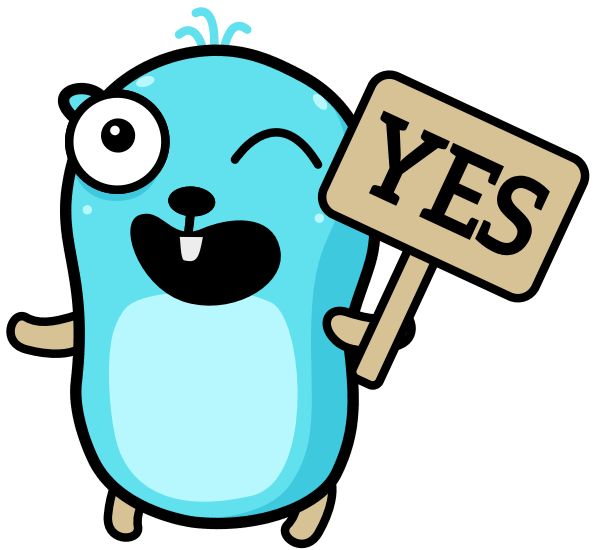
It’s time to Go

JUG Łódź, Bartosz Paluszkiewicz




DOCAPOSTE

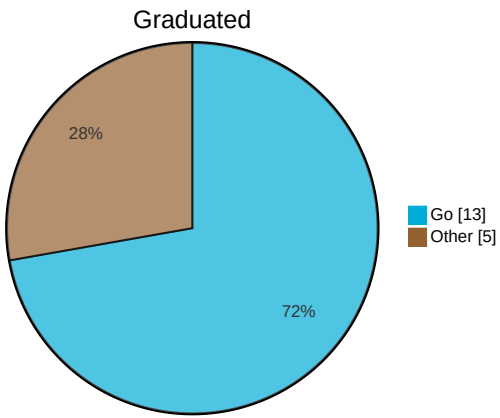
Gaming Secure Vault - international presence

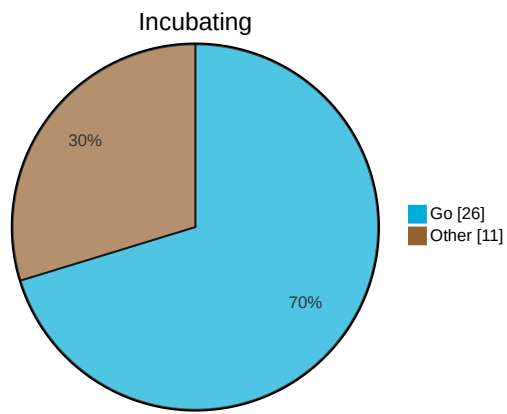
- Germany (2 regulations)
- France
- Colombia
- Bulgaria
- Portugal
- Switzerland
- Denmark
- Greece
- Netherlands
- Romania
- Argentina
- Spain

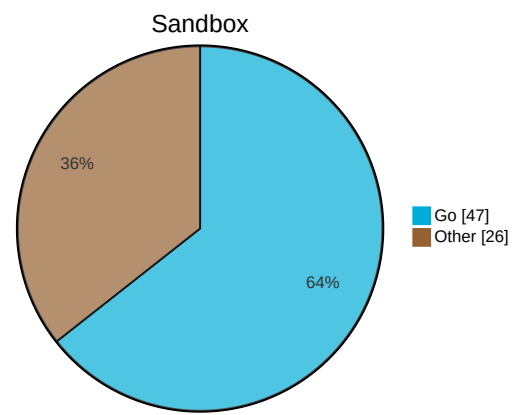
Did you Go there?

Is it used though?

