Daniel Martí

Maintainer at cuelang.org

CUE

Go APIs from 1.0 to 1.40



The Go Programming Language

Documents | Packages | The Project | Help | Blog | Search

Try Go

Pop-out

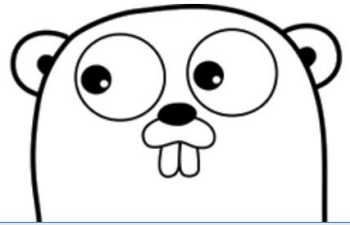
```
// You can edit this code!
// Click here and start typing.
package main

import "fmt"

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

Hello, World! | Run | Share | Tour

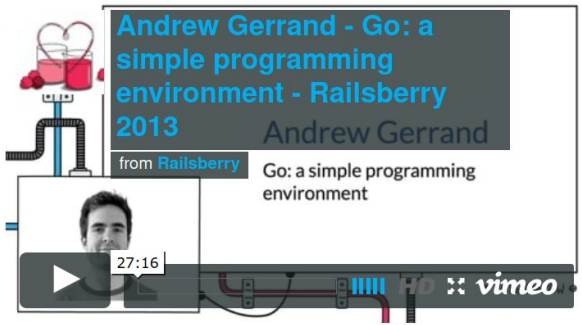
Go is an open source programming language that makes it easy to build simple, reliable, and efficient software.



Download Go

Binary distributions available for
Linux, Mac OS X, Windows, and more.

Featured video



Featured articles

Read more

2012 -> 2026

How have APIs evolved between Go 1.0 and 1.27?



Context (Go 1.7)

```
func DoWork(timeout time.Duration, cancel <-chan struct{})
error

type Worker struct {...}
func NewWorker(timeout time.Duration) *Worker
func (w *Worker) Stop()
```

Context (Go 1.7)

```
func DoWork(timeout time.Duration, cancel <-chan struct{})
error

type Worker struct {...}
func NewWorker(timeout time.Duration) *Worker
func (w *Worker) Stop()


func DoWork(ctx context.Context) error
…
ctx, cancel := context.WithTimeout(ctx, time.Second)
DoWork(ctx)
```

Error wrapping (Go 1.13)

```
return nil, io.ErrUnexpectedEOF
…
if err == io.ErrUnexpectedEOF { … }
```

Error wrapping (Go 1.13)

```
return nil, io.ErrUnexpectedEOF
…
if err == io.ErrUnexpectedEOF { … }
```

- github.com/pkg/errors.Wrap
- github.com/juju/errors.Annotate
- github.com/hashicorp/errwrap.Wrapf

Error wrapping (Go 1.13)

```
return nil, io.ErrUnexpectedEOF
…
if err == io.ErrUnexpectedEOF { … }
```

- github.com/pkg/errors.Wrap
- github.com/juju/errors.Annotate
- github.com/hashicorp/errwrap.Wrapf

```
return nil, errors.Wrap(io.ErrUnexpectedEOF,
    "expected a signature")
…
switch errors.Cause(err).(type) { … }
```

Error wrapping (Go 1.13)

```
fmt.Errorf("expected a signature: %w", io.ErrUnexpectedEOF)
…
if errors.Is(err, io.ErrUnexpectedEOF) { … }
```

Error wrapping (Go 1.13)

```
package fs

type PathError struct {
    Op   string
    Path string
    Err  error
}

func (e *PathError) Error() string
func (e *PathError) Unwrap() error
```

Multiple errors (Go 1.20)

```
func Process(items []Items) error
    stops at the first error? discards other errors?
```

Multiple errors (Go 1.20)

```
func Process(items []Items) error
    stops at the first error? discards other errors?

func Process(items []Items) []error
    cannot "if err != nil". non-nil empty slices…
```

Multiple errors (Go 1.20)

```
func Process(items []Items) error
    stops at the first error? discards other errors?

func Process(items []Items) []error
    cannot "if err != nil". non-nil empty slices…

github.com/hashicorp/go-multierror
    same as before; non-standard
```

Multiple errors (Go 1.20)

