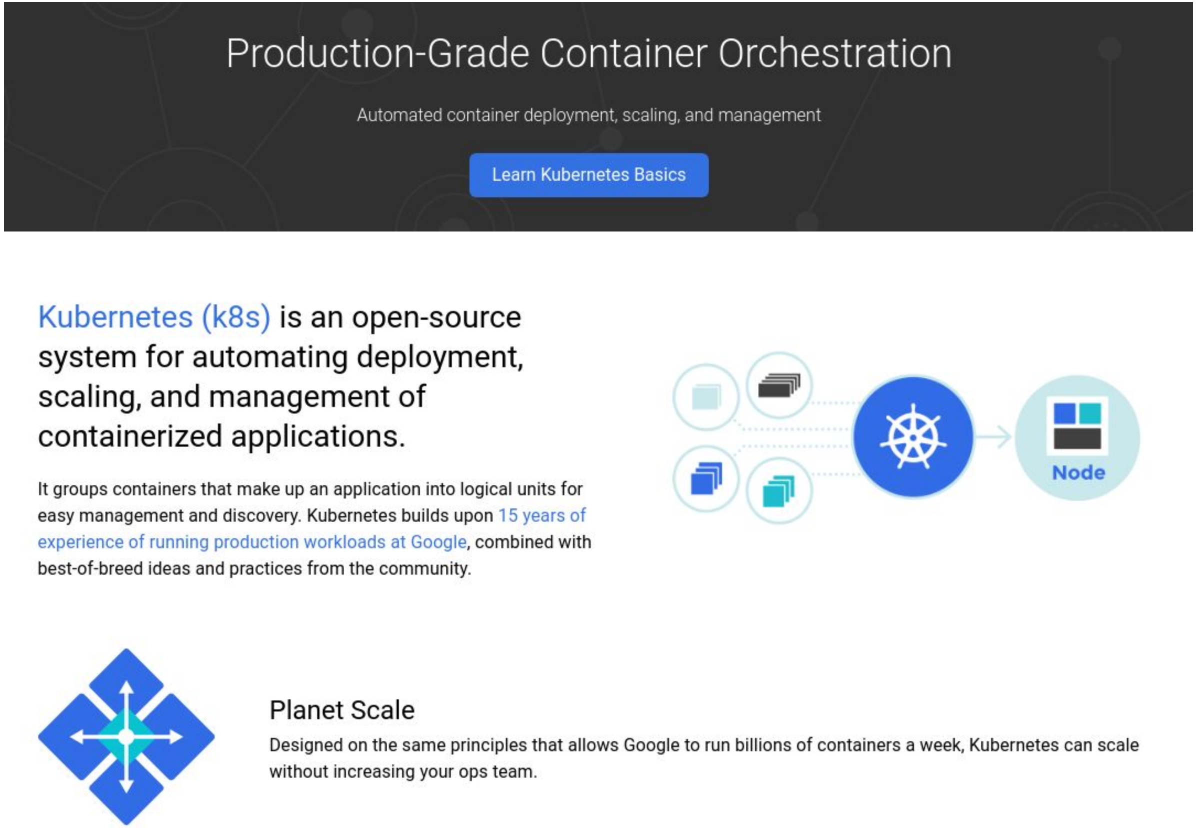
See our website or read the docs for more infos!

What is Go Ethereum?

Go Ethereum is one of the three original implementations (along with C++ and Python) of the Ethereum protocol. It is written in Go, fully open source and licensed under the GNU LGPL v3.

See our repository and downloads section for the code!