28.08.2022, source: https://gloutnikov.com/post/cncf-language-stats/

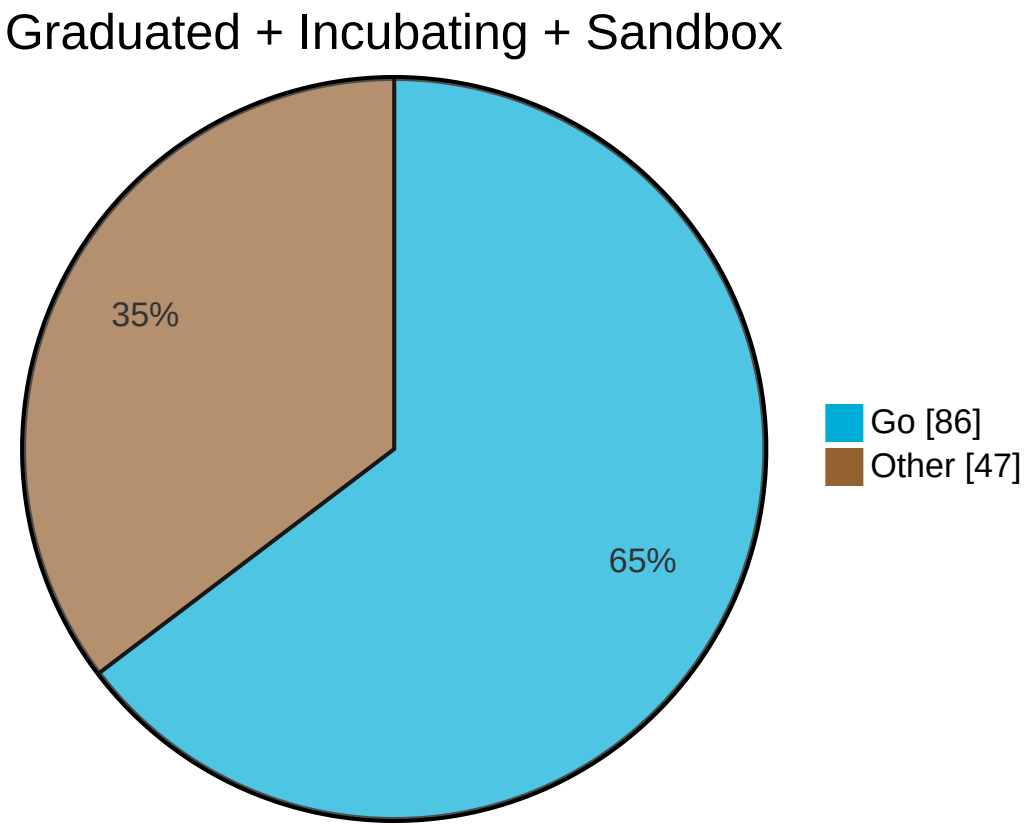






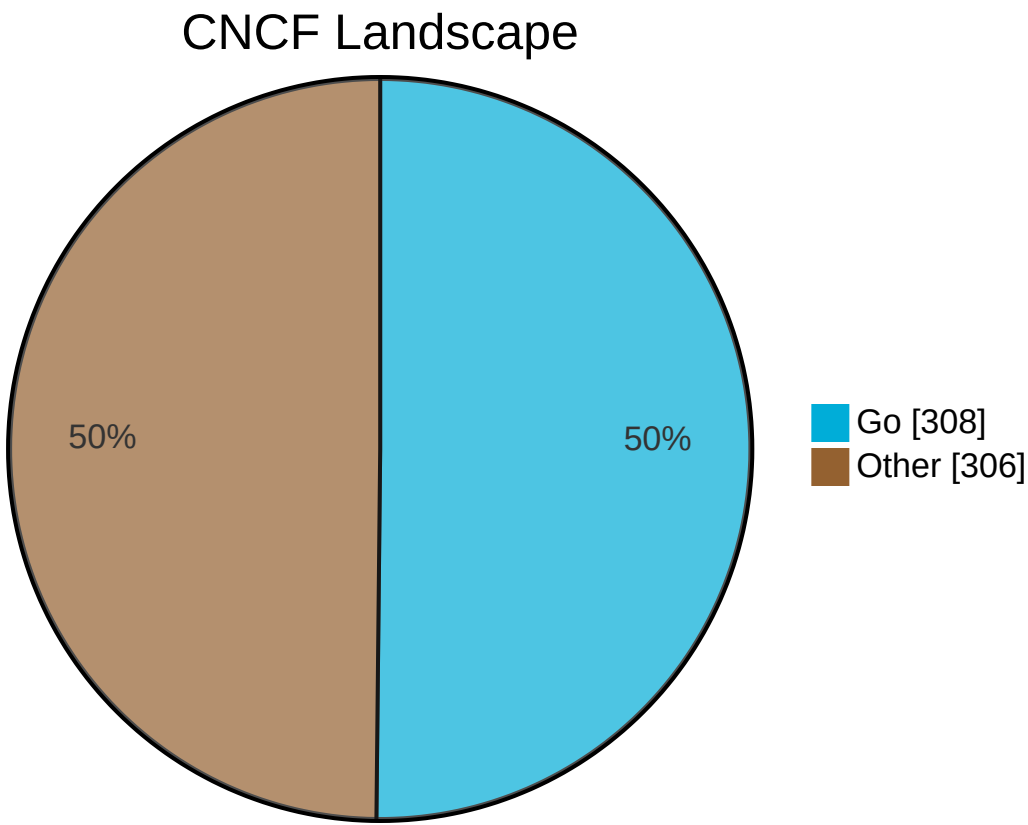
Is it used though?

