
GOLANG UNITED
Data Structures

GOLANG BASIC TYPES

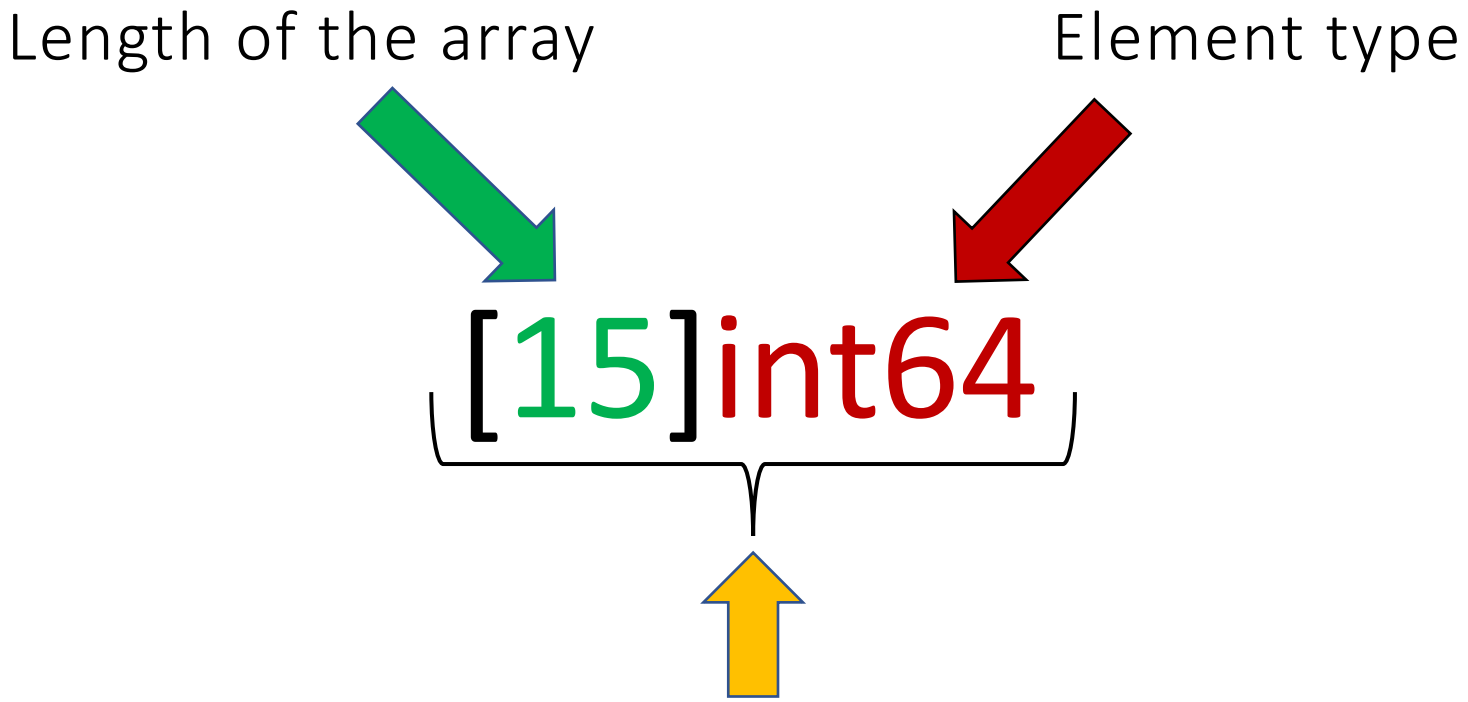
Basic data structures in Go

- Array – fixed collection of elements. – immutable list
- Slice – collection with dynamic size – mutable list
- Map - a collection of unordered pairs of key-value – dict
- Struct - typed collections of fields – data class,
  можно добавить методы, но это не ООП

length
element type
[15] int64

ARRAYS

Arrays in Go



Arrays in Go

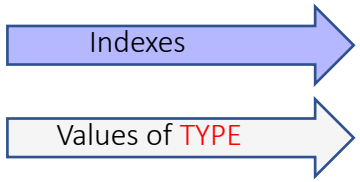
How to declare an array?

- Using VAR keyword

```
var arr [length]Type

var arr = [length]Type {item1, item2, … itemN}
```

- Using shorthand declaration

```
arr := [length]Type {item1, item2, … itemN}
```



0	1	2	3	4	..	..	N-1
$item_1$	$item_2$	$item_3$	$item_4$	$item_5$	..	..	$item_{N-1}$

Arrays in Go

```
func main() {
	var arr1 [3]int
	fmt.Println(arr1)
	var arr2 [4]int
	fmt.Println(arr2)
	fmt.Println(arr1 == arr2)
}
```

./prog.go:14:23: invalid operation: arr1 == arr2 (mismatched types [3]int and [4]int)

https://play.golang.org/p/Wl70u-JEFRa

Array type computed from it's size and type of the element, due to this array of 3 integers is not the same as arrays of 4 integers. You cannot compare variables with different types, it's true for array as well.

- Array – строго-типизированный объект.

Что это значит? Нельзя сравнивать массивы разной длины, или если элементы разных типов.

[15]int64 != [16]int64

Arrays in Go

Automatic array length declaration

`var arr1 [...]Type { item1, item2, …, itemN }`

len(arr1**)** - the length of the array (built-in function)

EXAMPLE

```
func main() {

    a := [...]string{"one", "two
", "three"}

    fmt.Println(a)
    fmt.Println(len(a))
    fmt.Printf("%T", a)
}
```

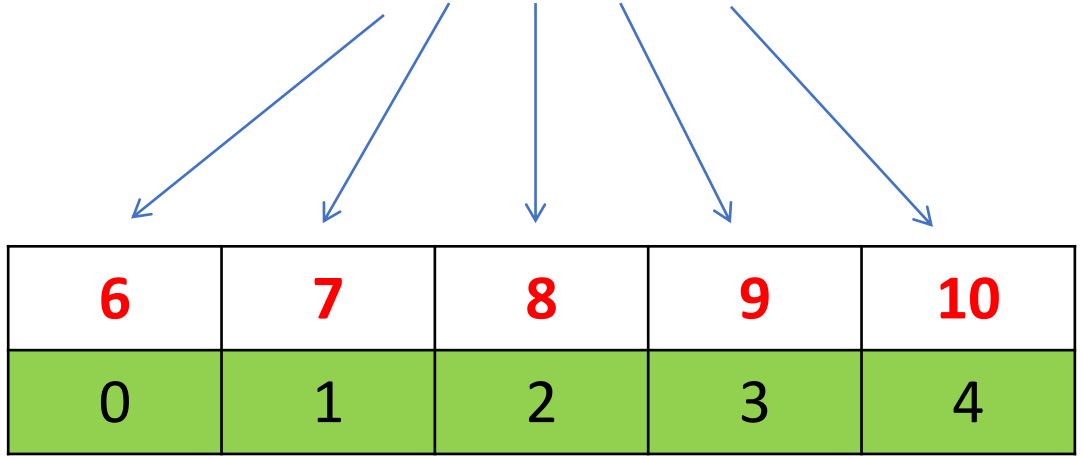
OUTPUT

```
[one two three]
3
[3]string
```

