

go command

```
go run . | go build ./... | go install pkg@v1.2.3
go test ./... | go vet ./... | go fmt ./...
go fix ./... # modernizers (1.26)
go mod init[ tidy ] [download|verify][vendor|why|graph]
go work init[use|sync|vendor]
go get pkg@ver # edit go.mod only
go tool <name> # tool directive (1.24)
go doc pkg.Name | go env -w K=V | go list -m all
go version -m bin | go clean -cache

module example.com/shelf
go 1.27
toolchain go1.27.2
require golang.org/x/sync v0.24.0
replace example.com/x => ../x
tool golang.org/x/tools/cmd/stringer
godebug default=go1.27
ignore ./scratch
```

Packages & imports

One package per directory; package main + func main() builds a binary. Capitalised names are exported. internal/ limits importers to the parent tree.

```
import (
    "fmt"
    str "strings" // alias
    _ "embed"     // side effect only
    . "math"      // dot (avoid)
)
```

Init order: imported packages, then package-level vars (dependency order), then every init() in file-name order. Unused import or variable = compile error.

Types, zero values, literals

type	zero	notes
bool	false	no truthiness
int8uint64, uint8uint64	0	wraps silently
int uint uintptr	0	32/64 bit
byteuint8 rune=int32	0	rune = code point
float32/64 complex64/128	0	IEEE 754
string	""	immutable bytes
pointer func map chan slice interface	nil	
array, struct	all zero	value types
0b101 0o17 0x1F 1_000_000 'x' "s" `raw` 1e3 2i float64(i) int(f) string(r) []byte(s) // explicit only		

Declarations, constants, iota

```
var x, y int = 1, 2      z := "short" // inside funcs
const Pi = 3.14159      // untyped constant
type ID int             // new defined type
type Alias = int         // alias, same type
type LoanState int
const (
    Available LoanState = iota // 0
    Borrowed                 // 1
    Lost                      // 2
)
const ( _ = iota; KB = 1 << (10*iota); MB )
```

Operators: + - * / % & | ^ && << >> && || ! == != < <= > >= <<; &^ clears bits; ++ -- are statements; no ternary.

Control flow

```
if v, err := f(); err != nil { return err } else if v > 0 {
for i := 0; i < n; i++ {      for cond {      for {
for i := range 10 {           // int (1.22)
for i, v := range s {         for k, v := range m {
for i, r := range "hello" {   // runes
for v := range ch {           for x := range seq { // func (1.23)
switch x (case 1, 2: default:} // no fall-through
switch { case a > b: fallthrough } // tagless
switch v := i.(type) { case int: case nil: }
outer: for { for { continue outer; break outer } }
goto done
```

Since 1.22 each iteration has its **own** loop variable.

Functions & defer

```
func div(a, b int) (q int, err error) {
    defer func() {
        if r := recover(); r != nil { err = fmt.Errorf("%v", r) }
    }()
    return a / b, nil // panics if b == 0
}
func sum(xs ...int) int { sum(s...) // variadic
f := func(n int) int { return n * 2 } // closure
p := new(int) *q := new(42) // new(expr) 1.26

defer: LIFO, arguments evaluated at the defer statement, runs on
return and panic. recover only in a deferred func. Results are
passed by value.
```

Slices, maps, strings

```
s := make([]int, len, cap) s = append(s, 1, 2)
s = append(s, t...)
s[lo:hi] s[lo:hi:max] // shares the backing array
copy(dst, src) clear(s) slices.Clone(s) var n []int
a := [3]int{1, 2, 3} b := a // array copy
m := map[string]int{"a": 1} v, ok := m["a"]
delete(m, "a") clear(m) // random iteration order
len(s) cap(s) min(a, b) max(a, b)
```

String = immutable UTF-8 bytes. len(s) bytes; range yields runes; []rune(s); strings.Builder for concatenation. Map keys must be comparable; writing a nil map panics.

Builtins

len cap append copy clear min max new make delete close panic recover print println complex real imag

Structs, embedding, tags

```
type Book struct {
    Title string `json:"title"`
    ISBN   ISBN   `json:"isbn,omitempty"`
    Year   int    `json:"year,omitzero"`
    Author // embedded: fields and methods promoted
}
b := Book{Title: "Go", Year: 2025} p := &b
p.Title // auto-dereference
```

Structs compare with == if all fields are comparable. Order fields large to small to cut padding.