28.08.2022, source: https://gloutnikov.com/post/cncf-language-stats/



Is it used though?

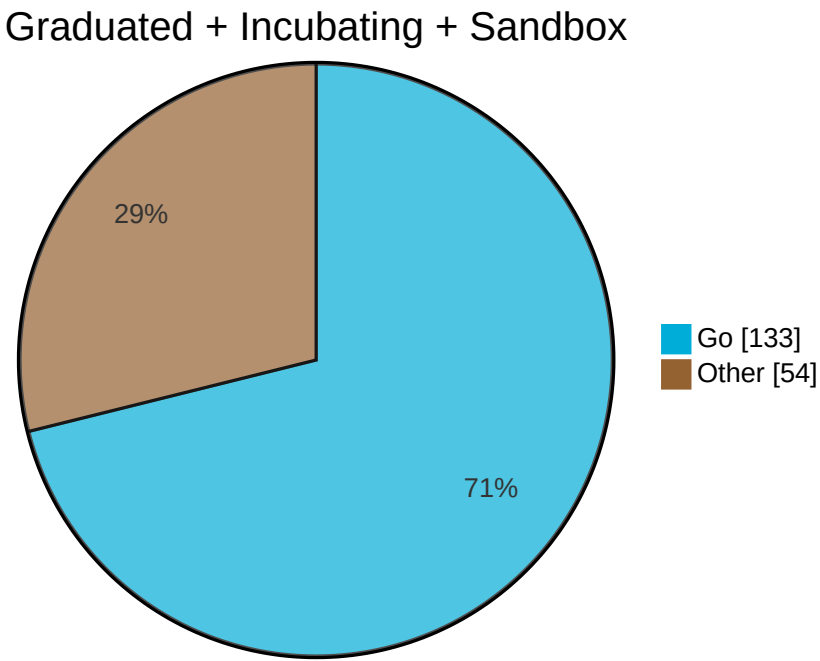
16.07.2023, source: https://jonathonhenderson.co.uk/2023/07/16/cncf-projects-by-language



Is it used though?

CNCF contributors

```
curl "https://raw.githubusercontent.com/cncf/tag-contributor-strategy/main/website/data/projects.json" \
| jq '[.[] | select(.language ≠ null)] | group_by(.language)[] | {key:.[0].language, count: length}'
```



What is Go?

1. Statically typed
2. Compiled (AOT)
3. Garbage collected
4. Highly concurrent
5. Simple (not always easy)
6. Fast
7. Open Source
8. Backed by Google

Go simple

```
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
```

```
abstract   continue   for          new         switch
assert     default    if           package     synchronized
boolean    do         goto         private     this
break      double     implements   protected   throw
byte       else       import       public      throws
case       enum       instanceof   return      transient
catch      extends    int          short       try
char       final      interface    static      void
class      finally    long         strictfp    volatile
const      float      native       super       while
_ (underscore)

exports      opens      requires     uses   yield
module       permits    sealed       var
non-sealed   provides   to           when
open         record     transitive   with
```

Go vs Java (tooling)

Go SDK

- go build
- go generate
- go test
- go tool cover
- go fmt
- go vet
- go get github.com/google/uuid
- go mod tidy

Go vs Java (tooling)

Experimental

- tools
  - godoc
  - gopls
  - imports
- vulncheck

Go vs Java (tooling)

External build tools

- make
- task

Let’s Go see the code

```
1  package main
2
3  func main() {
4    println("Hello, World!")
5  }
```

Let’s Go see the code

```
1  package main
2
3  func main() {
4    println("Hello, World!")
5  }
```

Let’s Go see the code

```
1  package main
2
3  func main() {
4    println("Hello, World!")
5  }
```