Arrays in Go

Array literals

```
array_name := [5]int{6, 7, 8, 9, 10}
```




```
arr := [5]int{0: 6, 1: 7, 2: 8, 3: 9, 4: 10}
arr := [5]int{0: 6, 7, 8, 9, 10}
```

Arrays in Go - Array literals

```
arr1 := [5]int{2: 6, 7, 8}
```

0	0	6	7	8
0	1	2	3	4

EXAMPLE

```
func main() {
    a := [5]int{2: 6
, 7, 8}
    c := [5]int{2: 6
, 0: 7, 8}

    fmt.Println(a)
    fmt.Println(c)
}
```

OUTPUT

```
[0 0 6 7 8]
[7 8 6 0 0]
```

Arrays in Go - Array literals

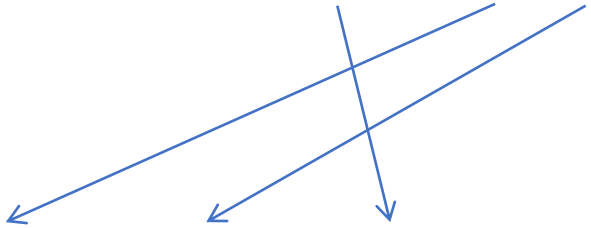
```
arr := [5]int{2: 6, 7, 8, 9}
```

0	0	6	7	8
0	1	2	3	4


```
./prog.go:9:26: array index 5 out of bounds [0:5]
```

Arrays in Go - Array literals

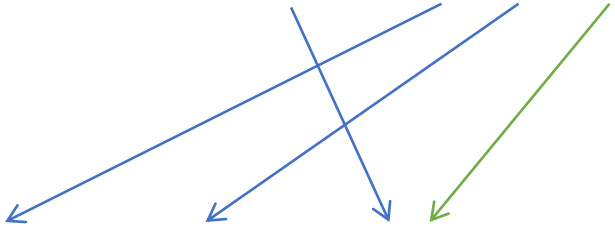
```
arr := [5]int{2: 6, 0: 7, 8}
```



7	8	6	0	0
0	1	2	3	4

Arrays in Go - Array literals

```
arr := [5]int{2: 6, 0: 7, 8, 9}
```



7	8	6	0	0
0	1	2	3	4

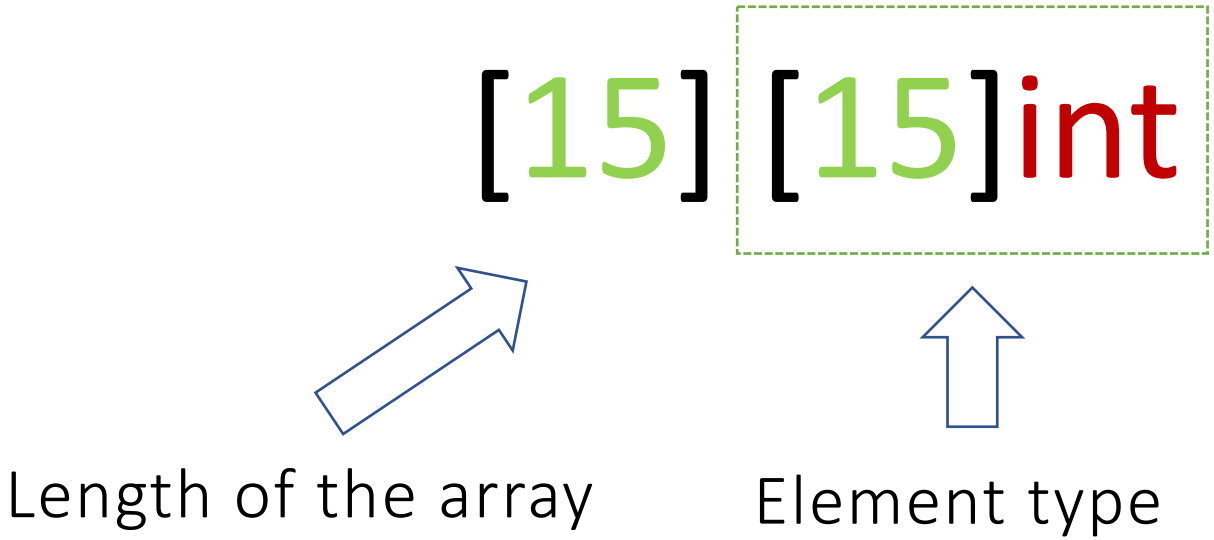

```
./prog.go:11:29: duplicate index in array literal: 2
```

Arrays in Go - Multi-dimensional arrays

[15]<type>

Type could be an array

Arrays in Go - Multi-dimensional arrays



Arrays in Go - Multi-dimensional arrays

EXAMPLE

```
func main() {
    a := [2][2]int{
        [2]int{1, 2},
        [2]int{3, 4},
    }
    b := [2][2]int{
        {1, 2},
        {3, 4},
    }
    fmt.Println(a)
    fmt.Println(b)
}
```

OUTPUT

```
[[1 2] [3 4]]
[[1 2] [3 4]]
```

Arrays in Go - How to pass an array?

- Create a copy

```
arr2 := arr1
```

arr1	0	1	..	N-1
	$item_1$	$item_2$	..	$item_{N-1}$
	0x400002	0x400006		0xXXXXXX

arr2	0	1	..	N-1
	$item_1$	$item_2$	..	$item_{N-1}$
	0x500004	0x500008		0xXXXXXXX

- Pass a pointer

```
arr2 := &arr1
```

arr2	0x500004

arr1	0	1	..	N-1
	$item_1$	$item_2$	..	$item_{N-1}$
	0x500004	0x500008		0xXXXXXXX

Due to their fixed length Array are not as much popular as Slice in Go.

SLICES

Slices in Go

Length is not set

Element type

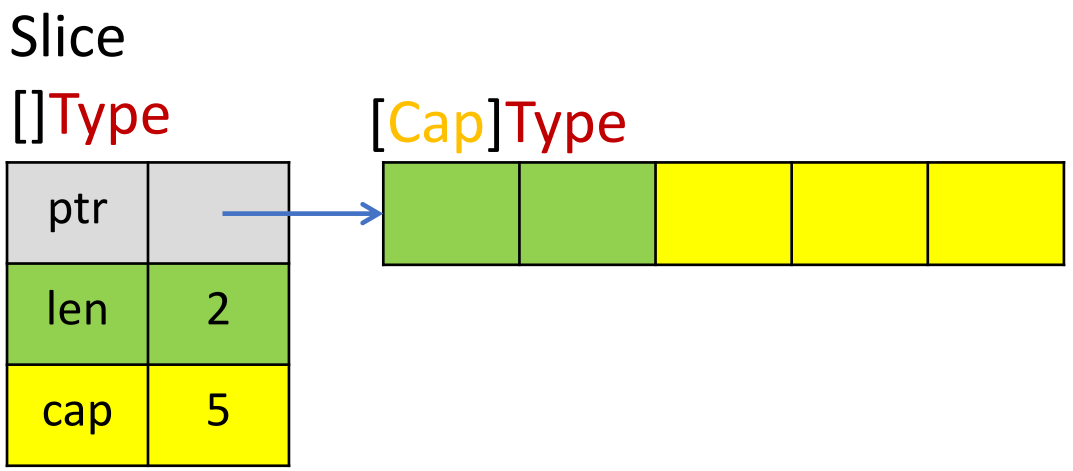
[]int64

Slices in Go - How to declare a slice?