Kubernetes



Production-Grade Container Orchestration

Automated container deployment, scaling, and management

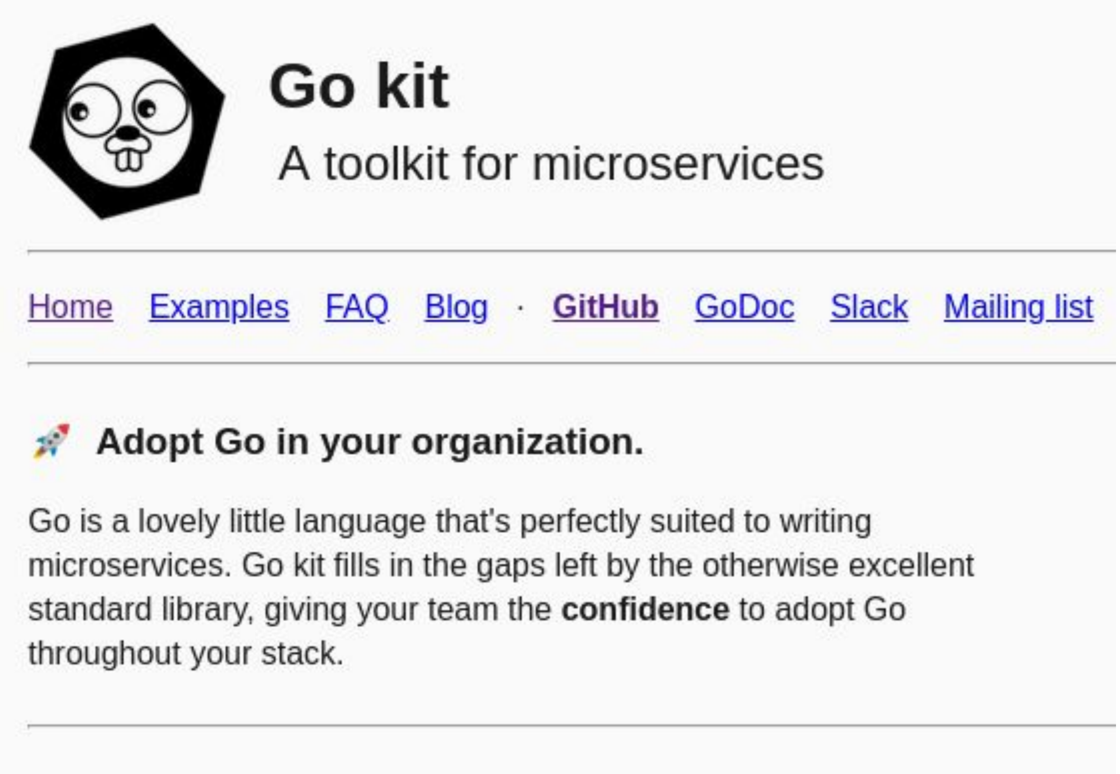
Kubernetes (k8s) is an open-source system for automating deployment, scaling, and management of containerized applications.

It groups containers that make up an application into logical units for easy management and discovery. Kubernetes builds upon 15 years of experience of running production workloads at Google, combined with best-of-breed ideas and practices from the community.

Planet Scale

Designed on the same principles that allows Google to run billions of containers a week, Kubernetes can scale without increasing your ops team.

Go Kit - For Microservices



Go kit

A toolkit for microservices

Home Examples FAQ Blog · GitHub GoDoc Slack Mailing list

🚀 Adopt Go in your organization.

Go is a lovely little language that's perfectly suited to writing microservices. Go kit fills in the gaps left by the otherwise excellent standard library, giving your team the **confidence** to adopt Go throughout your stack.