Let’s Go see the code

```
1  package main
2
3  func main() {
4    println("Hello, World!")
5  }
```

Let’s Go see the code

```
1  package main
2
3  func main() {
4    println("Hello, World!")
5  }
```

```
$ Hello, World!
```

Syntax 101

```
var go1 string = "gopher"
go2 := "gopher"
```

```
type User struct {
    login string
}
```

```
func login(login, password string) (User, error) {
  if login ≠ "admin" {
    err := fmt.Errorf("unknown user: %s", login)
    return User{}, err
}

  if password ≠ "admin" {
      return User{}, fmt.Errorf("wrong password")
  }

  u := User{login:login}
  return u, nil
}
```

Syntax 101

```
var go1 string = "gopher"
go2 := "gopher"
```

```
type User struct {
    login string
}
```

```
func login(login, password string) (User, error) {
  if login ≠ "admin" {
    err := fmt.Errorf("unknown user: %s", login)
    return User{}, err
}

  if password ≠ "admin" {
      return User{}, fmt.Errorf("wrong password")
  }

  u := User{login:login}
  return u, nil
}
```

Syntax 101

```
var go1 string = "gopher"
go2 := "gopher"
```

```
type User struct {
    login string
}
```

```
func login(login, password string) (User, error) {
  if login ≠ "admin" {
    err := fmt.Errorf("unknown user: %s", login)
    return User{}, err
}

  if password ≠ "admin" {
      return User{}, fmt.Errorf("wrong password")
  }

  u := User{login:login}
  return u, nil
}
```

Syntax 101

```
var go1 string = "gopher"
go2 := "gopher"
```

```
type User struct {
    login string
}
```

```
func login(login, password string) (User, error) {
  if login ≠ "admin" {
    err := fmt.Errorf("unknown user: %s", login)
    return User{}, err
}

  if password ≠ "admin" {
      return User{}, fmt.Errorf("wrong password")
  }

  u := User{login:login}
  return u, nil
}
```

Syntax 101

```
var go1 string = "gopher"
go2 := "gopher"
```

```
type User struct {
    login string
}
```

```
func login(login, password string) (User, error) {
  if login ≠ "admin" {
    err := fmt.Errorf("unknown user: %s", login)
    return User{}, err
}

  if password ≠ "admin" {
      return User{}, fmt.Errorf("wrong password")
  }

  u := User{login:login}
  return u, nil
}
```

Syntax 101

```
var go1 string = "gopher"
go2 := "gopher"
```

```
type User struct {
    login string
}
```

```
func login(login, password string) (User, error) {
  if login ≠ "admin" {
    err := fmt.Errorf("unknown user: %s", login)
    return User{}, err
}

  if password ≠ "admin" {
      return User{}, fmt.Errorf("wrong password")
  }

  u := User{login:login}
  return u, nil
}
```

Syntax 101

```
var go1 string = "gopher"
go2 := "gopher"
```

```
type User struct {
    login string
}
```

```
func login(login, password string) (User, error) {
  if login ≠ "admin" {
    err := fmt.Errorf("unknown user: %s", login)
    return User{}, err
}

  if password ≠ "admin" {
      return User{}, fmt.Errorf("wrong password")
  }

  u := User{login:login}
  return u, nil
}
```

Syntax 101

```
var go1 string = "gopher"
go2 := "gopher"
```

```
type User struct {
    login string
}
```

```
func login(login, password string) (User, error) {
  if login ≠ "admin" {
    err := fmt.Errorf("unknown user: %s", login)
    return User{}, err
}

  if password ≠ "admin" {
      return User{}, fmt.Errorf("wrong password")
  }

  u := User{login:login}
  return u, nil
}
```

Syntax 101

```
var go1 string = "gopher"
go2 := "gopher"
```

```
type User struct {
    login string
}
```

```
func login(login, password string) (User, error) {
  if login ≠ "admin" {
    err := fmt.Errorf("unknown user: %s", login)
    return User{}, err
}

  if password ≠ "admin" {
      return User{}, fmt.Errorf("wrong password")
  }

  u := User{login:login}
  return u, nil
}
```