```
func Process(items []Items) error
    stops at the first error? discards other errors?

func Process(items []Items) []error
    cannot "if err != nil". non-nil empty slices…

github.com/hashicorp/go-multierror
    same as before; non-standard

return errors.Join(errs...)
    or…
func (ProcessError) Unwrap() []error { … }
```

Generics (Go 1.18)

```
type Stack struct {
    items []interface{}
}

func (s *Stack) Push(item interface{})
func (s *Stack) Pop() (interface{}, error)

var st Stack
st.Push("foo")
x, err := st.Pop()
s := x.(string)
```

Generics (Go 1.18)

```
type Stack[T any] struct {
    items []T
}

func (s *Stack[T]) Push(item T)
func (s *Stack[T]) Pop() (T, error)

var st Stack[string]
st.Push("foo")
s, err := st.Pop()
```

Generic methods (Go 1.27)

```
type Stack[T any] struct {
    items []T
}

func (s *Stack[T]) Push(item T)
func (s *Stack[T]) Pop() (T, error)
func (s *Stack[T]) Map[U any](f func(T) U) Stack[U]
```

Iterators (Go 1.23)

```
func (DB) Usernames() []string
    scales very poorly
```

Iterators (Go 1.23)

```
func (DB) Usernames() []string
    scales very poorly

func (DB) Usernames() <-chan string
    reader cannot cancel, context switching overhead
```

Iterators (Go 1.23)

```
func (DB) Usernames() []string
    scales very poorly

func (DB) Usernames() <-chan string
    reader cannot cancel, context switching overhead

func (DB) Usernames() StringIterator
    calling Next/Get methods is tedious and error prone
```

Iterators (Go 1.23)

```
func (DB) Usernames() []string
    scales very poorly

func (DB) Usernames() <-chan string
    reader cannot cancel, context switching overhead

func (DB) Usernames() StringIterator
    calling Next/Get methods is tedious and error prone

func (DB) Usernames(yield func(string))
    harder to handle errors and control flow
```

Iterators (Go 1.23)

```
type Seq[V any] func(yield func(V) bool)

func (DB) Usernames() iter.Seq[string]


for username := range db.Usernames() {
    if username == "john" {
        break
    }
}
```

Iterators (Go 1.23)

```
type Seq[V any] func(yield func(V) bool)

func (DB) Usernames() iter.Seq[string]


for username := range db.Usernames() {
    if username == "john" {
        break
    }
}
```

Iterators (Go 1.23)

```
type Seq[V any] func(yield func(V) bool)

func (DB) Usernames() iter.Seq[string]


for username := range db.Usernames() {
    if username == "john" {
        break
    }
}
```

Iterators (Go 1.23)

```
type Seq[V any] func(yield func(V) bool)

func (DB) Usernames() iter.Seq[string]


for username := range db.Usernames() {
    if username == "john" {
        break
    }
}
```

2026 -> 2033

How will APIs evolve
from Go 1.27 to 1.33?

All **speculation** beyond here!
Based on GitHub proposals
and personal wishes.



- Type inferred composite literals (#12854, Go 1.28)

```
func NewWorker(parallel int) *Worker

w := pkg.NewWorker(10)
```

- Type inferred composite literals (#12854, Go 1.28)

```
func NewWorker(parallel int, log io.Writer) *Worker

w := pkg.NewWorker(10)
```

- Type inferred composite literals (#12854, Go 1.28)

```
type Config struct { … }
func NewWorker(cfg Config) *Worker

w := pkg.NewWorker(pkg.Config{ … })
```

- Type inferred composite literals (#12854, Go 1.28)

```
type Config struct { … }
func NewWorker(cfg Config) *Worker

w := pkg.NewWorker(pkg.Config{ … })
withDefaults := pkg.NewWorker(pkg.Config{})
```

Type inferred composite literals (#12854, Go 1.28)

```
type Config struct { … }
func NewWorker(cfg Config) *Worker

w := pkg.NewWorker(pkg.Config{ … })
withDefaults := pkg.NewWorker(pkg.Config{})
explicitZero := pkg.NewWorker(pkg.Config{Parallel: 0})
```

- Type inferred composite literals (#12854, Go 1.28)

