Date: / / /

(62) stack operations with slices, how do you remove elements from slice

for ex:-

Remove an element from end

```
func main() {
    a := []int{1,2,3,4,5}
    b := a[:len(a)-1]
    fmt.Printf(a)
}
```

Output
[1 2 3 4]

Remove an element from beginning

```
func main() {
    a := []int{1,2,3,4,5}
    b := a[1:]
    fmt.Println(b)
}
```

Output
[2 3 4 5]

Remove an element from middle

```
func main() {
    a := []int{1,2,3,4,5}
    b := append(a[:2], a[3:]...)
    fmt.Println(b)
}
```

Output
[1 2 4 5]

(63) what you do of you have a situation like this-

```
func main() {
    a := []int{1,2,3,4,5}
    fmt.Println(a)
    b := append(a[:2], a[3:]...)
    fmt.Println(b)
    fmt.Println(a)
}
```

Output
[1 2 3 4 5]
[1 2 4 5]
→ [1 2 4 5 5]

Here we working with the refernces of same underline array because these are all slicing operation, there is one underlying array and everything is being done on that underlying

Date / / /

(66) len function returns length of slice
cap function returns capacity of underlying array
append function to add elements to slice
- May cause expensive copy operation if underlying array is too small.

Copy refer to same underlying array

(67) How to declare maps (describe in key value format) Pair

for ex:-

```
func main () {
    statepopulations := maps[string] int {
        "California": 39250017,
        "Texas":      27852596,
    }
    fmt.Println(statepopulations)
}
```

list the type of key
list the type of value
key type string
value type int

output:

map[California:39250017 Texas:27852596]

We have a lot of options with key type but not infinite. ~~keys~~
Basic constraints on keys when you are creating map is that they have to be tested for equality.

- > Boolean
- > All Numeric type
- > Strings
- > pointer
- > interfaces
- > structs
- > arrays
- > channels

→ These all are tested for equality / equivalence check

- > Slices
- > maps
- > other functions

Not supported for equivalency check-in

Date / / /

is
array. So this, something to be very sensitive about. So if you are removing elements from the inside of the slice using the command like we have

b := append(a[:2], a[3:]...)

make sure that you don't have any other references to that underlying array otherwise you are going to get some unexpected behaviour.

To fix this we don't have a tool right now we can apply loop later we will learn.

(64) Declaration Style on array

a := [3] int {1,2,3}
a := [...] int {1,2,3} → This uses if you have to update the size of the array later then [...] will automatically update.
var a [3] int
↳ Each one of the integers are going to store with the value zero.

(65) Creation Styles of Slices:
→ Slice existing array or slice
→ Literal style → a := [] int {1,2,3}
→ via make function
: a := make([]int, 10) // create slice with capacity and length == 10
: a := make([]int, 10, 100) // " " and capacity = 100

The Size of slice is dynamic and so it can be determined at runtime.

(70) How to add a value in map-

forex:-

```
func main() {
    Statepopulation := make(map[string]int)
                    = map[string]int{
        "Rajasthan": 4000052,
    }
    Statepopulation["Karnataka"] = 20279436 7
    fmt.Println(Statepopulation["Karnataka"])
}
```

output
20279436 7

(71) If you are iterating through maps later on the return order of maps is not guaranteed.

forex:-

```
func main() {
    Statepopulations := make(map[string]int)
    Statepopulations = map[string]int{
        "Georgia": 25279543,
    }
    fmt.Println(Statepopulations)
    Statepopulations["Florida"] = 1097543
    fmt.Println(Statepopulations)
}
```

output

map[Georgia:25279543]
map[Florida:1097543 Georgia:25279543]

↑ The return order has changed.

forex:-

```
func main() {
    m := map[[]int]string{}
    fmt.Println(m)
}
```

map can't created as []int is not a valid key type

output:-
You will get an error
invalid map key type []int.

```
func main() {
    m := map[[3]int]string{}
    fmt.Println(m)
}
```

map can be created coz array is a valid key type

output:-
map[]

(68) Another Example for declaring maps by built-in make function

forex:-

```
func main() {
    Statepopulations := make(map[string]int)
    Statepopulations = map[string]int{
        "California": 25542397,
    }
    fmt.Println(Statepopulations)
}
```

output
map[California:25542397]

You may use this syntax if you are working on loop.

☆ Maps are always gonna be the length of the no of elements in it.

(69)

```
func main() {
    Statepopulations := make(map[string]int)
    Statepopulations = map[string]int{
        "California": 27754239,
    }
    fmt.Println(Statepopulation["California"])
}
```

output
27754239

built-in Date: / / /

(72) How to delete value from map. By delete function

for ex:-

```
func main () {
statepopulations := make (map[string] int)
statepopulations = map [string] int {
        "California" : 39250017,
        "Texas" : 27862596,
    }
statepopulations["Georgia"] = 1031037 1
delete (statepopulations, "Georgia")
fmt.println ( statepopulations )
}
```

output :-

map [California: 39250017 Texas: 27862596]

(73) If the key is not present in the map then it returns value 0.

for ex:-

```
func main () {
    s := map [string] int {
    "California" : 3927506,
    }
    fmt.println(s["California"])
}
```

output
0.

missing 0

If you go through such scenario where you are not sure if key is present or not then ~~go~~ use below syntax:-

```
func main () {
    s := map [string] int {
    "California" : 392543,
    }
    pop, ok := s["California"]
    fmt.println (pop, ok)
}
```

output
392543 true