

# Go Basics

WASA: Web and Software Architecture

---

Prof. Emanuele Panizzi

# GO Language

Build fast, reliable, and efficient software at scale

- ✓ Go is an open source programming language supported by Google
- ✓ Easy to learn and get started with
- ✓ Built-in concurrency and a robust standard library
- ✓ Growing ecosystem of partners, communities, and tools

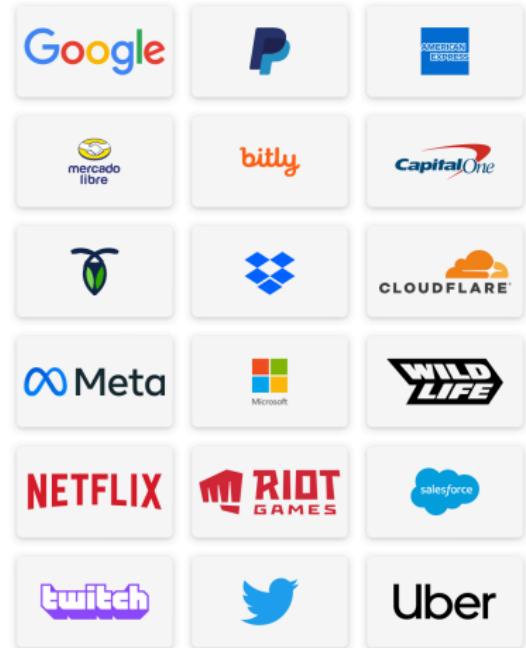


Figure 1: Companies using Go

# Go Playground

The screenshot shows the Go Playground interface. At the top, there's a navigation bar with links for "Why Go", "Get Started", "Docs", "Packages", "Play", and "Blog". Below the navigation bar is a toolbar with buttons for "Run", "Format", "Share", and a dropdown menu set to "Hello, World!". The main area is titled "The Go Playground" and contains a code editor. The code in the editor is:

```
1 // You can edit this code!
2 // Click here and start typing.
3 package main
4
5 import "fmt"
6
7 func main() {
8     fmt.Println("Hello, 世界")
9 }
```

The code is numbered from 1 to 35 on the left side. To the right of the code editor, there's a yellow rectangular area where the output of the code will be displayed.

**Figure 2:** <https://go.dev/play/>

# Packages

- Go functions are grouped into packages.
- Each package is made of one or more files in the same directory
- The package is declared at the beginning of the file, e.g.: package main below

```
package main  
//...
```

# Import

- A file can import other packages, e.g.:

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

# Hello World

- Let's import the `fmt` package, which contains functions for formatting and printing text

```
package main

import "fmt"

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

- You can use the functions of the imported package if their name begins with a capital letter
- same for variables (e.g., `math.Pi`)

## Go basic types

bool

string

int int8 int16 int32 int64

uint uint8 uint16 uint32 uint64 uintptr

byte // alias for uint8

rune // alias for int32, represents a Unicode code point

float32 float64

complex64 complex128

# Variables

```
package main
import "fmt"

// these are visible in the package
var b bool
var s string
var x, y float32

func main() {
    var i int // this is local in this function
    fmt.Println(i, b, s, x, y)
}
```

- Note that type is after the variable name(s)

## Initial values

```
var i int = 10
var b = true // takes the type of the initializer
var x, y = 1.5, 2.5
```

Zero values are used when there is no explicit initial value:

- 0 for numeric types,
- false for the boolean type, and
- "" (the empty string) for strings.

## Short declaration with initialization inside functions

```
package main
import "fmt"

func main() {
    k := 3
    fmt.Println(k)
}
```

# Constants

```
// ...
const k = 10
const zero = -273.15
// ...
```

# Arrays

```
package main
import "fmt"

func main() {
    var v [10]int
    odds := [5]int{1, 3, 5, 7, 9}
    i := 4
    v[0] = odds[i] // 9
    fmt.Println(v)
}

// [9 0 0 0 0 0 0 0 0 0]
```

## Slices

```
package main
import "fmt"

func main() {
    vowels := [5]string{"a", "e", "i", "o", "u"}
    var s []string = vowels[1:3] // [e, i] (half-open range)
    fmt.Println(s)
}
```

- default is zero for the slice's low bound and the length of the array for the high bound
- `vowels[0:5]`, `vowels[:5]`, `vowels[0:]`, and `vowels[:] are the same`

## Slices (contd)

- A slice does not store any data, it just describes a section of an underlying array.

```
package main
import "fmt"

func main() {
    vowels := [5]string{"a", "e", "i", "o", "u"}
    var s []string = vowels[1:3]
    t := vowels[1:3]
    fmt.Println(t) // [e i]
    s[0] = "x"
    fmt.Println(vowels) // [a x i o u]
    fmt.Println(t) // [x i]
}
```

# Literals

- Array Literal

```
[3]bool{true, true, false}
```

- Slice Literal

```
[]bool{true, true, false}
```

it creates an array and then a slice that references it

## Slices (contd)

- length: number of elements in the slice
- capacity: number of elements from low bound to end of array
- e.g.: var s []string = vowels[1:3]: len(s) = 3, cap(s) = 4
- builtin make() function:

```
package main
import "fmt"

func main() {
    b := make([]int, 0, 5)
    fmt.Println(len(b)) // 0
    fmt.Println(cap(b)) // 5
    fmt.Println(b) // [] or nil
}
```

# Maps

```
package main
import "fmt"

var temperatures map[string]int

func main() {
    temperatures = make(map[string]int)
    // must use make() to initialize a map
    temperatures["Rome"] = 27
    temperatures["London"] = 18
    temperatures["San Francisco"] = 21
    fmt.Println(temperatures)
}
```

## Maps (contd)

```
package main
import "fmt"

func main() {
    // with map literals
    var temp = map[string]int{
        "r": 27,
        "l": 18,
        "s": 21,
    }
    fmt.Println(temp) // map[l:18 r:27 s:21]
}
```

## Maps (contd)

- `delete(m, key)` to delete an element
- e.g. `delete(temperatures,"Rome")`
- `elem, ok := m[key]` to test that a key is present
- e.g.

```
//...
elem, ok := temperatures["London"] // 18 true
elem, ok = temperatures["Paris"]   // 0 false
//...
```

- (this is two-value assignment)

# Pointers

- The type `*T` is a pointer to a `T` value. Its zero value is `nil`.
- e.g., `var p *int`
- The `&` operator generates a pointer to its operand, e.g.:

```
i := 42  
p = &i
```

- Dereferencing

```
fmt.Println(*p) // read i through the pointer p  
*p = 21        // set i through the pointer p
```