```
type Config struct { … }
func NewWorker(cfg Config) *Worker

w := pkg.NewWorker(pkg.Config{ … })
withDefaults := pkg.NewWorker(pkg.Config{})
explicitZero := pkg.NewWorker(pkg.Config{Parallel: 0})

type Option func(*Worker)
func NewWorker(opts ...Option) *Worker // json/v2 does this!

withDefaults := pkg.NewWorker()
explicitZero := pkg.NewWorker(pkg.WithParallel(0))
```

- Type inferred composite literals (#12854, Go 1.28)

```
type Config struct { … }
func NewWorker(cfg Config) *Worker

w := pkg.NewWorker(pkg.Config{ … })
withDefaults := pkg.NewWorker({})
explicitZero := pkg.NewWorker({Parallel: 0})
```

Type inferred composite literals (#12854, Go 1.28)

```
type Config struct { … }
func NewWorker(cfg Config) *Worker

w := pkg.NewWorker(pkg.Config{ … })
withDefaults := pkg.NewWorker({})
explicitZero := pkg.NewWorker({Parallel: 0})
unbounded := pkg.NewWorker({Parallel: -1})
```

- Type inferred composite literals (#12854, Go 1.28)

```
func GetStats() (KernelStats, RuntimeStats, error)

return KernelStats{}, RuntimeStats{}, fmt.Errorf("message…")
```

- Type inferred composite literals (#12854, Go 1.28)

```
func GetStats() (KernelStats, RuntimeStats, error)

return KernelStats{}, RuntimeStats{}, fmt.Errorf("message…")


func GetStats() (*KernelStats, *RuntimeStats, error)

return nil, nil, fmt.Errorf("message…")
```

- Type inferred composite literals (#12854, Go 1.28)

```
func GetStats() (KernelStats, RuntimeStats, error)

return KernelStats{}, RuntimeStats{}, fmt.Errorf("message…")


return {}, {}, fmt.Errorf("message…")
```

- "new" builtin on expressions (Go 1.26)

```
type Config struct { Parallel int }
func NewWorker(cfg Config) *Worker

w := pkg.NewWorker(pkg.Config{ … })
explicitZero := pkg.NewWorker({Parallel: 0})
```

- "new" builtin on expressions (Go 1.26)

```
type Config struct { Parallel *int }
func NewWorker(cfg Config) *Worker

w := pkg.NewWorker(pkg.Config{ … })
unbounded := 0
explicitZero := pkg.NewWorker({Parallel: &unbounded})
```

- "new" builtin on expressions (Go 1.26)

```
type Config struct { Parallel *int }
func NewWorker(cfg Config) *Worker

w := pkg.NewWorker(pkg.Config{ … })
explicitZero := pkg.NewWorker({Parallel: new(0)})
```

"new" builtin on expressions (Go 1.26)

```
type Config struct { Parallel *int }
func NewWorker(cfg Config) *Worker

w := pkg.NewWorker(pkg.Config{ … })
explicitZero := pkg.NewWorker({Parallel: new(0)})
```

- Field doubles in size (pointer + int)
- Indirection jumps around memory
- Adds edge cases like nil pointer dereferencing

- Sum types as union types (#76920)

```
type Expr interface {
    exprNode()
}
func (CallExpr) exprNode() {}
func (FuncLit)  exprNode() {}
...
```

- Sum types as union types (#76920)

```
type Expr interface {
    exprNode()
}
func (CallExpr) exprNode() {}
func (FuncLit)  exprNode() {}
…

switch expr := expr.(type) {
case *ast.CallExpr:
case *ast.FuncLit:
… // no compiler help, no exhaustive switches
}
```

- Sum types as union types (#76920)

```
type Expr union {
    CallExpr *CallExpr
    FuncLit  *FuncLit
    …
}

switch expr := expr.(union) {
case CallExpr:
case FuncLit:
}
```

Sum types as union types (#76920)

