

Go Quick Reference

Syntax, types, concurrency, error handling essentials

Basics

Hello World

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

Run & Build

```
go run main.go      # compile and run
go build -o app .   # compile to binary
go test ./...       # run all tests
```

Module Init

```
go mod init github.com/user/project
go mod tidy      # sync dependencies
```

Variables & Types

Declaration

```
var name string = "Go"
age := 15          // short declaration
var x, y int = 1, 2
const Pi = 3.14159
```

Basic Types

bool	true, false
string	UTF-8 immutable byte sequence
int, int8..int64	Signed integers (platform / fixed width)
uint, uint8..uint64	Unsigned integers
float32, float64	IEEE-754 floating point
byte	Alias for uint8
rune	Alias for int32 (Unicode code point)

Zero Values

int, float	0
bool	false
string	"" (empty string)
pointer, slice, map	nil

Functions

Basic Function

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

Multiple Return Values

```
func divide(a, b float64) (float64, error) {
    if b == 0 {
        return 0, errors.New("division by zero")
    }
    return a / b, nil
}
```

Variadic & Anonymous

```
func sum(nums ...int) int {
    total := 0
    for _, n := range nums { total += n }
    return total
}
double := func(x int) int { return x * 2 }
```

Defer

```
func readFile(path string) {
    f, _ := os.Open(path)
    defer f.Close() // runs when function returns
}
```

Control Flow

If / Else

```
if x > 0 {
    fmt.Println("positive")
} else if x == 0 {
    fmt.Println("zero")
} else {
    fmt.Println("negative")
}
```

For Loop

```
for i := 0; i < 10; i++ { } // classic
for x < 100 { x *= 2 }     // while-style
for { break }              // infinite
for i, v := range slice { } // range
```

Switch

```
switch day {
case "Mon", "Tue":
    fmt.Println("early week")
case "Fri":
    fmt.Println("TGIF")
default:
    fmt.Println("other")
}
```

Structs & Methods

Struct Definition

```
type User struct {
    Name string
    Email string
    Age int
}
u := User{Name: "Alice", Email: "a@b.com", Age: 30}
```

Methods

```
func (u User) Greeting() string {
    return "Hi, " + u.Name
}
func (u *User) SetAge(age int) {
    u.Age = age // pointer receiver mutates
}
```

Embedding

```
type Admin struct {
    User // embedded struct
    Level string
}
a := Admin{User: User{Name: "Bob"}, Level: "super"}
fmt.Println(a.Name) // promoted field
```

Interfaces

Defining & Implementing

```
type Stringer interface {
    String() string
}
// implicit implementation - no "implements" keyword
func (u User) String() string {
    return u.Name
}
```

Common Interfaces

io.Reader	Read(p []byte) (n int, err error)
io.Writer	Write(p []byte) (n int, err error)
fmt.Stringer	String() string
error	Error() string

Type Assertion

```
var i interface{} = "hello"
s, ok := i.(string) // ok == true
switch v := i.(type) {
case string: fmt.Println(v)
case int:    fmt.Println(v * 2)
}
```

Goroutines & Channels

Goroutines

```
go func() {
    fmt.Println("running concurrently")
}()
time.Sleep(time.Second)
```

Channels

```
ch := make(chan int) // unbuffered
buf := make(chan int, 5) // buffered
ch <- 42 // send
val := <-ch // receive
```

Select

```
select {
case msg := <-ch1:
    fmt.Println(msg)
case ch2 <- 42:
    fmt.Println("sent")
case <-time.After(time.Second):
    fmt.Println("timeout")
}
```

Patterns

sync.WaitGroup	Wait for multiple goroutines to finish
sync.Mutex	Mutual exclusion lock for shared state
context.Context	Cancellation, deadlines, request-scoped values

Error Handling

Basic Pattern

```
result, err := doSomething()
if err != nil {
    return fmt.Errorf("failed: %w", err)
}
```

Custom Errors

```
type NotFoundError struct {
    ID string
}
func (e *NotFoundError) Error() string {
    return "not found: " + e.ID
}
```

errors Package

errors.New(msg)	Create simple error
fmt.Errorf("%w", err)	Wrap error with context
errors.Is(err, target)	Check error chain for match
errors.As(err, &target)	Extract typed error from chain

Go Quick Reference

Slices & Maps

Slices

```
s := []int{1, 2, 3}
s = append(s, 4, 5)
sub := s[1:3] // [2, 3]
cp := make([]int, len(s))
copy(cp, s)
```

Maps

```
m := map[string]int{"a": 1, "b": 2}
m["c"] = 3
val, ok := m["a"] // ok == true
delete(m, "b")
for k, v := range m { }
```

Slice Operations

len(s)	Number of elements
cap(s)	Underlying array capacity
append(s, elems...)	Append elements, may reallocate
copy(dst, src)	Copy elements between slices
slices.Sort(s)	Sort slice (Go 1.21+ slices pkg)

Packages & Imports

Import Styles

```
import "fmt"
import (
    "os"
    "strings"
    "github.com/user/pkg"
)
```

Visibility

Uppercase first letter = exported (public).
Lowercase first letter = unexported (package-private).
No keywords like public/private needed.

Common Standard Library

fmt	Formatted I/O (Print, Printf, Errorf)
os	OS functions (files, env, args)
io	I/O primitives (Reader, Writer)
net/http	HTTP client and server
encoding/json	JSON encode/decode
strings	String manipulation functions
strconv	String ↔ number conversions
testing	Unit test framework

Generics

Type Parameters

```
func Map[T, U any](s []T, f func(T) U) []U {
    r := make([]U, len(s))
    for i, v := range s { r[i] = f(v) }
    return r
}
```

Constraints

```
type Number interface {
    ~int | ~float64
}
func Sum[T Number](nums []T) T {
    var total T
    for _, n := range nums { total += n }
    return total
}
```

Testing

Basic Test

```
// file: math_test.go
func TestAdd(t *testing.T) {
    got := Add(2, 3)
    if got != 5 {
        t.Errorf("Add(2,3) = %d, want 5", got)
    }
}
```

Test Commands

go test	Run tests in current package
go test ./...	Run all tests recursively
go test -v	Verbose output
go test -run TestAdd	Run specific test by name
go test -bench .	Run benchmarks
go test -cover	Show coverage percentage