Syntax 101

```
var go1 string = "gopher"
go2 := "gopher"
```

```
type User struct {
    login string
}
```

```
func login(login, password string) (User, error) {
  if login ≠ "admin" {
    err := fmt.Errorf("unknown user: %s", login)
    return User{}, err
}

  if password ≠ "admin" {
      return User{}, fmt.Errorf("wrong password")
  }

  u := User{login:login}
  return u, nil
}
```

Syntax 101

```
var go1 string = "gopher"
go2 := "gopher"
```

```
type User struct {
    login string
}
```

```
func login(login, password string) (User, error) {
  if login ≠ "admin" {
    err := fmt.Errorf("unknown user: %s", login)
    return User{}, err
}

  if password ≠ "admin" {
      return User{}, fmt.Errorf("wrong password")
  }

  u := User{login:login}
  return u, nil
}
```

Syntax 101

```
var go1 string = "gopher"
go2 := "gopher"
```

```
type User struct {
    login string
}
```

```
func login(login, password string) (User, error) {
  if login ≠ "admin" {
    err := fmt.Errorf("unknown user: %s", login)
    return User{}, err
}

  if password ≠ "admin" {
      return User{}, fmt.Errorf("wrong password")
  }

  u := User{login:login}
  return u, nil
}
```

Syntax 102

```
var int i = 1
var int8 i8 = 1
var int16 i16 = 1
var int32 i32 = 1
var int64 i64 = 1
var uint ui = 1
var uint8 ui8 = 1
var uint16 ui16 = 1
var uint32 ui32 = 1
var uint64 ui64 = 1
var uintptr up = 1
var byte b = 1 // alias for uint8
// represents a Unicode code point
var rune r = 'a' // alias for int32
```

```
var float32 f32 = 1.0
var float64 f64 = 1.0
```

```
var complex64 c64 = 9 + 11i
var complex128 c128 = complex(21, 37)
```

```
var bool b = true
var string s = "gopher" // alias for []byte
```

Syntax 102

```
var int i = 1
var int8 i8 = 1
var int16 i16 = 1
var int32 i32 = 1
var int64 i64 = 1
var uint ui = 1
var uint8 ui8 = 1
var uint16 ui16 = 1
var uint32 ui32 = 1
var uint64 ui64 = 1
var uintptr up = 1
var byte b = 1 // alias for uint8
// represents a Unicode code point
var rune r = 'a' // alias for int32
```

```
var float32 f32 = 1.0
var float64 f64 = 1.0
```

```
var complex64 c64 = 9 + 11i
var complex128 c128 = complex(21, 37)
```

```
var bool b = true
var string s = "gopher" // alias for []byte
```

Syntax 102

```
var int i = 1
var int8 i8 = 1
var int16 i16 = 1
var int32 i32 = 1
var int64 i64 = 1
var uint ui = 1
var uint8 ui8 = 1
var uint16 ui16 = 1
var uint32 ui32 = 1
var uint64 ui64 = 1
var uintptr up = 1
var byte b = 1 // alias for uint8
// represents a Unicode code point
var rune r = 'a' // alias for int32
```

```
var float32 f32 = 1.0
var float64 f64 = 1.0
```

```
var complex64 c64 = 9 + 11i
var complex128 c128 = complex(21, 37)
```

```
var bool b = true
var string s = "gopher" // alias for []byte
```

Syntax 102

```
var int i = 1
var int8 i8 = 1
var int16 i16 = 1
var int32 i32 = 1
var int64 i64 = 1
var uint ui = 1
var uint8 ui8 = 1
var uint16 ui16 = 1
var uint32 ui32 = 1
var uint64 ui64 = 1
var uintptr up = 1
var byte b = 1 // alias for uint8
// represents a Unicode code point
var rune r = 'a' // alias for int32
```

```
var float32 f32 = 1.0
var float64 f64 = 1.0
```

```
var complex64 c64 = 9 + 11i
var complex128 c128 = complex(21, 37)
```

```
var bool b = true
var string s = "gopher" // alias for []byte
```

Syntax 102

```
var int i = 1
var int8 i8 = 1
var int16 i16 = 1
var int32 i32 = 1
var int64 i64 = 1
var uint ui = 1
var uint8 ui8 = 1
var uint16 ui16 = 1
var uint32 ui32 = 1
var uint64 ui64 = 1
var uintptr up = 1
var byte b = 1 // alias for uint8
// represents a Unicode code point
var rune r = 'a' // alias for int32
```

```
var float32 f32 = 1.0
var float64 f64 = 1.0
```

```
var complex64 c64 = 9 + 11i
var complex128 c128 = complex(21, 37)
```

```
var bool b = true
var string s = "gopher" // alias for []byte
```