Also, check out: https://github.com/micro/go-micro

SECTION FOUR

On Syntax

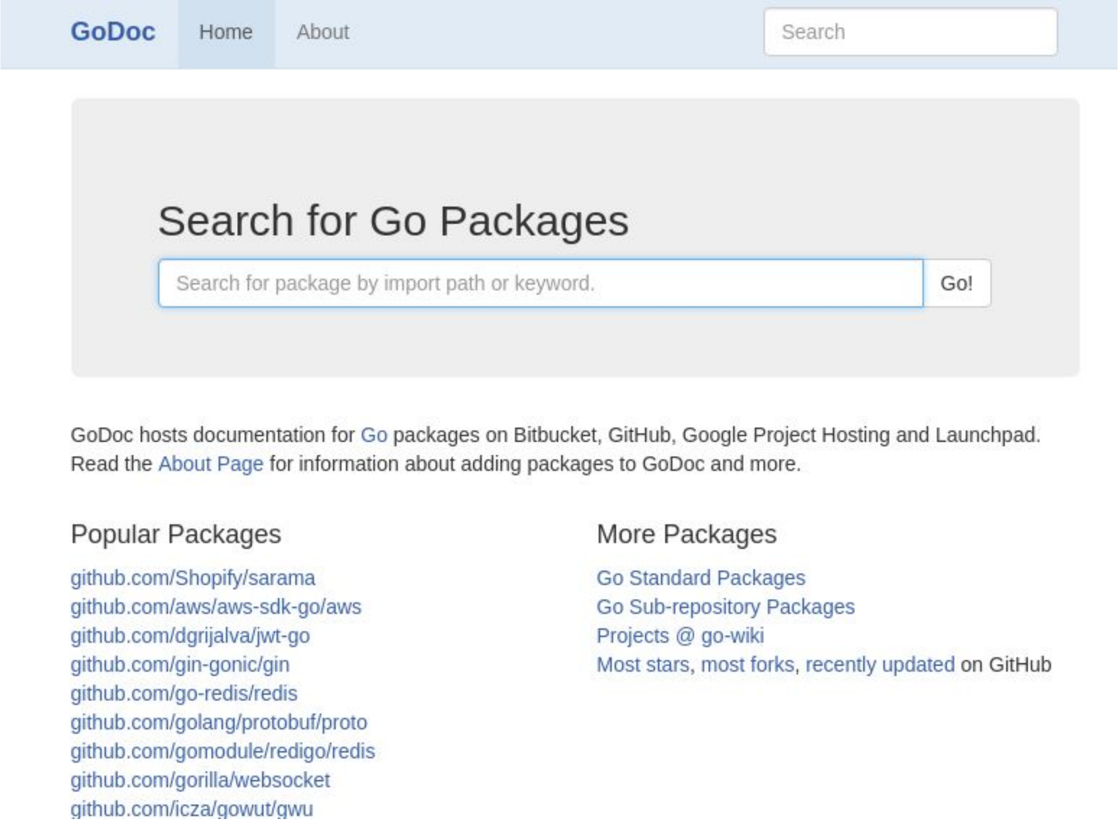
Syntax I

- No semicolons
- If/Else statements and loops don’t use ()
- Everything should go through go fmt tool
- You can only use " for strings, not single quotes - these are reserved for ‘runes’ (Unicode characters)

Syntax II

- Anything named with a Capital letter is exported
- Unused variables and packages are not allowed - your code won’t compile ;)

godoc



Search for Go Packages

GoDoc hosts documentation for Go packages on Bitbucket, GitHub, Google Project Hosting and Launchpad. Read the About Page for information about adding packages to GoDoc and more.