```
type Direction int

const (
    North Direction = iota // 0
    South                  // 1
    East                   // 2
    West                   // 3
)
```

Sum types as union types (#76920)

```
type Direction int

const (
    North Direction = iota // 0
    South                  // 1
    East                   // 2
    West                   // 3
)

var d Direction // = 0
```

Sum types as union types (#76920)

```
type Direction int

const (
    NoDirection Direction = iota // 0
    North                        // 1
    South                        // 2
    East                         // 3
    West                         // 4
)

var d Direction // = 0
var d2 Direction = 50
```

Sum types as union types (#76920)

```
type Direction union {
    North, South, East, West struct{}
}

var d Direction // = North
var d2 Direction = 50 // compile error


    And also the compiler's help with switches!
```

- Sum types as union types (#76920)

```
type Maybe[T any] union {
    Unset struct{}
    Set   T
}
```

Sum types as union types (#76920)

```
type Maybe[T any] union {
    Unset struct{}
    Set   T
}


type Config struct { Parallel cmp.Maybe[int] }
func NewWorker(cfg Config) *Worker

w := pkg.NewWorker(pkg.Config{ … })
explicitZero := pkg.NewWorker({Parallel: Maybe[int]{Set: 0}})
```

Sum types as union types (#76920)

```
type Maybe[T any] union {
    Unset struct{}
    Set   T
}


type Config struct { Parallel cmp.Maybe[int] }
func NewWorker(cfg Config) *Worker

w := pkg.NewWorker(pkg.Config{ … })
explicitZero := pkg.NewWorker({Parallel: {Set: 0}})
```

- Sum types as union types (#76920)

```
package os

func LookupEnv(key string)  (string, bool)

    if val, ok := os.LookupEnv("FOO"); ok { … }
```

- Sum types as union types (#76920)

```
package os

func LookupEnv(key string)  (string, bool)

    if val, ok := os.LookupEnv("FOO"); ok { … }


func LookupEnv(key string) cmp.Maybe[string]

    if val, ok := os.LookupEnv("FOO").Set; ok { … }
```

Typed struct tags (#74472)

```
type Before struct {
    A Foo       `json:",omitzero"`
    B Bar       `json:"b,case:ignore"`
    C time.Time `json:",format:'2006-01-02'"`
}
```

Typed struct tags (#74472)

```
type Before struct {
    A Foo       `json:",omitzero"`
    B Bar       `json:"b,case:ignore"`
    C time.Time `json:",format:'2006-01-02'"`
}
```

- Tricky syntax inside strings
- No compile time help, no LSP support
- Lack of namespacing (json v1? v2? v3…?)

Typed struct tags (#74472)

```
type Before struct {
    A Foo       `json:",omitzero"`
    B Bar       `json:"b,case:ignore"`
    C time.Time `json:",format:'2006-01-02'"`
}

import "encoding/json/v2" // for types and consts
type After struct {
    A Foo       {json.OmitZero}
    B Bar       {json.Name("B"), json.IgnoreCase}
    C time.Time {json.Format("2006-01-02")}
}
```

Short func literals (#21498)

```
func SortFunc[S ~[]E, E any](x S, cmp func(a, b E) int)

people := []Person{ … }
slices.SortFunc(people, func(a, b Person) int {
    return a.Age < b.Age
})
```

Short func literals (#21498)

```
func SortFunc[S ~[]E, E any](x S, cmp func(a, b E) int)

people := []Person{ … }
slices.SortFunc(people, func(a, b Person) int {
    return a.Age < b.Age
})
```

Short func literals (#21498)

```
func SortFunc[S ~[]E, E any](x S, cmp func(a, b E) int)

people := []Person{ … }
slices.SortFunc(people, func(a, b Person) int {
    return a.Age < b.Age
})
```

Short func literals (#21498)

```
func SortFunc[S ~[]E, E any](x S, cmp func(a, b E) int)

people := []Person{ … }
slices.SortFunc(people, (a, b) => a.Age < b.Age)
```

Short func literals (#21498)

