# 

GoLang

[**1. Key words 2**](#_oft20zmhw6tr)

[**2. Operations 2**](#_qgtonq4feb9q)

[**3. Structs 2**](#_niqp4ib5u7eu)

[**4. Data Types 3**](#_897oycx49nab)

[**5. Functions 3**](#_3ynwnkhgqkht)

[**6. Interface 4**](#_siysllyjowy6)

# 

# Key words

**for** i := 1 **;** i < N **;** i++ **{**

Statement

**}**

**for** var1 < var2 **{**

var1-- // or var2++

**}**

**for** \_ **,** v := **range** **{**  // range can be array, slice or map

Statement

**}**

**if** cond **{**

Statement

**}**

**switch {**

**case** var == smt **:**

Statement

**}**

**switch** var **{**

**case** smt **:**

Statement

**}**

# Operations

var = append(var, Val1, Val2)

var = append(var, slice...)

defer fmt.Println("fds") //executa esse comando por último

# Structs

* 1. General declaration

type NAME **struct** **{**

atr1 type

atr2 type

**}**

* 1. Assignment it to a variable

name1 := name **{**

atr1: value,

atr2: value,

**}**

# Data Types

* 1. array := [SIZEnumb] type { val1, val2, values }
  2. slice := [ ] type{ val1, val2, values }
  3. make0 := make( [ ] type, len, cap) //works for slice and array
  4. make1 := make( map[type]T ) //works for map
  5. matrix := [ ][ ] type {

[ ]type {

},

[ ]type {

},

}

* 1. mepe := map [type]T{ //T will be the values types (int, slice, struct)

"key1": values,

}

# Functions

* 1. Default

**func (**receiver**)** identifier **(**parameters**) (**returns**) {** code **}**

* 1. Function without parameters
     1. it will return an element of type type

**func** NAME**()** type**{**code**}**

* + 1. it will return two elements of type type1 and type2

**func** NAME**() (**type1**,** type2**) {** code **}**

* 1. Function with parameters
     1. variadic parameters of type type and will return nothing

**func** NAME**(**var ...type**) {** code **}**

* + 1. 2 normal parameters

**func** NAME**(**var type**,** var2 type2**)** type **{** code **}**

* 1. Anonymous function
     1. works inside main (runtime)

**func(**par**) {** code **} ()**

* + 1. assignment to a variable

var := **func(**par**) {** code **};** var()

* 1. Function that returns another function
     1. the return must contain the function

**func** NAME**() func() {** code **}**

* 1. Function that receives as parameter another function

**func** NAME**(**f func()**) {** code **}**

# Interface

* 1. Default

type NAME **interface** **{** code **}**

* 1. Every struct that has the area method belongs to the figure interface

**type** figure **interface {** area() **}**

1. .