Methods & method sets

```
func (b Book) Label() string { return b.Title }
func (s *Store) Add(b Book) { s.books = append(s.books, b) }
```

value	method set
T	value receivers only
*T	value + pointer receivers

Use a pointer receiver to mutate, for large structs or if any method needs one (be consistent). Addressable x.M() takes &x automatically. Method value f := x.M, expression T.M.

Interfaces

```
type Store interface { Get(ISBN) (Book, error) }
var _ Store = (*MemStore)(nil) // compile-time check
v, ok := i.(Book) switch x := i.(type) {
any = interface{ } fmt.Stringer error io.Reader
```

Satisfaction is implicit. Accept interfaces, return structs. An interface holding a nil pointer is **not** == nil: return nil, never a typed nil, as an error. Common: io.Reader Writer Closer, sort.Interface, http.Handler, json.Marshaler.

Generics & iterators

```
func Map[T, U any](s []T, f func(T) U) []U {
type Number interface{ ~int | ~int64 | ~float64 }
type Set[K comparable] map[K]struct{}
func Max[T cmp.Ordered](a, b T) T {
type List[E any] []E
func (l List[E]) Map[R any](f func(E) R) List[R] {} // 1.27
// iter.Seq[V] = func(yield func(V) bool)
func All(s []Book) iter.Seq2[int, Book] {
    return func(yield func(int, Book) bool) {
        for i, b := range s { if !yield(i, b) { return } }
    }
}
```

slices.Collect(seq) maps.Keys(m) slices.Sorted(maps.Keys(m)) next, stop := iter.Pull(seq); defer stop()

Interface methods cannot have type parameters.

Errors

```
errors.New("x") fmt.Errorf("get %s: %w", id, err)
errors.Is(err, ErrNotFound) errors.As(err, &target)
t, ok := errors.AsType[*PathError](err) // 1.26
errors.Join(e1, e2) errors.Unwrap(err)
var ErrNotFound = errors.New("not found") // sentinel
type NotFound struct{ ID string }
func (e *NotFound) Error() string { return e.ID }

if err != nil { return fmt.Errorf("...: %w", err) }. panic is
for bugs, not control flow; recover at API boundaries.
```

Concurrency

```
go f() // goroutine
ch := make(chan int, 3)
ch <- 1 v, ok := <-ch close(ch)
select {
case v := <-ch:
case ch2 <- x:
case <-time.After(time.Second):
default:
}
var wg sync.WaitGroup; wg.Go(func() {}); wg.Wait() // 1.25
g, ctx := errgroup.WithContext(ctx)
g.SetLimit(8); g.Go(f); g.Wait()
```

op	nil	closed	full/empty
send	blocks	panic	blocks
receive	blocks	zero,false	blocks
close	panic	panic	ok

Only the sender closes. Goroutines are not collected: give each an exit (context, done channel).

context & sync

```
ctx, cancel := context.WithTimeout(parent, 2*time.Second)
defer cancel() // WithCancel WithDeadline WithCancelCause
context.Cause(ctx) context.AfterFunc(ctx, f)
func Fetch(ctx context.Context, id string) error
var mu sync.Mutex; mu.Lock(); defer mu.Unlock()
sync.RWMutex sync.Once sync.OnceValue(f) sync.Pool sync.Map
var n atomic.Int64; n.Add(1); n.Load() atomic.Pointer[T]
```

go test -race finds data races. Never copy a Mutex. Don't store contexts in structs; use ctx.Value only for request-scoped data with unexported key types.

Testing

```
func TestAdd(t *testing.T) {
    for _, tc := range cases {
        t.Run(tc.name, func(t *testing.T) {
            t.Parallel(); ctx := t.Context(); t.Cleanup(f)
            if got != want { t.Errorf("got %w", got) }
        })
    }
}
func BenchmarkX(b *testing.B) { for b.Loop() { work() } }
func FuzzParse(f *testing.F) {
    f.Add("978"); f.Fuzz(func(t *testing.T, s string) {} )
}
func ExampleBook() { fmt.Println("x"); /* Output: x */ }
synctest.Test(t, func(t *testing.T) { synctest.Wait() })
```

go test -run X -v -race -cover -count=1 -bench . -benchmem -fuzz Fuzz -short -shuffle=on. Files *_test.go; package x_test = black box.

