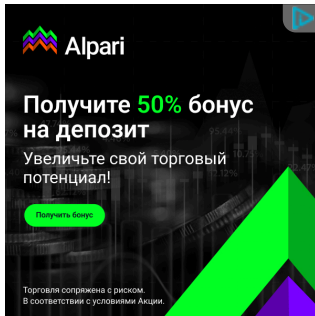




Go cheatsheet

This cheat sheet provided basic syntax and methods to help you using [Go](#).



Акция от Альпари - Получите до \$5000 на первый депозит

Альпари дарит 50% Бонус на депозит. Пополните от \$100 и получите Бонус

SPONSORED BY **ALPARI** / EXINITY

[Learn More](#)

Getting Started

hello.go

```
package main

import "fmt"

func main() {
    fmt.Println("Hello, world!")
}
```

Run directly

```
$ go run hello.go
Hello, world!
```

Or try it out in the [Go repl](#)

Variables

```
var s1 string
s1 = "Learn Go!"

// declare multiple variables at once
var b, c int = 1, 2
var d = true
```

Short declaration

```
s1 := "Learn Go!"           // string
b, c := 1, 2                // int
d := true                   // bool
```

Functions

```
package main

import "fmt"

// The entry point of the programs
func main() {
    fmt.Println("Hello world!")
    say("Hello Go!")
}

func say(message string) {
    fmt.Println("You said: ", message)
}
```

See: [Functions](#)

Comments

```
// Single line comment

/* Multi-
   line comment */
```

If statement

```
if true {
    fmt.Println("Yes!")
}
```

See: [Flow control](#)

Go Basic types

Strings

```
s1 := "Hello" + "World"

s2 := `A "raw" string literal
can include line breaks.`

// Outputs: 10
fmt.Println(len(s1))

// Outputs: Hello
fmt.Println(string(s1[0:5]))
```

Strings are of type `string`.

Numbers

```
num := 3           // int
num := 3.          // float64
num := 3 + 4i      // complex128
num := byte('a')  // byte (alias: uint8)

var u uint = 7      // uint (unsigned)
var p float32 = 22.7 // 32-bit float
```

Operators

```
x := 5
x++
fmt.Println("x + 4 =", x + 4)
fmt.Println("x * 4 =", x * 4)
```

See: [More Operators](#)

Booleans

```
isTrue  := true
isFalse := false
```

Operators

```
fmt.Println(true && true) // true
fmt.Println(true && false) // false
fmt.Println(true || true) // true
fmt.Println(true || false) // true
fmt.Println(!true)        // false
```

See: [More Operators](#)

Arrays

| | | | | | |
|---|---|---|---|----|----|
| | | | | | |
| 2 | 3 | 5 | 7 | 11 | 13 |
| 0 | 1 | 2 | 3 | 4 | 5 |

```
primes := [...]int{2, 3, 5, 7, 11, 13}
fmt.Println(len(primes)) // => 6

// Outputs: [2 3 5 7 11 13]
fmt.Println(primes)

// Same as [:3], Outputs: [2 3 5]
fmt.Println(primes[0:3])

var a [2]string
a[0] = "Hello"
a[1] = "World"

fmt.Println(a[0], a[1]) //=> Hello World
fmt.Println(a)         // => [Hello World]
```

2d array

```
var twoDimension [2][3]int
for i := 0; i < 2; i++ {
    for j := 0; j < 3; j++ {
        twoDimension[i][j] = i + j
    }
}

// => 2d:  [[0 1 2] [1 2 3]]
fmt.Println("2d: ", twoDimension)
```

Pointers

```
func main () {
    b := *getPointer()
    fmt.Println("Value is", b)
}

func getPointer () (myPointer *int) {
    a := 234
    return &a
}

a := new(int)
*a = 234
```

See: [Pointers](#)

Slices

```
s := make([]string, 3)
s[0] = "a"
s[1] = "b"
s = append(s, "d")
s = append(s, "e", "f")

fmt.Println(s)
```

```
fmt.Println(s[1])
fmt.Println(len(s))
fmt.Println(s[1:3])

slice := []int{2, 3, 4}
```

See also: [Slices example](#)

Constants

```
const s string = "constant"
const Phi = 1.618
const n = 5000000000
const d = 3e20 / n
fmt.Println(d)
```

Type conversions

```
i := 90
f := float64(i)
u := uint(i)

// Will be equal to the character Z
s := string(i)
```

