Go Cheatsheet v1.21+

Complete Reference for Seasoned Developers

Variables & Types

```go
// Variable declaration
var name string = "gopher"
var name = "gopher"       // Type inferred
name := "gopher"          // Short declaration (only in functi

// Constants
const Pi = 3.14
const (
    StatusOK = 200
    StatusCreated = 201
)

// iota (auto-incrementing)
const (
    Read = 1 << iota  // 1
    Write             // 2
    Execute           // 4
)

// Multiple assignment
i, j := 0, 1
```

Data Structures

```go
// Arrays (fixed size)
var arr [5]int                    // [0,0,0,0,0]
arr := [3]int{1, 2, 3}            // [1,2,3]
arr := [...]int{1, 2, 3, 4, 5}    // Size inferred

// Slices (dynamic size)
slice := []int{1, 2, 3}
slice := make([]int, 3)           // [0,0,0]
slice := make([]int, 3, 5)        // len=3, cap=5
slice = append(slice, 4, 5)       // [0,0,0,4,5]
copy(dest, src)                   // Copy elements

// Slice operations
slice := arr[1:4]                 // Elements 1-3
slice := arr[:3]                  // First 3 elements
slice := arr[2:]                  // From element 2
len(slice)                        // Length
cap(slice)                        // Capacity

// Maps
m := make(map[string]int)
m := map[string]int{"foo": 1, "bar": 2}
m["key"] = value                  // Set value
value, exists := m["key"]         // Check exists
delete(m, "key")                  // Remove key
```

Structs & Pointers

```go
// Struct definition
type Person struct {
    Name string
    Age  int
}

// Creating structs
p := Person{Name: "Bob", Age: 20}
p := Person{"Bob", 20}            // Same order as defined
p := new(Person)                  // Returns a pointer

// Pointers
x := 10
p := &x                           // Point to x
fmt.Println(*p)                   // Dereference
*p = 20                           // Change value

// Struct embedding (composition)
type Employee struct {
    Person                        // Embedded struct
    Salary int
}
emp.Name = "Bob"                  // Access embedded field
```

String Operations

```go
// String basics
s := "Hello, 世界"
len(s)                            // byte length, not char coun
s[0]                              // byte at position (not rune

// Rune handling (Unicode characters)
for i, r := range "Hello, 世界" {
    fmt.Printf("%d: %c\n", i, r)
}

// String manipulation
s := strings.Join([]string{"a", "b"}, ",")
parts := strings.Split("a,b,c", ",")
strings.Contains("seafood", "foo")        // true
strings.HasPrefix("prefix", "pre")        // true
strings.ToUpper("Hello")                  // "HELLO"
strings.TrimSpace(" Hello ")              // "Hello"
strings.Replace("hello", "l", "L", 1)     // "heLlo"
strings.Replace("hello", "l", "L", -1)    // "heLLo"

// String conversions
i, err := strconv.Atoi("42")              // string to int
s := strconv.Itoa(42)                     // int to string
b := []byte("Hello")                      // string to bytes
s := string([]byte{72, 101, 108, 108, 111}) // bytes to string
```

Control Flow

```go
// If statement
if x > 0 {
    // code
} else if x < 0 {
    // code
} else {
    // code
}

// If with short statement
if err := doSomething(); err != nil {
    // handle error
}

// Switch statement
switch os := runtime.GOOS; os {
case "darwin":
    // code
case "linux":
    // code
default:
    // code
}

// Switch with no expression (clean if-else)
switch {
case condition1:
    // code
case condition2:
    // code
}

// For loop
for i := 0; i < 10; i++ {
    // code
}

// For as while
for condition {
    // code
}

// Infinite loop
for {
    // code
    if done { break }
    if skip { continue }
}

// Range - iterate over slice, map, string, channel
for i, v := range slice {
    // i is index, v is value
}
for k, v := range map {
    // k is key, v is value
}
```

Functions

```go
// Basic function
func add(x int, y int) int {
    return x + y
}

// Multiple return values
func divMod(a, b int) (int, int) {
    return a / b, a % b
}

// Named return values
func split(sum int) (x, y int) {
    x = sum * 4 / 9
    y = sum - x
    return                    // Returns x, y
}

// Variadic functions
func sum(nums ...int) int {
    total := 0
    for _, num := range nums {
        total += num
    }
    return total
}
sum(1, 2, 3)
nums := []int{1, 2, 3}
sum(nums...)                  // Spread operator

// Function as value
f := func(x int) int {
    return x * x
}
fmt.Println(f(5))         // 25

// Closures
func adder() func(int) int {
    sum := 0
    return func(x int) int {
        sum += x
        return sum
    }
}
```

Defer, Panic & Recover

```go
// Defer (executes after surrounding function returns)
defer file.Close()        // Common use case
defer mu.Unlock()

// Multiple defers (executed in LIFO order)
defer fmt.Println("1")
defer fmt.Println("2")     // "2" prints before "1"

// Panic (similar to throwing exception)
panic("something went wrong")

// Recover (catch panic)
defer func() {
    if r := recover(); r != nil {
        fmt.Println("Recovered from", r)
    }
}()
```