```
func SortFunc[S ~[]E, E any](x S, cmp func(a, b E) int)

people := []Person{ … }
slices.SortFunc(people, (a, b) => a.Age < b.Age)

slices.SortFunc(people, (a, b) -> { a.Age < b.Age })
```

Short func literals (#21498)

```
func SortFunc[S ~[]E, E any](x S, cmp func(a, b E) int)

people := []Person{ … }
slices.SortFunc(people, (a, b) => a.Age < b.Age)

slices.SortFunc(people, (a, b) -> { a.Age < b.Age })

slices.SortFunc(people, \(a, b) { a.Age < b.Age })
```

Short func literals (#21498)

```
func SortFunc[S ~[]E, E any](x S, cmp func(a, b E) int)

people := []Person{ … }
slices.SortFunc(people, (a, b) => a.Age < b.Age)

slices.SortFunc(people, (a, b) -> { a.Age < b.Age })

slices.SortFunc(people, \(a, b) { a.Age < b.Age })

slices.SortFunc(people, func |a, b| int { a.Age < b.Age })
```

Short func literals (#21498)



```
func SortFunc[S ~[]E, E any](x S, cmp func(a

people := []Person{ … }
slices.SortFunc(people, (a, b) => a.Age < b.Age)

slices.SortFunc(people, (a, b) -> { a.Age < b.Age })

slices.SortFunc(people, \(a, b) { a.Age < b.Age })

slices.SortFunc(people, func |a, b| int { a.Age < b.Age })
```

Short func literals (#21498)

```
package iter
func (Seq[T]) Map[S any](f func(T) S) Seq[S]
func (Seq[T]) Filter(p func(T) bool) Seq[T]
```

Short func literals (#21498)

```
package iter
func (Seq[T]) Map[S any](f func(T) S) Seq[S]
func (Seq[T]) Filter(p func(T) bool) Seq[T]

package slices
func Map[T1, T2 any](s []T1, f func(T1) T2) []T2
func Filter[T any](s []T, keep func(T) bool) []T
```

Short func literals (#21498)

```
package iter
func (Seq[T]) Map[S any](f func(T) S) Seq[S]
func (Seq[T]) Filter(p func(T) bool) Seq[T]

package slices
func Map[T1, T2 any](s []T1, f func(T1) T2) []T2
func Filter[T any](s []T, keep func(T) bool) []T

package iter
func MergeFunc[V any](x, y iter.Seq[V], f func(V, V) int) iter.Seq[V]
```

Short func literals (#21498)

```
package iter
func (Seq[T]) Map[S any](f func(T) S) Seq[S]
func (Seq[T]) Filter(p func(T) bool) Seq[T]

