Go Cheat Sheet

by char1010101010 via cheatography.com/183653/cs/38233/

WaitGroups

```
func generateRandomNumber(ch chan<- int, id int, wg
*sync.WaitGroup) {
        defer wg.Done()
        ran d.S eed (ti me.N ow ().U ni xNano() +
int64(id))
        ran dom Number := rand.I ntn (100)
       ch <- random Number
}
func main() {
       var wg sync.W ait Group
       ch := make(chan int, 100)
        run tim e.G OMA XPR OCS(3)
       for i := 1; i <= 100; i++ {
                wg.A dd(1)
               go genera teR and omN umb er(ch, i,
&wg)
       }
       go func() {
                wg.W ait()
                clo se(ch)
       }()
       for random Number := range ch {
                fmt.Pr int ln( ran dom Number)
       }
}
```

Arrays

```
func mean(tab [5]int) (meanVal float64) {
   // for index, value := range collection
   for _, value := range tab {
        mea nVal+= (float 64) (value)
   }
    meanVal /= (float 64) (le n(tab))
    return
}
func main() {
   var table = [5]int{3, 4, 8, 9, 2}
   m := mean(t able) // pass by value
    fmt.Pr int f("r esult= %f\n", m)
```

Arrays (cont)

```
>}
```

Maps

```
func main() {
       m := make(m ap[ str ing ]int)
        m["k 1"] = 7
        m["k 2"] = 13
        fmt.Pr int ln( " map :", m) //map:
map[k1:7 k2:13]
       v1 := m["k 1"]
        fmt.Pr int ln( " v1: ", v1) //v1: 7
       v3 := m["k 3"]
        fmt.Pr int ln( " v3: ", v3) //v3: 0
        fmt.Pr int ln( " len :", len(m)) //len: 2
        del ete(m, " k2")
        fmt.Pr int ln( " map :", m) //map:
map[k1:7]
       _, prs := m["k 2"] //second return val to
see if key is present
        fmt.Pr int ln( " prs :", prs) //prs: false
       n := map[st rin g]i nt{ " foo ": 1, " -
bar ": 2}
        fmt.Pr int ln( " map :", n) //map:
map[bar:2 foo:1]
}
```

Method

```
func (pt *Point) norm() float64 {
    return math.S qrt (pt.x * pt.ypt.y)
}
```

Range

```
for i, num := range nums {
      fmt.Pr int ln( " index: ", i, " num: ", num)
}
```

Range

```
for i, num := range nums {
      fmt.Pr int ln( " index: ", i, " num: ", num)
}
```

Iota

```
const (
   ON = iota
   OFF = iota // optional here
    OTHER = iota
)
fmt.Pr int ln(ON, OFF, OTHER) // Will print: 0 1 2
```

Functions as Parameters

```
func main() {
   a := Point{ 2.,4.}
   b := Point{ 5.,9.}
   dist := calc(a ,b, Dis tance)
    fmt.Pr int f("r esult= %f\n", dist)
   dist = calc(a,b,
        func(p Point, q Point) flo at64{
            return math.A bs( p.x -q.x )+ mat h.A -
bs( p.y -q.y)
   })
    fmt.Pr int f("r esult= %f\n", dist)
}
```

Pointers and Structs

```
type Point struct {
   x int
   y int
}
func main() {
   pt := Point{8, 1}
    com ple men t(&pt)
    fmt.Pr int f("R esult= %d and %d\n", pt.x ,
pt.y)
}
func comple ment(p *Point) {
   p.x, p.y = -p.y, -p.x
}
```

Slices

```
func main() {
       s := make([ ]st ring, 3)
        fmt.Pr int ln( " emp :", s) //emp: [ ]
       s[0] = " a"
       s[1] = " b"
       s[2] = " c"
        fmt.Pr int ln( " set :", s) //set: [a b c]
        fmt.Pr int ln( " get :", s[2]) //get: c
        fmt.Pr int ln( " len :", len(s)) //len: 3
       s = append(s, " d")
       s = append(s, " e", " f")
        fmt.Pr int ln( " apd :", s) //apd: [a b c
d e f]
       c := make([ ]st ring, len(s))
        copy(c, s)
        fmt.Pr int ln( " cpy :", c) //cpy: [a b c
d e f]
       l := s[2:5]
        fmt.Pr int ln( " sl1 :", l) //sl1: [c d e]
       l = s[:5]
        fmt.Pr int ln( " sl2 :", l) //sl2: [a b c
d e]
       l = s[2:]
        fmt.Pr int ln( " sl3 :", l) //sl3: [c d e
f]
       t := []stri ng{ " g", " h", " i"}
        fmt.Pr int ln( " dcl :", t) //dcl: [g h i]
       twoD := make([ ][]int, 3)
       for i := 0; i < 3; i++ {
                inn erLen := i + 1
                twoD[i] = make([ ]int, innerLen)
               for j := 0; j < innerLen; j++ {
                        two D[i][j] = i + j
               }
       }
        fmt.Pr int ln( "2d: ", twoD) //2d: [[0] [1
2] [2 3 4]]
}
```