- Built-in function make()

```
slice := make([]int, 2, 5)
```

type length capacity



- Using VAR keyword

```
var slice []Type

var slice := []Type{item1, … itemN}
```

- Using shorthand declaration

```
slice := []Type {item1, … itemN}
```

Slices in Go - Internal structure

```
type _slice struct {
    elements   unsafe.Pointer    // pointer to the sequence of data
    len        int               // total number of elements currently contained
    cap        int               // total number of memory locations provisioned.
}
```

Benefits of slice

- Slices of different lengths *can* be assigned each other's values
- Slices can share the same backing array
- Slices can stay in the stack, only backing array should be in the heap
- Slice's type stay the same, but the pointer, length, and capacity changing

Slices in Go - Built-in functions

- make() – initialize slice structure and initialize elements depending on configuration.
- copy() – copy slice
- cap() – returns capacity of the array (length of the underlying array)
- len() – returns count of elements in slice currently
- append() – put an element into the end of the slice
- [:] – slicing operation, allows you to take a part of the existing array or slice

Slices in Go - Built-in functions

EXAMPLE

```go
func main() {
    var a []int
    var b = []int{1, 2, 3}
    c := []bool{true, false}
    d := []string{
        "one",
        "two",
    }
    e := make([]int, 3)
    f := new([]int)
    fmt.Println(a, b, c, d, e, f)
    fmt.Printf("%T\n", e)
    fmt.Printf("%T\n", f)
}
```