How to get int string?

```
i := 90

// need import "strconv"
s := strconv.Itoa(i)
fmt.Println(s) // Outputs: 90
```

Go Strings

Strings function

```
package main

import (
    "fmt"
    s "strings"
)

func main() {
    /* Need to import strings as s */
    fmt.Println(s.Contains("test", "e"))

    /* Build in */
    fmt.Println(len("hello")) // => 5
    // Outputs: 101
```

```
fmt.Println("hello"[1])
// Outputs: e
fmt.Println(string("hello"[1]))
```

```
}
```

fmt.Printf

```
package main
```

```
import (
    "fmt"
    "os"
)
```

```
type point struct {
    x, y int
}
```

```
func main() {
    p := point{1, 2}
    fmt.Printf("%v\n", p)           // => {1 2}
    fmt.Printf("%+v\n", p)         // => {x:1 y:2}
    fmt.Printf("%#v\n", p)         // => main.point{x:1, y:2}
    fmt.Printf("%T\n", p)          // => main.point
    fmt.Printf("%t\n", true)        // => TRUE
    fmt.Printf("%d\n", 123)         // => 123
    fmt.Printf("%b\n", 14)          // => 1110
    fmt.Printf("%c\n", 33)          // => !
    fmt.Printf("%x\n", 456)         // => 1c8
    fmt.Printf("%f\n", 78.9)        // => 78.9
    fmt.Printf("%e\n", 123400000.0) // => 1.23E+08
    fmt.Printf("%E\n", 123400000.0) // => 1.23E+08
    fmt.Printf("%s\n", "\"string\"") // => "string"
    fmt.Printf("%q\n", "\"string\"") // => "\"string\""
    fmt.Printf("%x\n", "hex this") // => 6.86578E+15
    fmt.Printf("%p\n", &p)          // => 0xc00002c040
    fmt.Printf("|%6d|%6d|\n", 12, 345) // => | 12| 345|
    fmt.Printf("|%6.2f|%6.2f|\n", 1.2, 3.45) // => | 1.20| 3.45|
    fmt.Printf("|%-6.2f|%-6.2f|\n", 1.2, 3.45) // => |1.20 |3.45 |
    fmt.Printf("|%6s|%6s|\n", "foo", "b") // => | foo| b|
    fmt.Printf("|%-6s|%-6s|\n", "foo", "b") // => |foo |b |

    s := fmt.Sprintf("a %s", "string")
    fmt.Println(s)

    fmt.Fprintf(os.Stderr, "an %s\n", "error")
}
```

See also: [fmt](#)

Function examples

```
Contains("test", "es")
```

```
true
```

| | |
|-------------------------------|-------------|
| Count("test", "t") | 2 |
| HasPrefix("test", "te") | true |
| HasSuffix("test", "st") | true |
| Index("test", "e") | 1 |
| Join([]string{"a", "b"}, "-") | a-b |
| Repeat("a", 5) | aaaaa |
| Replace("foo", "o", "0", -1) | f00 |
| Replace("foo", "o", "0", 1) | f0o |
| Split("a-b-c-d-e", "-") | [a b c d e] |
| ToLower("TEST") | test |
| ToUpper("test") | TEST |

Go Flow control

Conditional

```
a := 10

if a > 20 {
    fmt.Println(">")
} else if a < 20 {
    fmt.Println("<")
} else {
    fmt.Println("=")
}
```

Statements in if

```
x := "hello go!"

if count := len(x); count > 0 {
    fmt.Println("Yes")
}

if _, err := doThing(); err != nil {
    fmt.Println("Uh oh")
}
```

Switch

```

x := 42.0
switch x {
case 0:
case 1, 2:
    fmt.Println("Multiple matches")
case 42: // Don't "fall through".
    fmt.Println("reached")
case 43:
    fmt.Println("Unreached")
default:
    fmt.Println("Optional")
}

```

See: [Switch](#)

For loop

```

for i := 0; i <= 10; i++ {
    fmt.Println("i: ", i)
}

```

For-Range loop

```

nums := []int{2, 3, 4}
sum := 0
for _, num := range nums {
    sum += num
}
fmt.Println("sum:", sum)

```

While loop

```

i := 1
for i <= 3 {
    fmt.Println(i)
    i++
}

```

Continue keyword

```

for i := 0; i <= 5; i++ {
    if i % 2 == 0 {
        continue
    }
    fmt.Println(i)
}

```

Break keyword

```

for {
    fmt.Println("loop")
}

```


Go Structs & Maps

Defining

```
package main

import (
    "fmt"
)

type Vertex struct {
    X int
    Y int
}

func main() {
    v := Vertex{1, 2}
    v.X = 4
    fmt.Println(v.X, v.Y) // => 4 2
}
```

See: [Structs](#)

Literals

```
v := Vertex{X: 1, Y: 2}
// Field names can be omitted
v := Vertex{1, 2}
// Y is implicit
v := Vertex{X: 1}
```

You can also put field names.