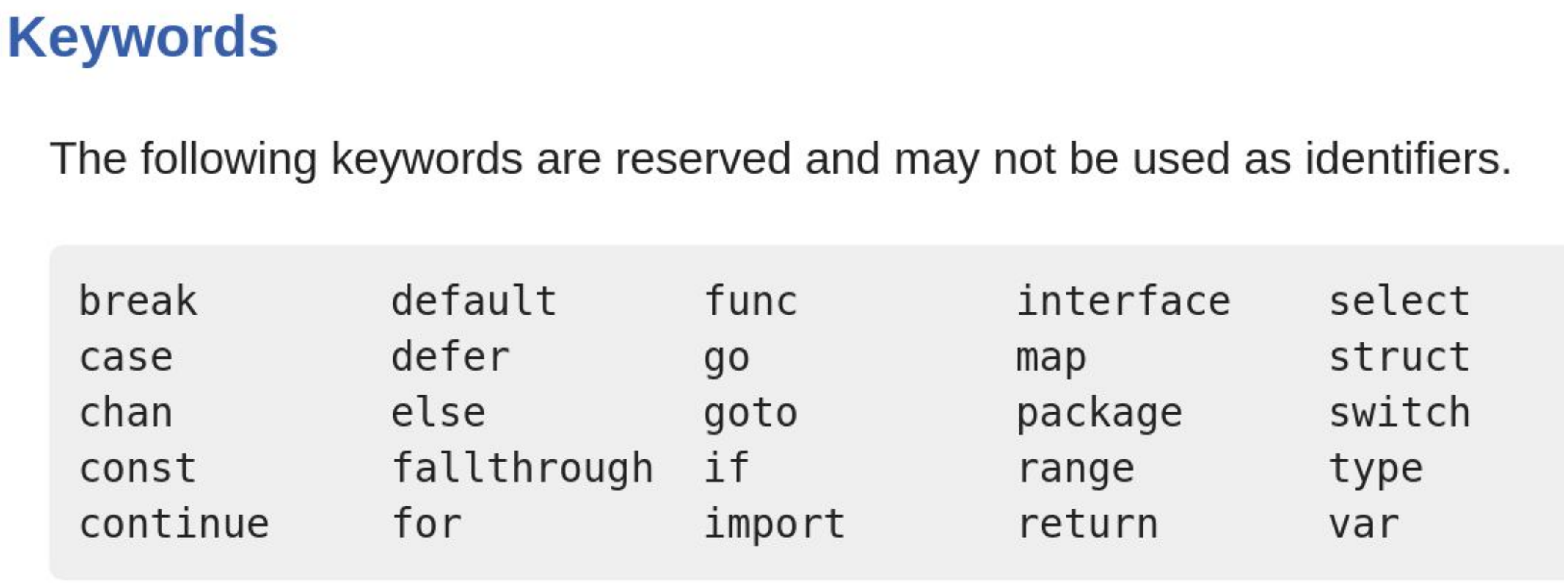
Popular Packages

github.com/Shopify/sarama
github.com/aws/aws-sdk-go/aws
github.com/dgrijalva/jwt-go
github.com/gin-gonic/gin
github.com/go-redis/redis
github.com/golang/protobuf/proto
github.com/gomodule/redigo/redis
github.com/gorilla/websocket
github.com/icza/gowut/gwu

More Packages

Go Standard Packages
Go Sub-repository Packages
Projects @ go-wiki
Most stars, most forks, recently updated on GitHub

25!