OUTPUT

```
[]
[1 2 3]
[true false]
[one two]
[0 0 0]
&[]
[]int
*[]int
```

Slices in Go - Mutations

```
slice1 := []int{1,2,3}
```

slice1 = []int

ptr	
len	3
cap	3

[3]int
Backing array

1	2	3

```
slice1 = append(slice1, 4)
```

slice1 = []int

ptr	
len	4
cap	6

[3]int
Old backing array

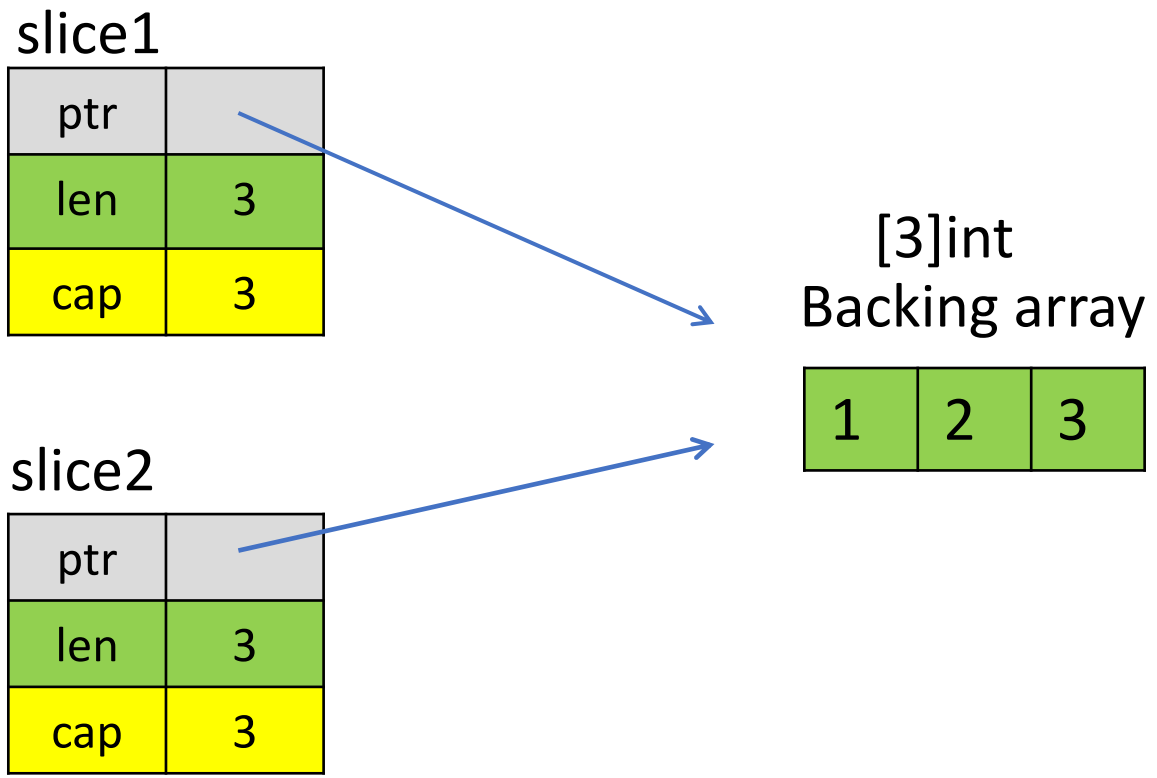
1	2	3

[4]int
New backing array

1	2	3	4		

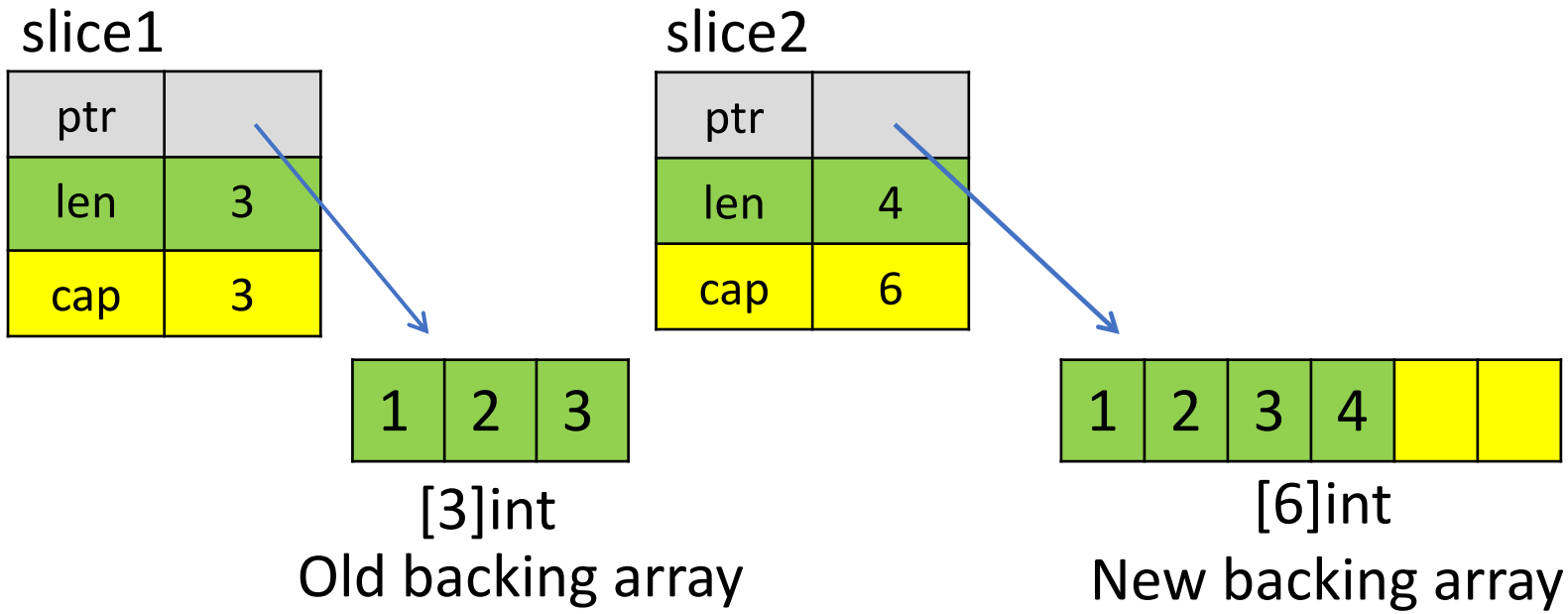
Slices in Go – possible “gotcha”

```
slice1 := []int{1,2,3}
slice2 := slice1
```



Slices in Go – possible “gotcha”

```
slice2 = append(slice2, 4)
```



Slices in Go – slicing slice

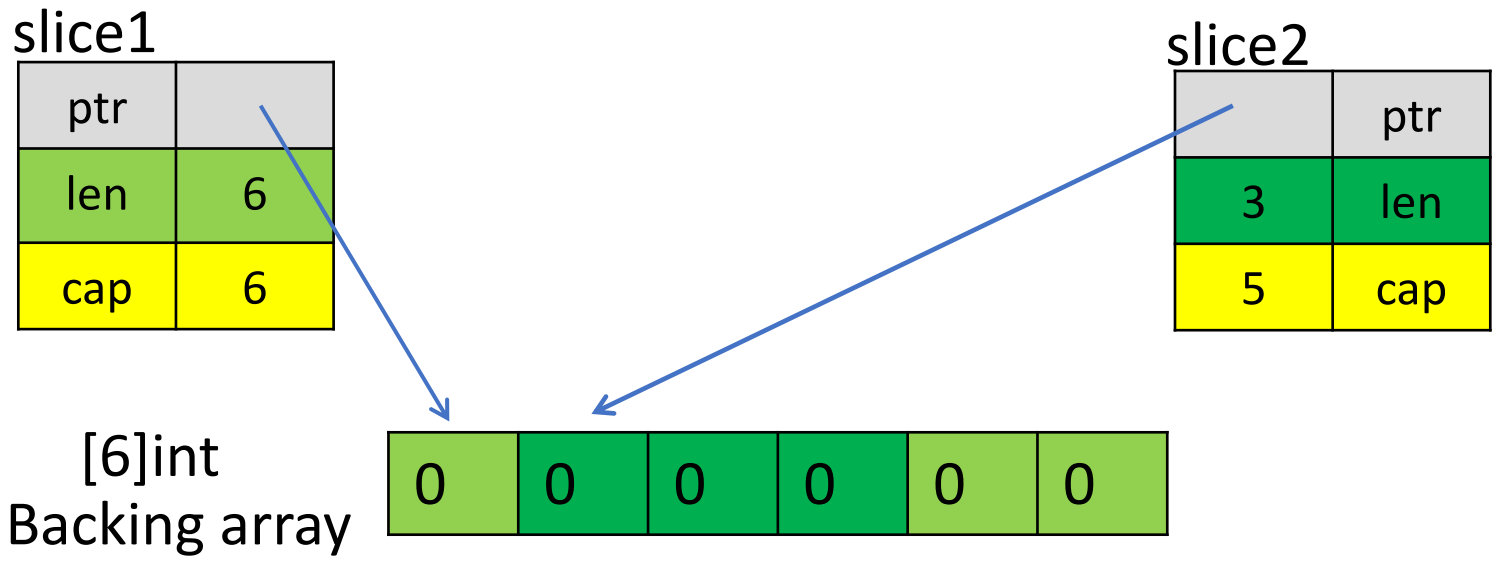
```
slice[A:B] -> [A:B)
```

Go index A trough B, but not including B

https://www.cs.utexas.edu/users/EWD/ewd08xx/EWD831.PDF

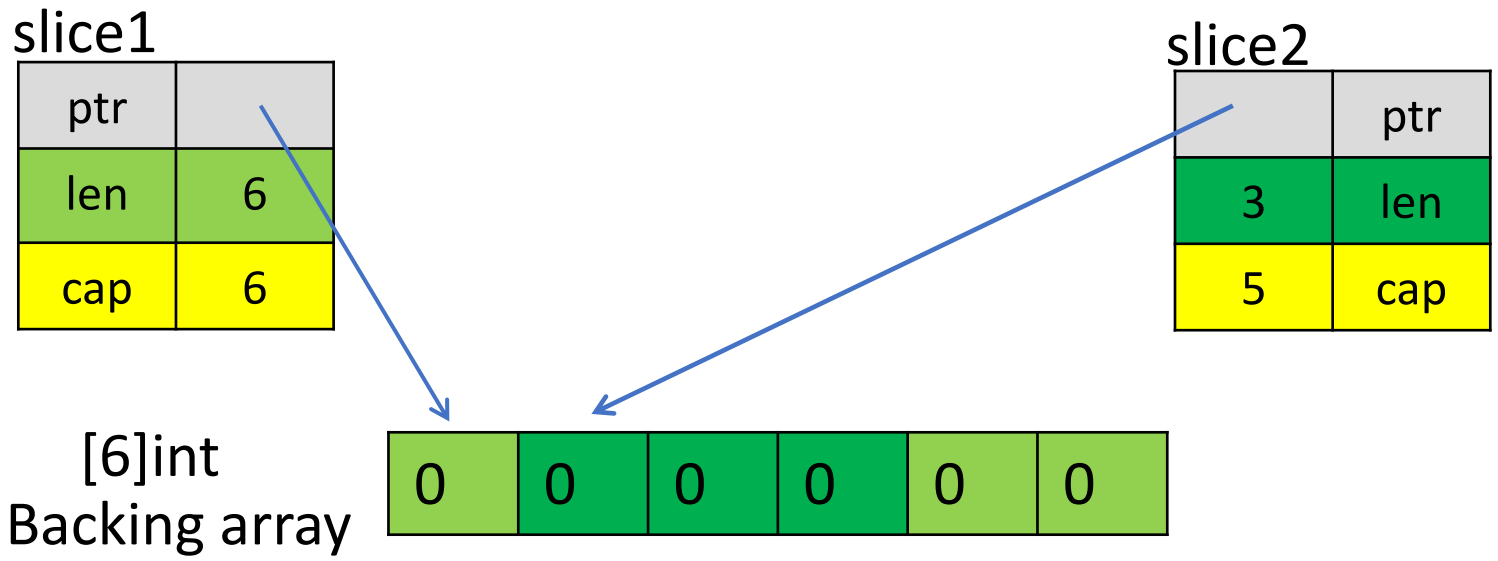
Slices in Go – slicing slice

```
slice1 := make([]int, 6)

slice2 = slice[1:4]
```