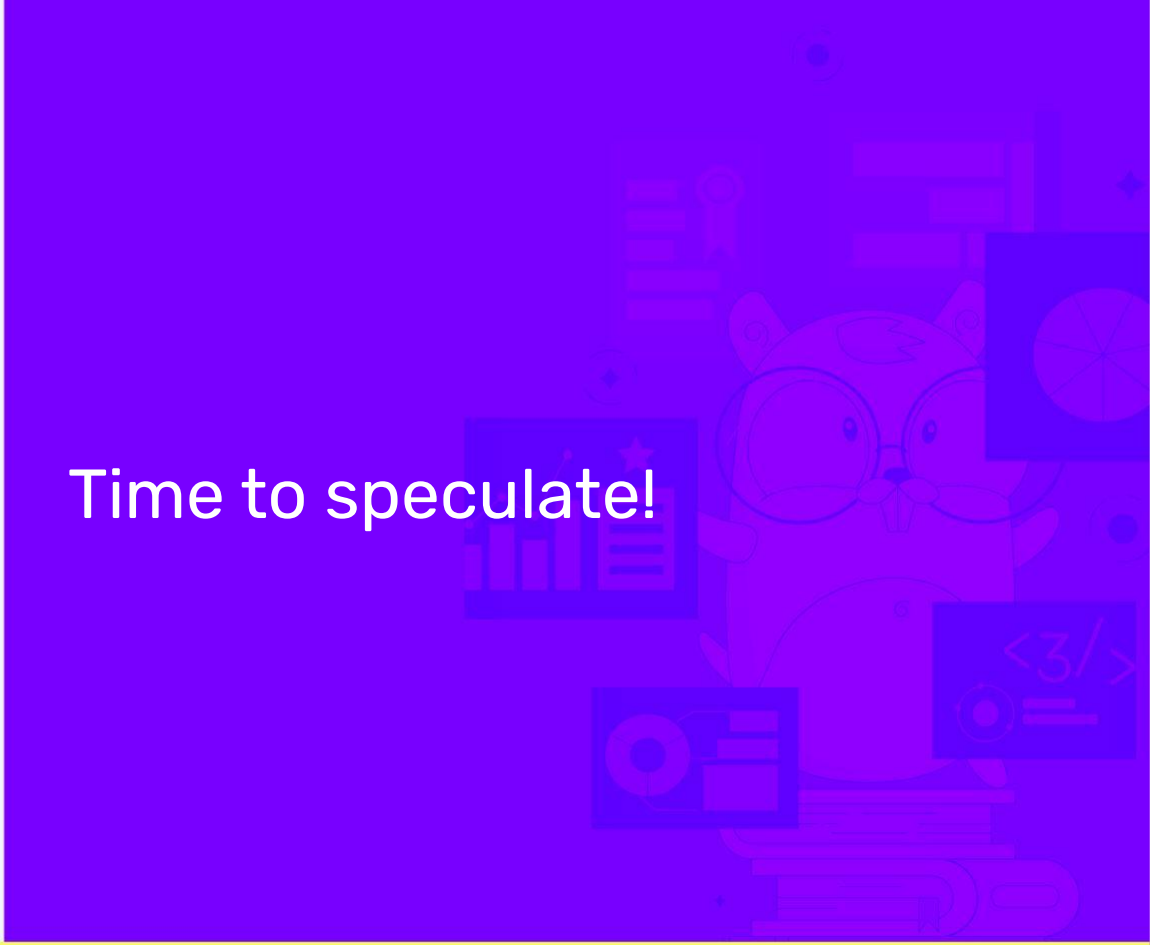
package slices
func Map[T1, T2 any](s []T1, f func(T1) T2) []T2
func Filter[T any](s []T, keep func(T) bool) []T

package iter
func MergeFunc[V any](x, y iter.Seq[V], f func(V, V) int) iter.Seq[V]
```



API contracts

Time to speculate!



API contracts

```
package io

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

API contracts

```
package io

type Reader interfa
    Read(p []byte)
}
```

type Reader

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

Reader is the interface that wraps the basic Read method.

Read reads up to len(p) bytes into p. It returns the number of bytes read (0 <= n <= len(p)) and any error encountered. Even if Read returns n < len(p), it may use all of p as scratch space during the call. If some data is available but not len(p) bytes, Read conventionally returns what is available instead of waiting for more.

When Read encounters an error or end-of-file condition after successfully reading n > 0 bytes, it returns the number of bytes read. It may return the (non-nil) error from the same call or return the error (and n == 0) from a subsequent call. An instance of this general case is that a Reader returning a non-zero number of bytes at the end of the input stream may return either err == EOF or err == nil. The next Read should return 0, EOF.

Callers should always process the n > 0 bytes returned before considering the error err. Doing so correctly handles I/O errors that happen after reading some bytes and also both of the allowed EOF behaviors.

If len(p) == 0, Read should always return n == 0. It may return a non-nil error if some error condition is known, such as EOF.

Implementations of Read are discouraged from returning a zero byte count with a nil error, except when len(p) == 0. Callers should treat a return of 0 and nil as indicating that nothing happened; in particular it does not indicate EOF.

Implementations must not retain p.

API contracts

```
package io

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

- Read must return EOF itself, not an error wrapping EOF
- Functions should return EOF only to signal a graceful end of input
- it may use all of p as scratch space during the call
- Implementations must not retain p

API contracts - type prefixes?

```
package io

func Copy(dst nonnil Writer, src nonnil Reader) (int64, error)

type Writer interface {
    Write(p readonly []byte) (n int, err error)
}
```

API contracts - type prefixes?