Maps

```
m := make(map[string]int)
m["k1"] = 7
m["k2"] = 13
fmt.Println(m) // => map[k1:7 k2:13]

v1 := m["k1"]
fmt.Println(v1) // => 7
fmt.Println(len(m)) // => 2

delete(m, "k2")
fmt.Println(m) // => map[k1:7]

_, prs := m["k2"]
fmt.Println(prs) // => false
```

```
n := map[string]int{"foo": 1, "bar": 2}
fmt.Println(n) // => map[bar:2 foo:1]
```

Pointers to structs

```
v := &Vertex{1, 2}
v.X = 2
```

Doing `v.X` is the same as doing `(*v).X`, when `v` is a pointer.

Go Functions

Multiple arguments

```
func plus(a int, b int) int {
    return a + b
}
func plusPlus(a, b, c int) int {
    return a + b + c
}
fmt.Println(plus(1, 2))
fmt.Println(plusPlus(1, 2, 3))
```

Multiple return

```
func vals() (int, int) {
    return 3, 7
}

a, b := vals()
fmt.Println(a) // => 3
fmt.Println(b) // => 7
```

Function literals

```
r1, r2 := func() (string, string) {
    x := []string{"hello", "quickref.me"}
    return x[0], x[1]
}()

// => hello quickref.me
fmt.Println(r1, r2)
```

Naked returns

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

```
x, y := split(17)
fmt.Println(x)    // => 7
fmt.Println(y)    // => 10
```

Note that using naked returns hurts readability.

Variadic functions

```
func sum(nums ...int) {
    fmt.Print(nums, " ")
    total := 0
    for _, num := range nums {
        total += num
    }
    fmt.Println(total)
}

sum(1, 2)    //=> [1 2] 3
sum(1, 2, 3) // => [1 2 3] 6

nums := []int{1, 2, 3, 4}
sum(nums...) // => [1 2 3 4] 10
```

init function

```
import --> const --> var --> init()
```

```
var num = setNumber()

func setNumber() int {
    return 42
}

func init() {
    num = 0
}

func main() {
    fmt.Println(num) // => 0
}
```

Functions as values

```
func main() {
    // assign a function to a name
    add := func(a, b int) int {
        return a + b
    }
    // use the name to call the function
    fmt.Println(add(3, 4)) // => 7
}
```

Closures 1

```
func scope() func() int{
    outer_var := 2
    foo := func() int {return outer_var}
}
```

```
    return foo
}

// Output: 2
fmt.Println(scope())
```

Closures 2

```
func outer() (func() int, int) {
    outer_var := 2
    inner := func() int {
        outer_var += 99
        return outer_var
    }
    inner()
    return inner, outer_var
}

inner, val := outer()
fmt.Println(inner()) // => 200
fmt.Println(val)     // => 101
```

Go Packages

Importing

```
import "fmt"
import "math/rand"
```

Same as

```
import (
    "fmt" // gives fmt.Println
    "math/rand" // gives rand.Intn
)
```

See: [Importing](#)

Aliases

```
import r "math/rand"
```

```
import (
    "fmt"
    r "math/rand"
)
```

```
r.Intn()
```

Packages

```
package main
```

```
// An internal package may be imported only by another package  
// that is inside the tree rooted at the parent of the internal directory  
package internal
```

See: [Internal packages](#)

Exporting names

```
// Begin with a capital letter  
func Hello () {  
    ...  
}
```

See: [Exported names](#)

Go Concurrency

Goroutines

```
package main
```

```
import (  
    "fmt"  
    "time"  
)
```

```
func f(from string) {  
    for i := 0; i < 3; i++ {  
        fmt.Println(from, ":", i)  
    }  
}
```

```
func main() {  
    f("direct")  
    go f("goroutine")  
  
    go func(msg string) {  
        fmt.Println(msg)  
    }("going")  
  
    time.Sleep(time.Second)  
    fmt.Println("done")  
}
```

See: [Goroutines](#), [Channels](#)