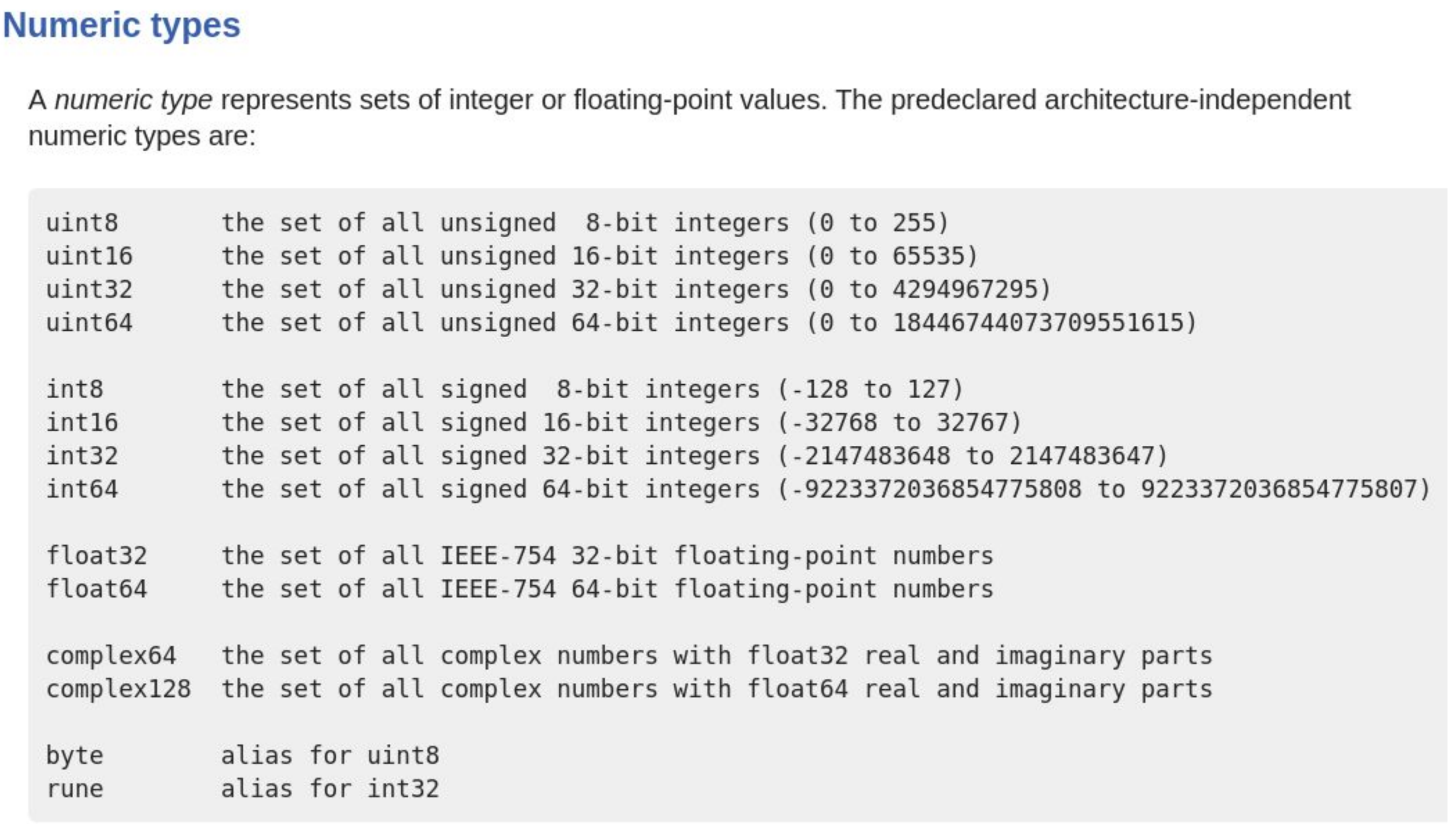


Keywords

The following keywords are reserved and may not be used as identifiers.

```
break        default      func         interface    select
case         defer        go           map          struct
chan         else         goto         package      switch
const        fallthrough  if           range        type
continue     for          import       return       var
```

Numeric Types + **string** and **bool**



Numeric types

A *numeric type* represents sets of integer or floating-point values. The predeclared architecture-independent numeric types are:

```
uint8       the set of all unsigned  8-bit integers (0 to 255)
uint16      the set of all unsigned 16-bit integers (0 to 65535)
uint32      the set of all unsigned 32-bit integers (0 to 4294967295)
uint64      the set of all unsigned 64-bit integers (0 to 18446744073709551615)

int8        the set of all signed  8-bit integers (-128 to 127)
int16       the set of all signed 16-bit integers (-32768 to 32767)
int32       the set of all signed 32-bit integers (-2147483648 to 2147483647)
int64       the set of all signed 64-bit integers (-9223372036854775808 to 9223372036854775807)

float32     the set of all IEEE-754 32-bit floating-point numbers
float64     the set of all IEEE-754 64-bit floating-point numbers

complex64   the set of all complex numbers with float32 real and imaginary parts
complex128  the set of all complex numbers with float64 real and imaginary parts

byte        alias for uint8
rune        alias for int32
```

SECTION FIVE

Let’s get into code

HELLO WORLD

```go
package main

import "fmt"

func main() {
    fmt.Println("hello world")
}
```

To run the program, put the code in hello-world.go and use go run.