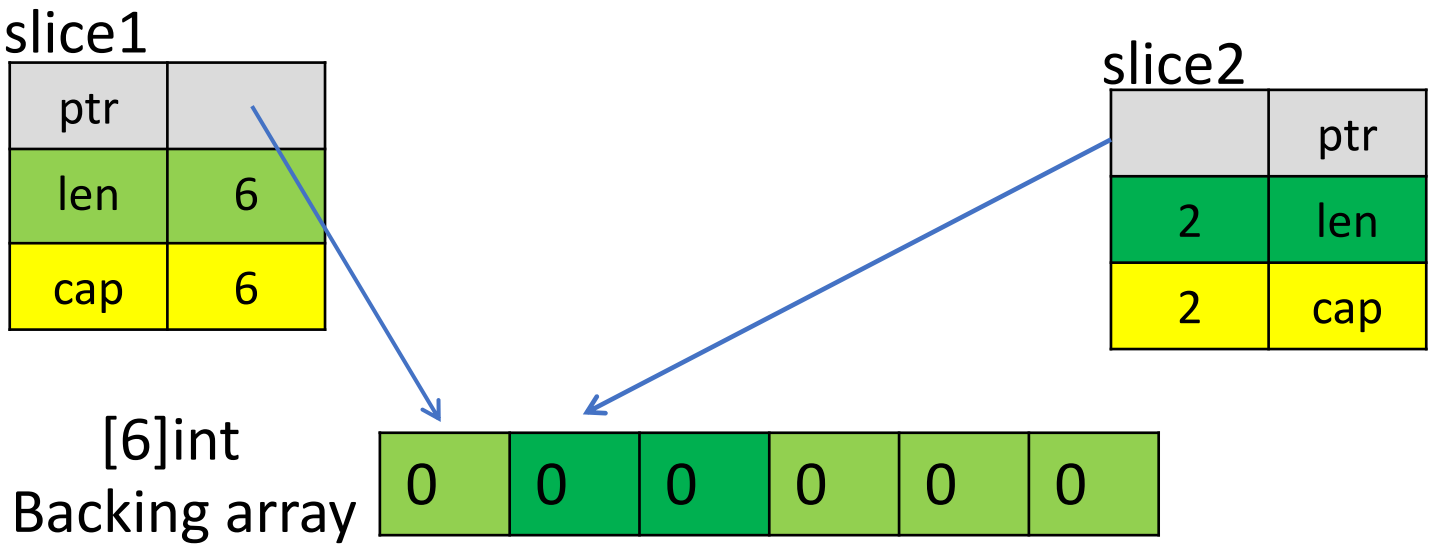
Slices in Go – slicing slice

```
slice1 := make([]int, 6)

slice2 = slice[1:1+3]
```

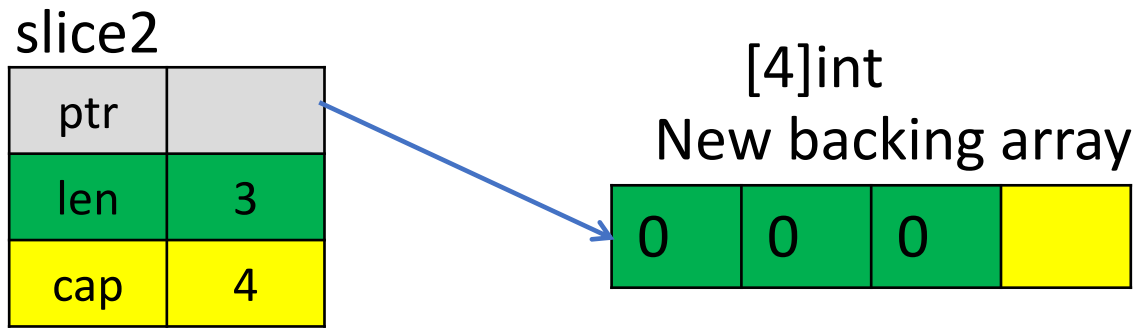


Slices in Go – slicing slice

```
slice1 := make([]int, 6)

slice2 = slice[1:1+2:2]
```




```
slice2 = append(slice2, 1)
```



Slices in Go

EXAMPLE

```
func main() {
    s := make([]int, 6)
    fmt.Printf("%v, len=%d, cap=%d\n", s, len(s), c
ap(s))

    s = s[1:4]
    fmt.Printf("%v, len=%d, cap=%d\n", s, len(s), c
ap(s))

    s = s[:cap(s)]
    fmt.Printf("%v, len=%d, cap=%d\n", s, len(s), c
ap(s))

    // panic
    s = s[:15]
}
```

https://play.golang.org/p/ej5CE4xb5ev

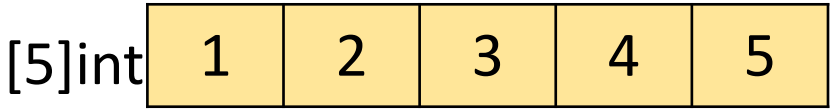
OUTPUT

```
[0 0 0 0 0 0], len=6, cap=6
[0 0 0], len=3, cap=5
[0 0 0 0 0], len=5, cap=5

panic: runtime error: slice bounds out of
range [:15] with capacity 5

goroutine 1 [running]:
main.main()
        /tmp/sandbox272174082/prog.go:1 8
+0x380
```

Slices in Go

Growing slices

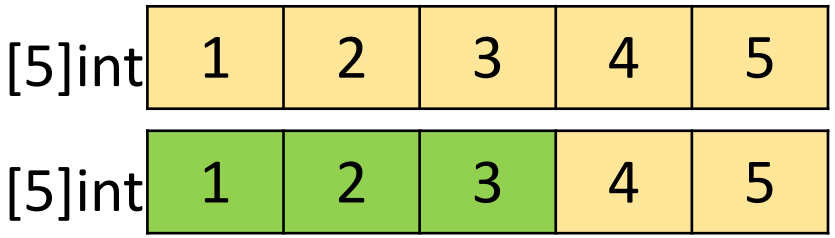
```
a := [...]int {1, 2, 3, 4, 5}
```



Slices in Go

Growing slices

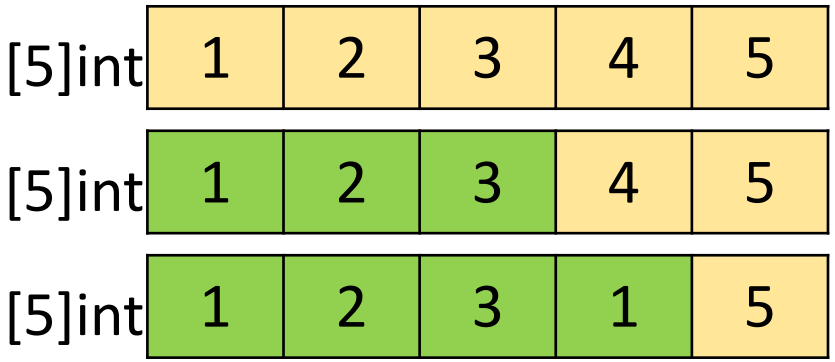
```
a := [...]int {1, 2, 3, 4, 5}

s := a[0:3]
```