Methods & Interfaces

```go
// Method definition (on struct)
type Rectangle struct {
    width, height float64
}

// Value receiver (gets copy)
func (r Rectangle) Area() float64 {
    return r.width * r.height
}

// Pointer receiver (can modify receiver)
func (r *Rectangle) Scale(factor float64) {
    r.width *= factor
    r.height *= factor
}

// Interface definition
type Shape interface {
    Area() float64
}

// Implicit implementation (no "implements" keyword)
func CalculateArea(s Shape) float64 {
    return s.Area()
}

// Empty interface (accepts any value)
func PrintAny(v interface{}) {
    fmt.Println(v)
}

// Type assertion
val, ok := v.(string)      // Check if v is string
if !ok {
    // handle not a string
}

// Type switch
switch v := v.(type) {
case int:
    fmt.Println("int:", v)
case string:
    fmt.Println("string:", v)
default:
    fmt.Println("unknown type")
}
```

Concurrency

```go
// Goroutines (lightweight threads)
go func() {
    // code runs concurrently
}()

// Channels (communicate between goroutines)
ch := make(chan int)       // Unbuffered channel
ch := make(chan int, 10)   // Buffered channel

// Send/receive on channel
ch <- 42                   // Send value
val := <-ch                // Receive value
val, ok := <-ch            // Check if closed

// Select statement (multiplex channels)
select {
case v := <-ch1:
    // handle value from ch1
case v := <-ch2:
    // handle value from ch2
case ch3 <- x:
    // sent x on ch3
default:
    // run if no channels ready
}

// Channel directions
func receive(ch <-chan int) {}   // Receive-only
func send(ch chan<- int) {}      // Send-only

// Close channel (no more sends allowed)
close(ch)

// WaitGroup (wait for goroutines)
var wg sync.WaitGroup
wg.Add(1)
go func() {
    defer wg.Done()
    // code
}()
wg.Wait()
```

Error Handling

```go
// Error interface
type error interface {
    Error() string
}

// Returning errors
if err != nil {
    return nil, err
}

// Create errors
errors.New("error message")
fmt.Errorf("error: %v", value)

// Custom error
type MyError struct {
    Code int
    Msg  string
}

func (e *MyError) Error() string {
    return fmt.Sprintf("code %d: %s", e.Code, e.Msg)
}

// Handle errors with if err != nil
if err != nil {
    log.Fatal(err)
}

// Multiple error checks pattern
func doStuff() error {
    err := step1()
    if err != nil {
        return fmt.Errorf("step1 failed: %w", err)
    }

    err = step2()
    if err != nil {
        return fmt.Errorf("step2 failed: %w", err)
    }
    return nil
}
```

Packages & Modules

```go
// Package declaration
package main

// Imports
import (
    "fmt"
    "strings"
)

// Import with alias
import (
    "encoding/json"
    str "strings"
)

// Blank import (init() runs only)
import _ "image/png"

// Exported names (capitalized)
func ExportedFunc() {}     // Accessible outside
func privateFunc() {}      // Package-private

// Package initialization
var DBConn = initDB()
func init() {
    // Run before main()
}

// Create module
// go mod init github.com/user/module

// Add dependency
// go get github.com/pkg/errors
```

File Operations

```go
// Reading files
data, err := os.ReadFile("file.txt")

// Using bufio for line reading
file, err := os.Open("file.txt")
if err != nil {
    return err
}
defer file.Close()

scanner := bufio.NewScanner(file)
for scanner.Scan() {
    line := scanner.Text()
    // process line
}

// Writing files
err := os.WriteFile("file.txt", data, 0644)

// File options (os.O_CREATE|os.O_WRONLY|os.O_APPEND)
file, err := os.OpenFile(
    "file.txt",
    os.O_WRONLY|os.O_CREATE,
    0644,
)
if err != nil {
    return err
}
defer file.Close()
```

Testing

```go
// In file ending with _test.go
package mypackage

import "testing"

// Test function (run with go test)
func TestAdd(t *testing.T) {
    got := Add(2, 3)
    want := 5
    if got != want {
        t.Errorf("Add(2, 3) = %d; want %d", got, want)
    }
}

// Table-driven tests
func TestMultiply(t *testing.T) {
    tests := []struct {
        x, y, want int
    }{
        {2, 3, 6},
        {-1, 5, -5},
        {0, 10, 0},
    }
    for _, tt := range tests {
        got := Multiply(tt.x, tt.y)
        if got != tt.want {
            t.Errorf("Multiply(%d, %d) = %d; want %d",
                     tt.x, tt.y, got, tt.want)
        }
    }
}

// Benchmarks
func BenchmarkFib(b *testing.B) {
    for i := 0; i < b.N; i++ {
        Fibonacci(10)
    }
}
```

Standard Library

```go
// Time operations
now := time.Now()
future := now.Add(time.Hour * 24)
duration := future.Sub(now)
time.Sleep(time.Millisecond * 100)
formatted := now.Format("2006-01-02 15:04:05")

// JSON handling
type Person struct {
    Name string `json:"name"`
    Age  int    `json:"age,omitempty"`
}

// Marshal (Go struct to JSON)
data, err := json.Marshal(person)

// Unmarshal (JSON to Go struct)
err := json.Unmarshal(data, &person)

// HTTP client
resp, err := http.Get("https://example.com")
if err != nil {
    return err
}
defer resp.Body.Close()
body, err := io.ReadAll(resp.Body)

// HTTP server
http.HandleFunc("/", func(w http.ResponseWriter, r *http.Requ
    fmt.Fprintf(w, "Hello, %q", r.URL.Path)
})
http.ListenAndServe(":8080", nil)
```

Quick Links: golang.org • pkg.go.dev • Go Tour • Effective Go