Standard library one-liners

```
fmt.Printf("%v %v %v %t %q %x %08.3f %5d %-5s %p %w\n")
strings.Split Join Fields TrimSpace Cut HasPrefix Builder
strconv.Itoa Atoi ParseFloat Quote bufio.NewScanner(os.Stdin)
os.ReadFile os.WriteFile os.Open root, _ := os.OpenRoot(dir)
time.Now() d := 90*time.Second
t.Format("2006-01-02T15:04:05Z07:00")
json.Marshal(v) json.NewDecoder(r).Decode(&v) // v1
// v2: encoding/json/v2 + jsontext (1.27)
slog.Info("loan", "isbn", id, slog.Int("n", 3))
mux.HandleFunc("GET /books/{isbn}", h)
r.PathValue("isbn")
srv := &http.Server{Addr: ":8080", Handler: mux,
    ReadHeaderTimeout: 5 * time.Second}
rows, err := db.QueryContext(ctx, q, args...)
defer rows.Close() // then check rows.Err()
```

Build, directives, cross-compile

```
//go:build linux && amd64
//go:generate stringer -type=LoanState
//go:embed web/* //go:debug panic!l=1
//go:fix inline //go:noninline //go:nosplit
GOOS=linux GOARCH=arm64 CGO_ENABLED=0 go build -trimpath \
    -ldflags "-s -w -X main.version=1.2.3" ./cmd/shelf
GOOS=wasi1 GOARCH=wasm go build GOOS=js GOARCH=wasm
```

Files named x_linux.go, x_arm64.go are constrained by name. Env: GOPATH GOBIN GOPROXY GOSUMDB GOPRIVATE GOFLAGS GOTOOLCHAIN GOEXPERIMENT.

Diagnostics & performance

```
import _ "net/http/pprof" // /debug/pprof/*
go test -cpuprofile cpu.out -bench .
go tool pprof -http=:0 cpu.out go tool trace trace.out
GOC=100 GOMEMLIMIT=2GiB GODEBUG=gctrace=1
go build -pgo=auto # default.pgo in main package dir
go build -gcflags=-m # escape analysis
govulncheck ./... staticcheck ./... golangci-lint run
```

Profiles: cpu, heap, allocs, goroutine, goroutineleak, block, mutex, threadcreate. Measure first, then change.

Reflection, unsafe, cgo, wasm

reflect: reflect.TypeOf(x) ValueOf(x).Kind().Elem().Field(i).Tag.Get("json").Set* (value must be addressable). Laws: interface to reflection object and back; modify only settable. **unsafe**: Sizeof Alignof Offsetof Pointer String StringData Slice SliceData. Not covered by the Go 1 promise. **cgo**: import "C" after a comment preamble; C.CString needs C.free; never keep Go pointers in C memory. **Wasm**: //go:wasmexport name (1.24), wasip1 hosts such as wazero and wasmtime.

Tooling & style

gofmt -l -d -s -w, goimports; go vet analyzers: printf, copylocks, loopclosure, lostcancel, nilness, unusedresult, stdversion. Doc comment starts with the name: // Book is ... Short names, no stutter (catalog.Book), package names lower-case, no underscores. Errors lower-case, no punctuation. Return early; make the zero value useful.

Changed since older tutorials — GOPATH → modules (1.11, default 1.16) · go get installs → go install pkg@ver (1.18) · tools.go → tool directives (1.24) · interface{} → any (1.18) · no generics → generics (1.18), generic type aliases (1.24), generic methods (1.27) · v := v → per-iteration loop variables (1.22) · for i := 0; i < n; i++ → range n (1.22) and range over func (1.23) · io/ioutil → os/io (1.16) · rand.Seed → auto-seeded, math/rand/v2 (1.20/1.22) · // +build → //go:build (1.17) · b.N loop → b.Loop (1.24) · wg.Add(1); go ... → wg.Go (1.25) · runtime.SetFinalizer → runtime.AddCleanup (1.24) · gorilla/mux & co. → ServeMux patterns (1.22) · log → log/slog (1.21) · errors.As → errors.AsType (1.26) · encoding/json v1 → v2 (1.27) · golint → vet, gopls, staticcheck, go fix modernizers (1.26) · time.After / Timer.Reset leaks fixed by timer rework (1.23) · GOMAXPROCS is container-aware (1.25).