VARIABLES I

```
var a = "initial"
fmt.Println(a)

var b, c int = 1, 2
fmt.Println(b, c)

var d = true
fmt.Println(d)
```

You can declare multiple variables at once. Go will infer the type of initialized variables.

VARIABLES II

```
var e int
fmt.Println(e)

f := "short"
fmt.Println(f)

const n = 500
```

Variables declared without a corresponding initialization are zero-valued. For example, the zero value for an int is 0.

LOOPS IN GO I

```
i := 1
for i <= 3 {
  fmt.Println(i)
  i = i + 1
}

for j := 7; j <= 9; j++ {
  fmt.Println(j)
}
```

for is Go’s only looping construct.

LOOPS IN GO II

```
for {
  fmt.Println("loop")
  break
}
```

for {} is an infinite loop. We can ‘break’ or ‘continue’ within a loop

IF/ELSE STATEMENT

```
if a == 7 {
  // do something
}
```

There are no () brackets for the condition in an if else condition.

SWITCH I

```go
i := 3
switch i {
case 1:
    fmt.Println("one")
case 2:
    fmt.Println("two")
case 3:
    fallthrough
case 4:
    fmt.Println("more than 3")
}
```

fallthrough has to be explicit.

SWITCH II

```
whatAmI := func(i interface{}) {
    switch t := i.(type) {
    case bool:
        fmt.Println("I'm a bool")
    case int:
        fmt.Println("I'm an int")
    default:
        fmt.Printf("Don't know type %T\n", t)
    }
}
whatAmI(true)
whatAmI(23)
```

You can **switch by type**

POINTERS

```
type person struct {
    name string
    age  int
}

peterPtr := &person{"Peter", 32}

fmt.Println(peterPtr) // &{Peter 32}
fmt.Println(&peterPtr) // 0x40e128 (for example)
```

ERRORS

```
resp, err := http.Get("https://picsum.photos/200/300/")
if err != nil {
    log.Fatal("Error:", err)
}

defer resp.Body.Close()

file, err := os.Create("img/image1.jpg")
if err != nil {
    log.Fatal(err)
}

// do other things
```

FUNCTIONS

```
func square(a int) int {
  return a * a
}

func sumThree(a, b, c int) int {
  return a + b + c
}
```

Go requires explicit returns, i.e. it won’t automatically return the value of the last expression.

ARRAYS

```
var arr [3]int

arr[1] = 123

b := [5]int{1, 2, 3, 4, 5}

fmt.Println(len(b))
```

You can declare and initialize an array in one line.

SLICES

```
var arr []int

b := []int{“cat”,”dog”,”mouse”,}

append(b, “pigeon”)

c := make([]string, len(b))
copy(c, b)
```

Unlike arrays, slices are typed only by the elements they contain (not the number of elements).

MAPS I

```
people := map[string]int{
  “Bob”: 35, “Richard”: 23, “Kate: 28,
}
// get
people[“Bob”] // prints 35
// set
people[“Olivia”] = 27
// delete
delete(people, “Bob”)
```

Go provides a built-in map type that implements a hash table.

MAPS II

```
// Initializes a map with space for 15 items
m := make(map[string]int32, 15)

// check if the items exists
r1, ok := m["route"]
if ok {
 // do something with value
}
```

STRUCTS I

```go
type pet struct {
  name string
  kind string
  age int
}

myCat := pet{ "Dino", "cat", 3}
yourDog := pet{
  name: "Barky",
  kind: "dog",
  age: 5,
}
```

Note that the last comma (trailing) is NOT optional

STRUCTS II

```
func (p *pet) sayName() {
    fmt.Printf("My name is %v\n", p.name)
}

myCat := pet{ "Dino", "cat", 3}

myCat.sayName() // prints "My name is Dino"
```

GOROUTINES

```
// To create a goroutine - just prefix with "go"

go func() {
    fmt.Println("printing")
}()
```

Note: Main func doesn’t wait for the ‘spawned’ goroutines to finish.

