GO Map

In Go, Maps is an unordered collection of key and its associated value.

They are very good for looking up fast.

The key type must have the operations == and != defined , like string, int, float.

Hence arrays, slices and structs cannot be used as key type, but pointers and interface types can.

Struct can be used as a key when we provide Key() or Hash() method, so that a unique numeric or string key can be calculated from the struct's fields.

A map is a reference type and declared in general as:

```
var map1 map[KeyType]ValueType
```

Ex :-

```
var map1 map[int]string
```

Go Map Example :-

```
package main

import "fmt"

func main() {
    x := map[string]int{
        "Kate": 28,
        "Jhon": 37,
        "Raj":  20,
    }
    fmt.Println(x)
```

```
    fmt.Println("\n", x["Raj"])
}
```

Output :-

```
map[Jhon:37 Kate:28 Raj:20]

 20
```

Go Map Insert and Update Operation.

Update and insert operation are similar in go map.

If the map does not contain the provided key the insert operation will takes place and if the key is present in the map then update operation takes place.

```
package main

import "fmt"

func main() {
    m := make(map[string]int)
    fmt.Println(m)
    m["Key1"] = 10
    m["key2"] = 20
    m["key3"] = 30
    fmt.Println(m)
    m["Key2"] = 5555

    fmt.Println(m)
}
```

Output :-

```
map[]
map[Key1:10 key2:20 key3:30]
map[Key1:10 Key2:5555 key2:20 key3:30]
```

Go Map Delete Operation.

- Syntax :-

```
delete(map,key)
```

Example :-

```
package main

import "fmt"

func main() {
    m := make(map[string]int)
    m["Key1"] = 10
    m["Key2"] = 20
    m["Key3"] = 30
    fmt.Println(m)
    delete(m, "Key3")
    fmt.Println(m)
}
```

Output :-

```
map[Key1:10 Key2:20 Key3:30]
map[Key1:10 Key2:20]
```

Go Map Retrieve Element

- Syntax

```
elem = m[key]
```

Example :-

```
package main

import "fmt"

func main() {
    m := make(map[string]int)
    m["Key1"] = 10
    m["Key2"] = 20
    m["Key3"] = 30
    fmt.Println(m)
    fmt.Println("The value: ", m["Key2"])
}
```

Output :-

```
map[Key1:10 Key2:20 Key3:30]
The value:  20
```

- We can also test if a key is present in the table with two value example

  ➔ Syntax :-

```
elem,ok = m[key]
```

- If key is not present, then the value of elem is the default value of element type.

- If the type of elem is int then value of elem is zero.

Example:-

```
package main

import "fmt"

type Vertex struct {
    Latitude, Longitude float64
}

var m = map[string]Vertex{
    "XYZ": Vertex{40.68433, -74.39967},
    "ABC": Vertex{37.42202, -122.08408},
}

func main() {
    fmt.Println(m)
}
```

Output :-

```
map[Key3:30 key1:10 key2:20]
The value:  20 Present?  true
he Value:  0 Present?  false
```

- In Go Maps are like struct, But it requires keys.

Go Map of Struct

```go
package main

import "fmt"

type Vertex struct {
    Latitude, Longitude float64
}

var m = map[string]Vertex{
    "XYZ": Vertex{40.68433, -74.39967},
    "ABC": Vertex{37.42202, -122.08408},
}

func main() {
    fmt.Println(m)
}
```

Output :-

```
map[ABC:{37.42202 -122.08408} XYZ:{40.68433 -74.39967}]
```