Syntax 102

```
var int i = 1
var int8 i8 = 1
var int16 i16 = 1
var int32 i32 = 1
var int64 i64 = 1
var uint ui = 1
var uint8 ui8 = 1
var uint16 ui16 = 1
var uint32 ui32 = 1
var uint64 ui64 = 1
var uintptr up = 1
var byte b = 1 // alias for uint8
// represents a Unicode code point
var rune r = 'a' // alias for int32
```

```
var float32 f32 = 1.0
var float64 f64 = 1.0
```

```
var complex64 c64 = 9 + 11i
var complex128 c128 = complex(21, 37)
```

```
var bool b = true
var string s = "gopher" // alias for []byte
```

Syntax 102

```
var int i = 1
var int8 i8 = 1
var int16 i16 = 1
var int32 i32 = 1
var int64 i64 = 1
var uint ui = 1
var uint8 ui8 = 1
var uint16 ui16 = 1
var uint32 ui32 = 1
var uint64 ui64 = 1
var uintptr up = 1
var byte b = 1 // alias for uint8
// represents a Unicode code point
var rune r = 'a' // alias for int32
```

```
var float32 f32 = 1.0
var float64 f64 = 1.0
```

```
var complex64 c64 = 9 + 11i
var complex128 c128 = complex(21, 37)
```

```
var bool b = true
var string s = "gopher" // alias for []byte
```

Syntax 102

```
var int i = 1
var int8 i8 = 1
var int16 i16 = 1
var int32 i32 = 1
var int64 i64 = 1
var uint ui = 1
var uint8 ui8 = 1
var uint16 ui16 = 1
var uint32 ui32 = 1
var uint64 ui64 = 1
var uintptr up = 1
var byte b = 1 // alias for uint8
// represents a Unicode code point
var rune r = 'a' // alias for int32
```

```
var float32 f32 = 1.0
var float64 f64 = 1.0
```

```
var complex64 c64 = 9 + 11i
var complex128 c128 = complex(21, 37)
```

```
var bool b = true
var string s = "gopher" // alias for []byte
```

Syntax 102

```
var int i = 1
var int8 i8 = 1
var int16 i16 = 1
var int32 i32 = 1
var int64 i64 = 1
var uint ui = 1
var uint8 ui8 = 1
var uint16 ui16 = 1
var uint32 ui32 = 1
var uint64 ui64 = 1
var uintptr up = 1
var byte b = 1 // alias for uint8
// represents a Unicode code point
var rune r = 'a' // alias for int32
```

```
var float32 f32 = 1.0
var float64 f64 = 1.0
```

```
var complex64 c64 = 9 + 11i
var complex128 c128 = complex(21, 37)
```

```
var bool b = true
var string s = "gopher" // alias for []byte
```

Syntax 103

```
people := map[string]Person{
    "admin": Person{"Gopher"},
}

for k, v := range people {
    fmt.Printf("%s: %s\n", k, v.Name)
}
```

```
switch os := runtime.GOOS; os {
  case "darwin":
     fmt.Println("Apple")
  case "linux":
     fmt.Println("Penguin")
  default:
     fmt.Println("Nobody cares")
}
```

Syntax 103

```
people := map[string]Person{
    "admin": Person{"Gopher"},
}

for k, v := range people {
    fmt.Printf("%s: %s\n", k, v.Name)
}
```

```
switch os := runtime.GOOS; os {
  case "darwin":
     fmt.Println("Apple")
  case "linux":
     fmt.Println("Penguin")
  default:
     fmt.Println("Nobody cares")
}
```

Syntax 103

```
people := map[string]Person{
    "admin": Person{"Gopher"},
}

for k, v := range people {
    fmt.Printf("%s: %s\n", k, v.Name)
}
```

```
switch os := runtime.GOOS; os {
  case "darwin":
     fmt.Println("Apple")
  case "linux":
     fmt.Println("Penguin")
  default:
     fmt.Println("Nobody cares")
}
```