```
package io

func Copy(dst nonnil Writer, src nonnil Reader) (int64, error)

type Writer interface {
    Write(p readonly []byte) (n int, err error)
}


package os

func Open(name string) (mustclose *File, error)
```

API contracts - RFC key words?

```
// Reader is the interface that wraps the basic Read method.
//
// Read MUST return EOF itself…
// Read SHOULD return EOF to signal…
//
// Read MAY use all of p as scratch space…
// Read MUST NOT retain p…
type Reader interface {
    Read(p []byte) (n int, err error)
}
```

- "go test" analyzers (#82464)

```
//go:build goanalysis

package io

import (
    //go:analyze importers check.Reader
    _ "io/internal/check"
)
```

- "go test" analyzers (#82464)

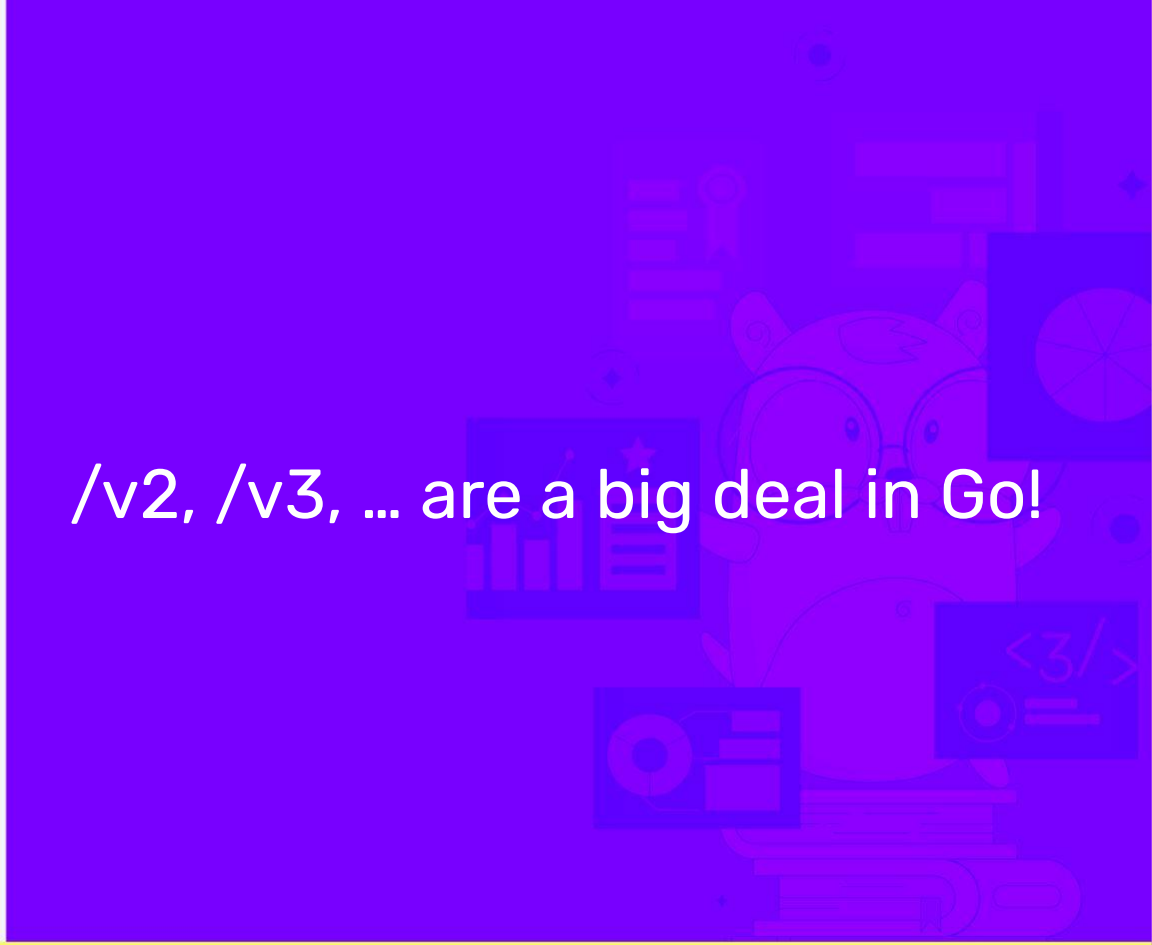
```
//go:build goanalysis