CHANNELS

```
func main() {
    messages := make(chan string)

    go func() { messages <- "ping" }()

    msg := <-messages
    fmt.Println(msg)
}

// this will print “ping” and exit.
```

Go provides a built-in map type that implements a hash table.

HELPFUL STRUCTURES

```
a := []int{12, 34, 45}
for i, num := range a {
    fmt.Println(num)
}

func twoSquares(a, b int) (int, int) {
    return a * a, b * b
}
```

Range, Multiple returns

PRINT FUNCTIONS

```
fmt.Printf(“Hello, My name is %v”, “Slim Shady”)
```

There are multiple “print verbs” available

SECTION SIX

Get Set Up

Get set up

1. Download Go: https://golang.org/dl
2. Install Go: https://golang.org/doc/install
3. Best Environments: a) VS Code + Go plugin b) JetBrains GoLand

SECTION SEVEN

Exercises

Exercises - Part I

1. Write a program that has a number (as a variable declared inside) and based on whether it’s an ‘even’ or ‘odd’ number, it prints out a message with which one it is. Once you’re done, make it so the program expects user to enter a number and once it has it, it prints out whether it was even or odd. (same idea but number will come from a user inputting it in command line - so you will be *scanning* for input (kind of like a prompt in JS)
2. Create a type ‘person’, with name, age, and profession. Then create a slice of people (with info filled in). Print a sentence about each of the people in this structure: “*Name* is *Age* years old. *Name* is a *Profession*.”
3. Create a program that contains a function that converts temperature from Celsius to Fahrenheit

Exercises - Part II

1. Write a program that from a slice of integers that contains certain numbers more than once (like [3, 4, 3, 5, 4, 7]) produces a slice of integers with only unique numbers (essentially, filtering out any duplicates)
2. Reverse a string
3. Create a program that on ‘go run main.go’ downloads a random image from this API: https://picsum.photos/200/300/?random
4. Write a program that takes in a string or a chunk of text (meaning a just a bigger string :)) - and creates a map (of char (‘rune’) to number, and it calculates how many times each character is present in the string. Then it should print out in the format of: “a: 32” “b: 7”
5. If you want to make it more interesting as a bonus you can lowercase everything and then get a real measure of any letter (and not have separate measures for ‘a’ and ‘A’ for example)