Slices in Go

Growing slices

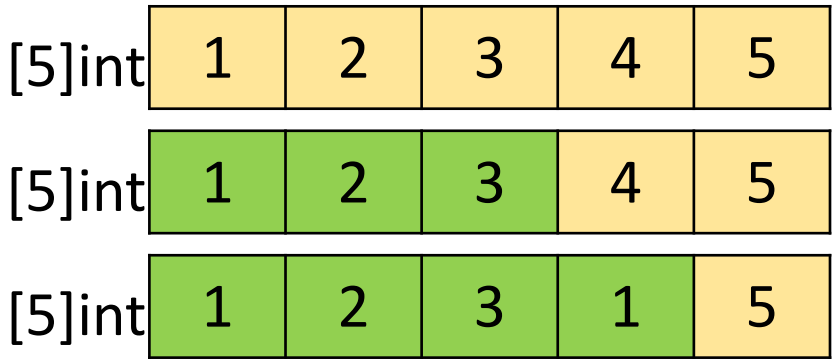
```
a := [...]int {1, 2, 3, 4, 5}

s := a[0:3]

s1 := append(s, 1)
```

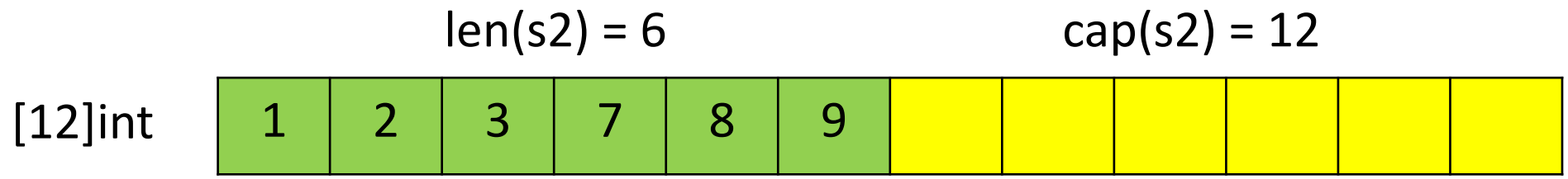


Slices in Go

Growing slices

```
a := [...]int {1, 2, 3, 4, 5}

s := a[0:3]

s1 := append(s, 1)

s2 := append (s, 7, 8, 9)
```





Slices in Go

EXAMPLE

```
func main() {
    a := [...]int{1, 2, 3, 4, 5}
    fmt.Println("a:", a)
    s := a[0:3]
    fmt.Println("s:", s, "len=", len(s), "cap=", cap(
s))
    s1 := append(s, 1)
    fmt.Println("s1:", s1, len(s), cap(s))
    fmt.Println("a:", a)
    s2 := append(s, 7, 8, 9)

    fmt.Println("a:", a)
    fmt.Println("s:", s, len(s), cap(s))
    fmt.Println("s1:", s1, len(s1), cap(s1))    fmt.P
rintln("s2:", s2, len(s2), cap(s2))
}
```

OUTPUT

```
a: [1 2 3 4 5]
s: [1 2 3] len= 3 cap= 5
s1: [1 2 3 1] 3 5
a: [1 2 3 1 5]
a: [1 2 3 1 5]
s: [1 2 3] 3 5
s1: [1 2 3 1] 4 5
s2: [1 2 3 7 8 9] 6 12
```

https://play.golang.org/p/6_a2MeGOGBg

Slices in Go

EXAMPLE

```
func main() {
    var x []int

    for i := 0; i < 10; i++ {
        x = append(x, i)
        fmt.Printf("%d cap=%d\t%v\n"
, i, cap(x), x)
    }
}
```

OUTPUT

```
0 cap=2 [0]
1 cap=2 [0 1]
2 cap=4 [0 1 2]
3 cap=4 [0 1 2 3]
4 cap=8 [0 1 2 3 4]
5 cap=8 [0 1 2 3 4 5]
6 cap=8 [0 1 2 3 4 5 6]
7 cap=8 [0 1 2 3 4 5 6 7]
8 cap=16 [0 1 2 3 4 5 6 7 8]
9 cap=16 [0 1 2 3 4 5 6 7 8 9]
```