package io

import (
    //go:analyze importers analysis.RFCChecker
    //go:analyze importers analysis.LeanProofChecker
    _ "golang.org/x/tools/go/analysis"
)
```

Module major versions

/v2, /v3, … are a big deal in Go!



Major versions today

```
github.com/foo/bar -> github.com/foo/bar/v2
```

- Extend v1's lifespan instead (deprecations, etc)

Major versions today

github.com/foo/bar -> github.com/foo/bar/v2

- Extend v1's lifespan instead (deprecations, etc)
- Implement v1 in terms of v2? (json/v2 does this!)

```
import jsonv2 "encoding/json/v2"

func Marshal(v any) ([]byte, error) {
    return jsonv2.Marshal(v, &jsonopts.DefaultOptionsV1)
}
```

Major versions today

github.com/foo/bar -> github.com/foo/bar/v2

- Extend v1's lifespan instead (deprecations, etc)
- Implement v1 in terms of v2? (json/v2 does this!)
  - Hard to get rid of tech debt

```
import jsonv2 "encoding/json/v2"

func Marshal(v any) ([]byte, error) {
    return jsonv2.Marshal(v, &jsonopts.DefaultOptionsV1)
}
```

Major versions today

```
// For better compatibility with v1, accept almost any unescaped name.
n := len(tag) - len(strings.TrimLeftFunc(tag, func(r rune) bool {
    // reserve comma, backslash, and quotes
    return !strings.ContainsRune(",\\'\"`", r)
}))
```

Major versions today

```
// For better compatibility with v1, accept almost any unescaped name.
n := len(tag) - len(strings.TrimLeftFunc(tag, func(r rune) bool {
    // reserve comma, backslash, and quotes
    return !strings.ContainsRune(",\\'\"`", r)
}))



if mo.Flags.Get(jsonflags.StringifyWithLegacySemantics) {
    // the `string` tag option does not apply to interface types
    mo.Flags.Clear(jsonflags.StringTag)
}
```

Major versions today

```
import jsonv2 "encoding/json/v2"

// Deprecated: switch to json/v2.
//go:fix inline
func Marshal(v any) ([]byte, error) {
    return jsonv2.Marshal(v, &jsonopts.DefaultOptionsV1)
}


    $ go fix ./...
```

Major versions tomorrow?

PORTING.md in v1 for human and AI readers:

- **What is covered by "go fix"**
- **Which *regretful features* are dropped**
- **Example-based guides for the rest**
- **Notes on subtle behavior differences**

Major versions tomorrow?

PORTING.md in v1 for human and AI readers:

- **What is covered by "go fix"**
- **Which *regretful features* are dropped**
- **Example-based guides for the rest**
- **Notes on subtle behavior differences**

Could be followed downstream by an AI agent:

- **Guided by tests and sandboxed**
- **~90% automated, ~10% needing human decision**
- **Audit report of what was done**

Major v

PORT
- W
- W
- F
- N

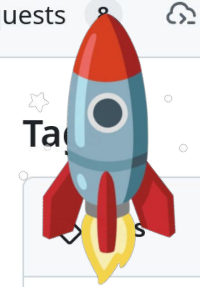
Coul
- C
- A
- A

google / go-github

Code | Issues 21 | Pull requests | Agents | Actions | Security and

Releases

Tags



Ta

v93.0.0

3 weeks ago 1e7e5ad zip tar.gz Notes

v92.0.0

on Sep 14 5149b4d zip tar.gz Notes

v91.0.0

on Sep 3 a13dc0f zip tar.gz Notes

v90.0.0

on Aug 4 32b8631 zip tar.gz Notes

Thank you!

@mvdan.cc