Syntax 103

```
people := map[string]Person{
    "admin": Person{"Gopher"},
}

for k, v := range people {
    fmt.Printf("%s: %s\n", k, v.Name)
}
```

```
switch os := runtime.GOOS; os {
  case "darwin":
     fmt.Println("Apple")
  case "linux":
     fmt.Println("Penguin")
  default:
     fmt.Println("Nobody cares")
}
```

Syntax 103

```
people := map[string]Person{
    "admin": Person{"Gopher"},
}

for k, v := range people {
    fmt.Printf("%s: %s\n", k, v.Name)
}
```

```
switch os := runtime.GOOS; os {
  case "darwin":
     fmt.Println("Apple")
  case "linux":
     fmt.Println("Penguin")
  default:
     fmt.Println("Nobody cares")
}
```

Syntax 103

```
people := map[string]Person{
    "admin": Person{"Gopher"},
}

for k, v := range people {
    fmt.Printf("%s: %s\n", k, v.Name)
}
```

```
switch os := runtime.GOOS; os {
  case "darwin":
     fmt.Println("Apple")
  case "linux":
     fmt.Println("Penguin")
  default:
     fmt.Println("Nobody cares")
}
```

Syntax 103

```
people := map[string]Person{
    "admin": Person{"Gopher"},
}

for k, v := range people {
    fmt.Printf("%s: %s\n", k, v.Name)
}
```

```
switch os := runtime.GOOS; os {
  case "darwin":
     fmt.Println("Apple")
  case "linux":
     fmt.Println("Penguin")
  default:
     fmt.Println("Nobody cares")
}
```

Syntax 103

```
people := map[string]Person{
    "admin": Person{"Gopher"},
}

for k, v := range people {
    fmt.Printf("%s: %s\n", k, v.Name)
}
```

```
switch os := runtime.GOOS; os {
  case "darwin":
     fmt.Println("Apple")
  case "linux":
     fmt.Println("Penguin")
  default:
     fmt.Println("Nobody cares")
}
```

Syntax 103

```
people := map[string]Person{
    "admin": Person{"Gopher"},
}

for k, v := range people {
    fmt.Printf("%s: %s\n", k, v.Name)
}
```

```
switch os := runtime.GOOS; os {
  case "darwin":
     fmt.Println("Apple")
  case "linux":
     fmt.Println("Penguin")
  default:
     fmt.Println("Nobody cares")
}
```

Syntax 103

```
people := map[string]Person{
    "admin": Person{"Gopher"},
}

for k, v := range people {
    fmt.Printf("%s: %s\n", k, v.Name)
}
```

```
switch os := runtime.GOOS; os {
  case "darwin":
     fmt.Println("Apple")
  case "linux":
     fmt.Println("Penguin")
  default:
     fmt.Println("Nobody cares")
}
```

Syntax 103

```
people := map[string]Person{
    "admin": Person{"Gopher"},
}

for k, v := range people {
    fmt.Printf("%s: %s\n", k, v.Name)
}
```

```
switch os := runtime.GOOS; os {
  case "darwin":
     fmt.Println("Apple")
  case "linux":
     fmt.Println("Penguin")
  default:
     fmt.Println("Nobody cares")
}
```

Syntax 104

```
const admin = "admin"
people := map[string]Person{
  admin: Person{"Gopher"},
}

if adm, ok := people[admin]; ok {
    fmt.Printf("Admin is here, %s\n", adm.Name)
} else {
    people[admin] = Person{"default"}
}
```

```
type Reader interface {
    Read(b []byte) (n int, err error)
}
```

```
type HandlerFunc func (r ResponseWriter, w *Request)
```

Syntax 104

```
const admin = "admin"
people := map[string]Person{
  admin: Person{"Gopher"},
}

if adm, ok := people[admin]; ok {
    fmt.Printf("Admin is here, %s\n", adm.Name)
} else {
    people[admin] = Person{"default"}
}
```

```
type Reader interface {
    Read(b []byte) (n int, err error)
}
```

```
type HandlerFunc func (r ResponseWriter, w *Request)
```

Syntax 104

```
const admin = "admin"
people := map[string]Person{
  admin: Person{"Gopher"},
}

if adm, ok := people[admin]; ok {
    fmt.Printf("Admin is here, %s\n", adm.Name)
} else {
    people[admin] = Person{"default"}
}
```

```
type Reader interface {
    Read(b []byte) (n int, err error)
}
```

```
type HandlerFunc func (r ResponseWriter, w *Request)
```