https://play.golang.org/p/SYurc-gV4PW

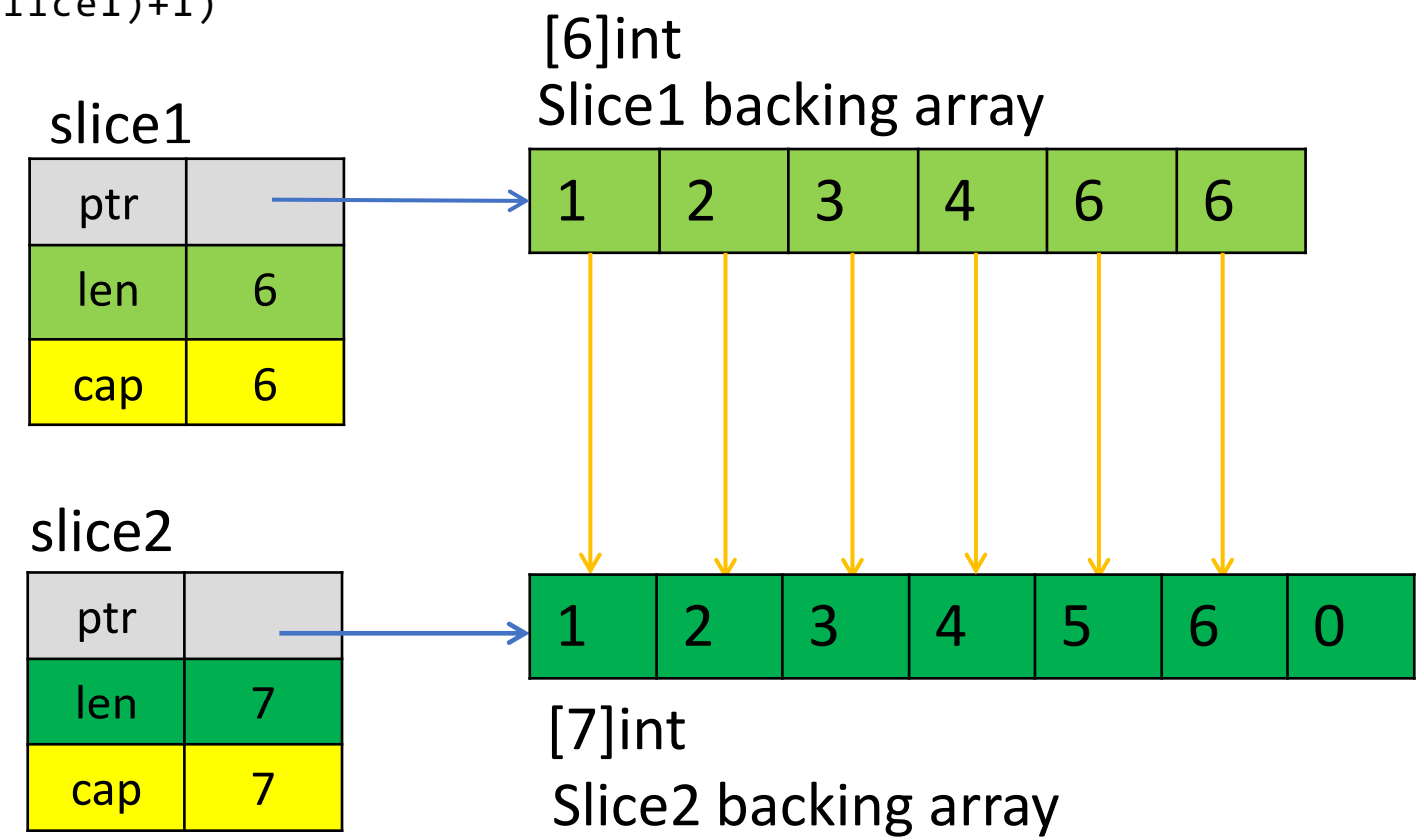
Slices in Go - Compare slices

We can compare slices only with nil

```
var s []int                 // len(s) == 0, s == nil

s = nil                     // len(s) == 0, s == nil

s = []int(nil)              // len(s) == 0, s == nil

s = []int{}                 // len(s) == 0, s != nil

s = make([]int, 0)          // len(s) == 0, s != nil
```

Slices in Go – copy slice

```
slice1 := []int{1,2,3,4,5,6}

slice2 = make([]int, len(slice1)+1)
```




```
copy(slice2, slice1)
```

Slices in Go

EXAMPLE

```
func main() {
    s := []int{1,2,3}
    fmt.Println(s)
    changeSlice(s)
    fmt.Println(s)
}

func changeSlice(s []int) {
    if len(s) > 0 {
        s[0] = 99
    }
}
```

Slices in Go

EXAMPLE

```go
func main() {
    s := []int{1,2,3}
    fmt.Println(s)
    changeSlice(s)
    fmt.Println(s)
}

func changeSlice(s []int) {
    if len(s) > 0 {
        s[0] = 99
    }
}
```

OUTPUT

```
[1 2 3]
[99 2 3]
```

https://play.golang.org/p/BMIc_4UgI68

MAPS

Maps in Go

Как и в python-е, в к-ве ключа должен быть immutable тип, поэтому slice не подходит
В отличии от питона в к-ве ключа может быть int.

Key type

Value type

map[T1]T2

Maps in Go

map[int64]string

key	value
1	“One”
2	“Two”
3	“Three”

map[[2]bool]string

key	value
{true, false}	“One”
{false, true}	“Two”
{true, true}	“Three”

Maps in Go

How to declare a map?

- Using VAR keyword

```
var map_name map[Type1]Type2

var map_name = map[Type1]Type2 {k1:v1, k2:v2, … k3:v3}
```

- Using shorthand declaration

```
map_name := map[Type1]Type2 {k1:v1, k2:v2, … k3:v3}

map_name := map[Type1]Type2 {
        k1:v1,
        k2:v2,
        k3:v3,
}
```

Maps in Go

How to declare a map?

- Using make function

```
map_name := make(map[Type1]Type2)
```

- Using new function (pointer)

```
map_name := new(map[Type1]Type2)
```

Maps in Go

EXAMPLE

```go
func main() {
    var a map[int]int
    var b = map[string]int{"1": 1, "2": 2, "3": 3}

    c := map[float32]bool{1.1: true, 2.2: false}
    d := map[[2]int]string{
        [2]int{1, 2}: "one",
        [2]int{2, 3}: "two",
    }

    e := make(map[int]int, 3)
    f := new(map[int]int)

    fmt.Println(a, b, c, d, e, f)
    fmt.Printf("%T\n", e)
    fmt.Printf("%T\n", f)
}
```

OUTPUT

```
map[]
map[1:1 2:2 3:3]
map[1.1:true 2.2:false]
map[[1 2]:one [2 3]:two]
map[]
&map[]


map[int]int
*map[int]int
```

https://play.golang.org/p/zqifR5MYYOa

Maps in Go - Add/set value in the map

EXAMPLE

```
m := map[int]string{}
m[1] = "one"
m[5] = "five"
```

key	value
1	“one”
5	“five”

```
m[5] = "six"
```

key	value
1	“one”
5	“six”

https://play.golang.org/p/y9Rb8tkQHLd

Maps in Go - Add/set value in the map

EXAMPLE

```go
func main() {
    m := map[string]int{}

    m["one"] = 1
    m["zero"] = 0

    one := m["one"]
    zero := m["zero"]
    five := m["five"]

    fmt.Println(one)
    fmt.Println(zero)
    fmt.Println(five)
}
```

https://play.golang.org/p/-oypaOhBHP9

OUTPUT

```
1
0
0
```

Maps in Go - Second return value

EXAMPLE

```
func main() {
    m := map[string]int{"zero": 0, "one": 1}


    zero, ok := m["zero"]
    if !ok {
        fmt.Println("map doesn't contain `ze
ro`")   }

    five, ok := m["five"]
    if !ok {
        fmt.Println("map doesn't contain `fi
ve`")   }

    _, ok = m["five"]
    five, _ = m["five"] // == five = m["five
"]  fmt.Println(zero, five)
}
```

OUTPUT

```
map doesn't contain `five`
0 0
```

https://play.golang.org/p/GaEX7sTPOVx

Maps in Go - nil maps

EXAMPLE

```go
func main() {
    var m map[string]int // nil map

    a, ok := m["a"]

    fmt.Println(a, ok)

    // panic, map is nil
    m["a"] = 1
}
```

OUTPUT

```
0 false
panic: assignment to entry in nil map

goroutine 1 [running]:
main.main()
        /tmp/sandbox925121883/prog.go:14 +0x120
```

https://play.golang.org/p/neHLVwhRSof

Maps in Go - Delete from the map

EXAMPLE

```
func main() {
    m := map[string]int{"one":
 1, "two": 2}
    fmt.Println(m, len(m))

    delete(m, "three")
    fmt.Println(m, len(m))

    delete(m, "one")
    fmt.Println(m, len(m))

    var nilMap map[int]int
    //no panics
    delete(nilMap, 1)
}
```