WaitGroup

```

package main

import (
    "fmt"
    "sync"
    "time"
)

func w(id int, wg *sync.WaitGroup) {
    defer wg.Done()
    fmt.Printf("%d starting\n", id)

    time.Sleep(time.Second)
    fmt.Printf("%d done\n", id)
}

func main() {
    var wg sync.WaitGroup
    for i := 1; i <= 5; i++ {
        wg.Add(1)
        go w(i, &wg)
    }
    wg.Wait()
}

```

Closing channels

```

ch <- 1
ch <- 2
ch <- 3
close(ch) // Closes a channel

// Iterate the channel until closed
for i := range ch {
    ...
}

// Closed if `ok == false`
v, ok := <- ch

```

See: [Range and close](#)

Buffered channels

```

ch := make(chan int, 2)
ch <- 1
ch <- 2
ch <- 3
// fatal error:
// all goroutines are asleep - deadlock

```

See: [Buffered channels](#)

Go Error control

Deferring functions

```
func main() {
    defer func() {
        fmt.Println("Done")
    }()
    fmt.Println("Working...")
}
```

Lambda defer

```
func main() {
    var d = int64(0)
    defer func(d *int64) {
        fmt.Printf("& %v Unix Sec\n", *d)
    }(&d)
    fmt.Print("Done ")
    d = time.Now().Unix()
}
```

The defer func uses current value of d, unless we use a pointer to get final value at end of main.

Defer

```
func main() {
    defer fmt.Println("Done")
    fmt.Println("Working...")
}
```

See: [Defer, panic and recover](#)

Go Methods

Receivers

```
type Vertex struct {
    X, Y float64
}

func (v Vertex) Abs() float64 {
    return math.Sqrt(v.X * v.X + v.Y * v.Y)
}
```

```
v := Vertex{1, 2}
v.Abs()
```

Mutation

```
func (v *Vertex) Scale(f float64) {
    v.X = v.X * f
    v.Y = v.Y * f
}
```

```
v := Vertex{6, 12}
v.Scale(0.5)
// `v` is updated
```

See: [Pointer receivers](#)

Go Interfaces

A basic interface

```
type Shape interface {
    Area() float64
    Perimeter() float64
}
```

Struct

```
type Rectangle struct {
    Length, Width float64
}
```

Struct `Rectangle` implicitly implements interface `Shape` by implementing all of its methods.

Methods

```
func (r Rectangle) Area() float64 {
    return r.Length * r.Width
}

func (r Rectangle) Perimeter() float64 {
    return 2 * (r.Length + r.Width)
}
```

The methods defined in `Shape` are implemented in `Rectangle`.

Interface example

```
func main() {
    var r Shape = Rectangle{Length: 3, Width: 4}
```



```
fmt.Printf("Type of r: %T, Area: %v, Perimeter: %v.", r, r.Area(), r.Perimeter())
}
```

Miscellaneous

Keywords

| | | |
|-----------|-------------|----------|
| 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 | | |

Operators and punctuation

| | | | | | | | | |
|---|----|-----|-----|----|----|-----|---|---|
| + | & | += | &= | && | == | != | (|) |
| - | | -= | = | | < | <= | [|] |
| * | ^ | *= | ^= | <- | > | >= | { | } |
| / | << | /= | <<= | ++ | = | := | , | ; |
| % | >> | %= | >>= | -- | ! | ... | . | : |
| | &^ | &^= | | | | | | |

Also see

- [Devhints](#) (devhints.io)
- [A tour of Go](#) (tour.go.dev)
- [Go wiki](#) (github.com)
- [Effective Go](#) (go.dev)
- [Go by Example](#) (gobyexample.com)
- [Awesome Go](#) (awesome-go.com)
- [JustForFunc Youtube](#) (youtube.com)
- [Style Guide](#) (github.com)



Plantillas de CV
Gratis



Descarga
Word o PDF



Crea tu CV en
Sólo 3 Pasos



Generador de CV
con IA



+ Carta de
Presentación

OnlineCV

Top Cheatsheet

Python Cheat
Quick Referen

Vim Cheatshe
Quick Referen

Remote Work
Quick Referen

Homebrew Cl
Quick Referen

JavaScript Che
Quick Referen

Bash Cheatshe
Quick Referen

PyTorch Chea
Quick Referen

Taskset Cheat
Quick Referen

Recent Cheatsheet



QuickRef.ME

Share quick reference and cheat sheet for developers.

中文版 #Notes