SECTION EIGHT

Example Snippets 🍀

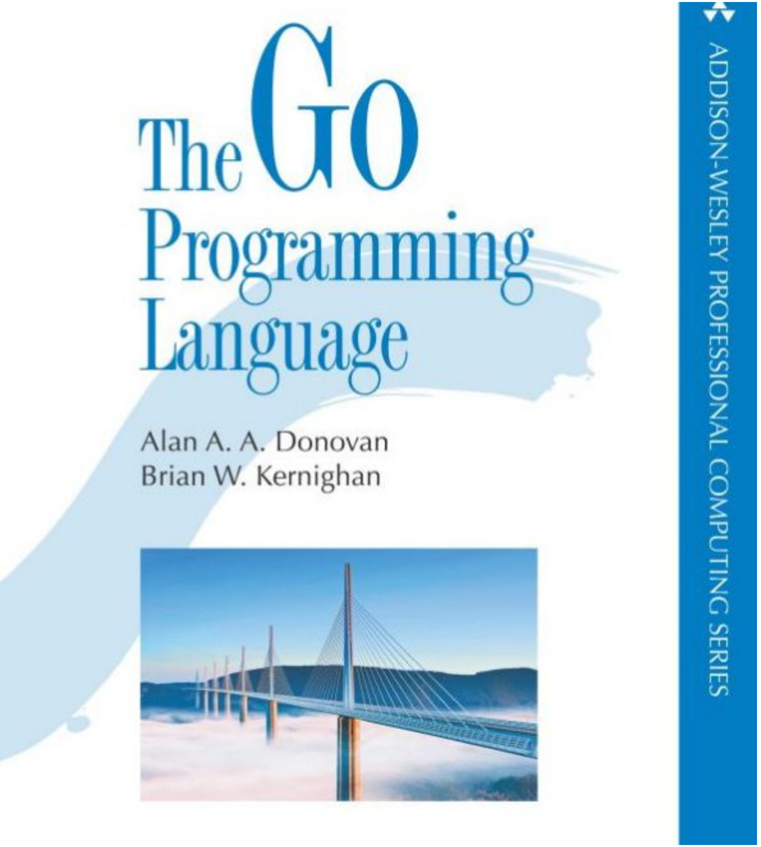
Examples

1. Interfaces example: https://github.com/kallaway/golang-snippets/blob/master/interfaces/shape/main.go
2. Save Images (error handling example): https://github.com/kallaway/golang-snippets/blob/master/scripts/save-image/main.go
3. Simple server: https://github.com/kallaway/golang-snippets/blob/master/web/001-hello-server/main.go

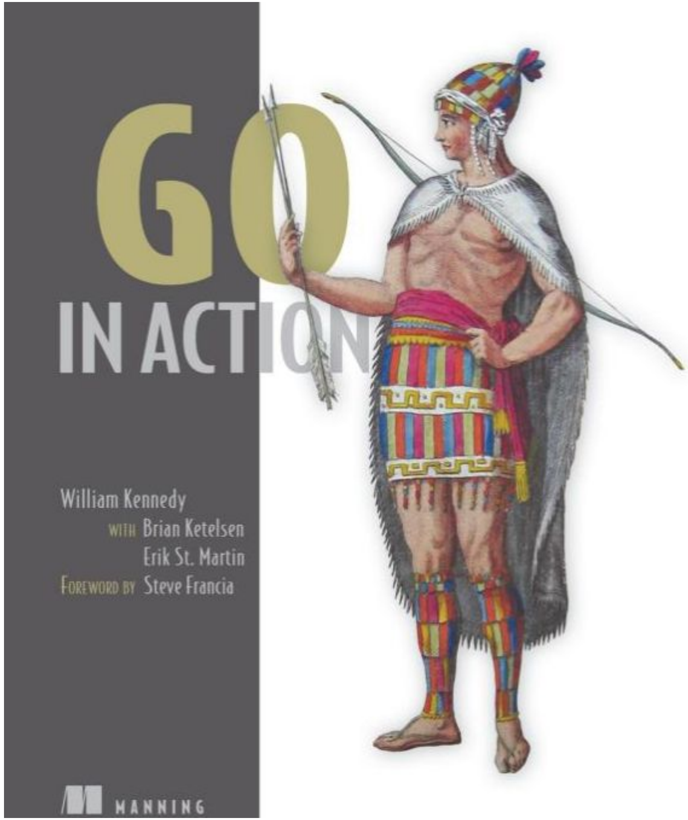
SECTION NINE

Next Steps

Books - The Go Programming Language


The Go Programming Language
Alan A. A. Donovan
Brian W. Kernighan
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Books - Go in Action



Courses

1. **Go: The Complete Developer's Guide (Golang) by Stephen Grider https://www.udemy.com/go-the-complete-developers-guide/**
2. Learn How To Code: Google's Go (golang) Programming Language by Todd McLeod https://www.udemy.com/learn-how-to-code/
3. **Web Development with Go: Learn to Create Real Web Apps in Go https://www.usegolang.com/**
4. Complete Go Bootcamp: Go from zero to hero (Golang) by Jose Portilla, Inanc Gumus https://www.udemy.com/learn-go-the-complete-bootcamp-course-golang/

Your Next Steps...

1. Tour of Go https://tour.golang.org
2. Go by Example https://gobyexample.com
3. Effective Go https://golang.org/doc/effective_go.html
4. Exercism http://exercism.io/languages/go/about
5. Gophercises https://gophercises.com
6. Everything else: https://github.com/avelino/awesome-go

Bonus for those who stuck around:

a) https://github.com/ashleymcnamara/gophers

b) https://gopherize.me