OUTPUT

```
map[one:1 two:2] 2
map[one:1 two:2] 2
map[two:2] 1
```

https://play.golang.org/p/a8d_baOeYB3

Maps in Go - Elements are not addressable

EXAMPLE

```
func main() {
    m := map[string]int{"one": 1, "two": 2}

    p := &m["one"]
    fmt.Println(p)
}
```

OUTPUT

```
./prog.go:10:7: cannot take the address of
m["one"]
```

https://play.golang.org/p/mPieIihaBAl

Maps in Go - Map iteration

EXAMPLE

```
func main() {
    m := map[string]int{"one": 1
, "two": 2}

    for k, v := range m {
        fmt.Printf("key = %v, va
l = %v\n", k, v)
    }

    fmt.Println()
    for k := range m {
        fmt.Printf("key = %v\n",
 k)
    }
}
```

https://play.golang.org/p/R9fbBe96bM5

OUTPUT

```
key = one, val = 1
key = two, val = 2

key = two
key = one
```

Maps in Go - Reference type

EXAMPLE

```
func main() {
    m := map[string]int{"one": 1, "
two": 2}
    fmt.Println(m)
    changeMap(m)
    fmt.Println(m)
}


func changeMap(m map[string]int) {
    m["one"] = 11
    m["three"] = 3
    delete(m, "two")
}
```

https://play.golang.org/p/bRMnsb7FVtJ

OUTPUT

```
map[one:1 two:2]
map[one:11 three:3]
```

Maps in Go - unordered

EXAMPLE

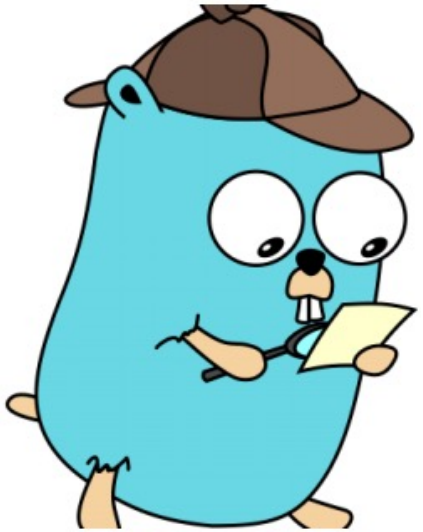
```
func main() {
    pt := map[int]string{
        1: "Helium",
        2: "Hydrogen",
        3: "Lithium",
        4: "Beryllium",
        5: "Boron",
        6: "Carbon",
    }
    for i := 1; i <= 3; i++ {
        fmt.Printf("Iteration: %v\n", i
)
        printMap(pt)
    }
}

func printMap(m map[int]string) {
    for i := range m {
        fmt.Printf("key: [%v] value: [%
v]\n", i, m[i])
    }
}
```

https://play.golang.org/p/ZOw_a0IJhBv

OUTPUT

```
Iteration: 1
key: [3] value: [Lithium]
key: [4] value: [Beryllium]
key: [5] value: [Boron]
key: [6] value: [Carbon]
key: [1] value: [Helium]
key: [2] value: [Hydrogen]
Iteration: 2
key: [6] value: [Carbon]
key: [1] value: [Helium]
key: [2] value: [Hydrogen]
key: [3] value: [Lithium]
key: [4] value: [Beryllium]
key: [5] value: [Boron]
Iteration: 3
key: [1] value: [Helium]
key: [2] value: [Hydrogen]
key: [3] value: [Lithium]
key: [4] value: [Beryllium]
key: [5] value: [Boron]
key: [6] value: [Carbon]
```

Maps in Go - Useful tricks



- len(m)
- Reference type (can be nil)
- Compare is the same as for slices

SORT

Sort package

EXAMPLE

```
package main

import (
    "fmt"
    "sort"
)

func main() {
    s := []int{5, 2, 6, 3, 1, 4} // uns
orted
    sort.Ints(s)
    fmt.Println(s)
}
```

FUNCTIONS

- func Float64s(x []float64)
- Float64sAreSorted(x []float64) bool
- Ints(x []int)
- IntsAreSorted(x []int) bool
- IsSorted(data Interface) bool
- Search(n int, f func(int) bool) int
- SearchFloat64s(a []float64, x float64) int
- SearchInts(a []int, x int) int
- SearchStrings(a []string, x string) int
- Slice(x interface{}, less func(i, j int) bool)
- SliceIsSorted(x interface{}, less func(i, j int) bool) bool
- SliceStable(x interface{}, less func(i, j int) bool)
- Sort(data Interface)
- Stable(data Interface)
- Strings(x []string)
- StringsAreSorted(x []string) bool

Sort package – sort a collection of integers

EXAMPLE

```
package main

import (
    "fmt"
    "sort"
)
func main() {
    intSlice := []int{4, 5, 2, 1, 3, 9, 7, 8, 6, 10}
    fmt.Println(sort.IntsAreSorted(intSlice))        // false
    sort.Ints(intSlice)
    fmt.Println(intSlice)
       // [1 2 3 4 5 6 7 8 9 10]
    fmt.Println(sort.IntsAreSorted(intSlice))        // true
}
```

Sort custom data structures with Len, Less, and Swap

```go
package main

import (
    "fmt"
    "sort"
)

type LengthBasedStrings []string

func (s LengthBasedStrings) Len() int {
    return len(s)
}
func (s LengthBasedStrings) Swap(i, j int) {
    s[i], s[j] = s[j], s[i]
}
func (s LengthBasedStrings) Less(i, j int) bool {
    return len(s[i]) < len(s[j])
}

func main() {
    words := []string{"never", "gonna", "give", "you", "up", "never", "gonna", "let", "you", "down"}
    sort.Sort(LengthBasedStrings(words))
    fmt.Println("Sorting by Length:", words)
          //Sorting by Length: [up let you you give down never never gonna gonna]
}
```

Questions

Homework

Homework

Task 1: Arrays

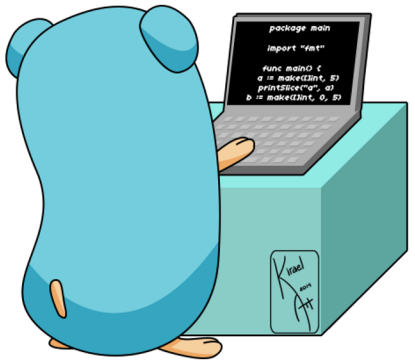
Implement function that returns an average value of array (sum / N)
input -> [1,2,3,4,5,6]
output -> 3.5

Task 2: Slices

function that returns the copy of the original slice in reverse order. The type of elements is int64.
Input -> (1, 2, 5, 15)
Output -> (15, 5, 2, 1)

Task 3: Maps

function that returns map values sorted in order of increasing keys.
Input -> {2: "a", 0: "b", 1: "c"}
Output -> ["b", "c", "a"]
Input -> {10: "aa", 0: "bb", 500: "cc"}
Output -> ["bb", "aa", "cc"]



Thanks