Syntax 104

```
const admin = "admin"
people := map[string]Person{
  admin: Person{"Gopher"},
}

if adm, ok := people[admin]; ok {
    fmt.Printf("Admin is here, %s\n", adm.Name)
} else {
    people[admin] = Person{"default"}
}
```

```
type Reader interface {
    Read(b []byte) (n int, err error)
}
```

```
type HandlerFunc func (r ResponseWriter, w *Request)
```

Syntax 104

```
const admin = "admin"
people := map[string]Person{
  admin: Person{"Gopher"},
}

if adm, ok := people[admin]; ok {
    fmt.Printf("Admin is here, %s\n", adm.Name)
} else {
    people[admin] = Person{"default"}
}
```

```
type Reader interface {
    Read(b []byte) (n int, err error)
}
```

```
type HandlerFunc func (r ResponseWriter, w *Request)
```

Syntax 104

```
const admin = "admin"
people := map[string]Person{
  admin: Person{"Gopher"},
}

if adm, ok := people[admin]; ok {
    fmt.Printf("Admin is here, %s\n", adm.Name)
} else {
    people[admin] = Person{"default"}
}
```

```
type Reader interface {
    Read(b []byte) (n int, err error)
}
```

```
type HandlerFunc func (r ResponseWriter, w *Request)
```

Syntax 104

```
const admin = "admin"
people := map[string]Person{
  admin: Person{"Gopher"},
}

if adm, ok := people[admin]; ok {
    fmt.Printf("Admin is here, %s\n", adm.Name)
} else {
    people[admin] = Person{"default"}
}
```

```
type Reader interface {
    Read(b []byte) (n int, err error)
}
```

```
type HandlerFunc func (r ResponseWriter, w *Request)
```

Syntax 104

```
const admin = "admin"
people := map[string]Person{
  admin: Person{"Gopher"},
}

if adm, ok := people[admin]; ok {
    fmt.Printf("Admin is here, %s\n", adm.Name)
} else {
    people[admin] = Person{"default"}
}
```

```
type Reader interface {
    Read(b []byte) (n int, err error)
}
```

```
type HandlerFunc func (r ResponseWriter, w *Request)
```

Let’s Go see some REAL code

Go proverbs

https://go-proverbs.github.io/

- Don’t communicate by sharing memory, share memory by communicating.
- Concurrency is not parallelism.
- Channels orchestrate; mutexes serialize.
- The bigger the interface, the weaker the abstraction.
- Make the zero value useful.
- interface{} says nothing.
- Gofmt’s style is no one’s favorite, yet gofmt is everyone’s favorite.
- A little copying is better than a little dependency.
- Syscall must always be guarded with build tags.
- Cgo must always be guarded with build tags.
- Cgo is not Go.
- With the unsafe package there are no guarantees.
- Clear is better than clever.
- Reflection is never clear.
- Errors are values.
- Don’t just check errors, handle them gracefully.
- Design the architecture, name the components, document the details.
- Documentation is for users.
- Don’t panic.

Don’t communicate by sharing memory, share memory by communicating.

Don’t communicate by sharing memory, share memory by communicating.

```
c := make(chan string)
go func () {
c ← "ping"
}()
println(←c)
```

Don’t communicate by sharing memory, share memory by communicating.

```
c := make(chan string)
go func () {
c ← "ping"
}()
println(←c)
```

Don’t communicate by sharing memory, share memory by communicating.

```
c := make(chan string)
go func () {
c ← "ping"
}()
println(←c)
```

Don’t communicate by sharing memory, share memory by communicating.

```
c := make(chan string)
go func () {
c ← "ping"
}()
println(←c)
```

Don’t communicate by sharing memory, share memory by communicating.

```
c := make(chan string)
go func () {
c ← "ping"
}()
println(←c)
```

The bigger the interface, the weaker the abstraction.

```
type Reader interface {
Read(p []byte) (n int, err error)
}

type Closer interface {
Close() error
}

type ReadCloser interface {
Reader
Closer
}
```

Credits

- Gophers illustrations by MariaLetta
- Copilot by GitHub

Further reading

- A tour of Go
- Effective Go
- The Go blog
- YouTube Go Class by Matt Holiday
- 100 Go mistakes and how to avoid them
- Uber Go guide
- Go style guide
- Three Dots Tech blog