Introduction to Programming in Go

6. Maps

Module Topics

1. Maps in Go
2. Declaring and Creating Maps
3. Working with Map Entries
4. Map Iteration

Maps in Go

Maps in Go Basics

1. A map in Go is a reference to an underlying hash table.
2. Map elements are (key, value) pairs declared as

   ```
   var map1 map[keytype]valuetype
   ```

3. Maps have types defined by combining the key type and value type.
4. Key type must have == and != operators defined.
5. Valuetype can be any Go type.
6. Maps are fast – access in near constant time.
7. Maps are passed by reference to functions.

Declaring and Creating Maps

Declaring and Creating Maps

1. Maps are created using the make() function.
2. Maps can also be created by providing a map literal.
3. map[key] = value overwrites value if the key already exists.
4. map[key] = value adds value if the key does not exist.
5. Duplicate keys in a map literal are not allowed.
6. Maps can be declared with an initial capacity.

Creating Maps with Literals

```
// Example 06-01 Creating maps with literals
...
var errs map[int]string

func main() {
   severity := map[string]string{
      "Blue":   "normal",
      "Orange": "moderate",
      "Red":    "severe"}

   severity["Black"] = "apocalyptic"

   fmt.Println(severity, " size =", len(severity))
   fmt.Println(errs, " size =", len(errs))
}
```

```
[Module06]$ go run ex06-01.go
map[Blue:normal Orange:moderate Red:severe Black:apocalyptic]
size = 4
map[]  size = 0
```

Creating Maps with make()

```
//Example 06-02 Creating maps with make()
...
var errs map[int]string

func main() {
   errs = make(map[int]string)
   errs[0] = "Hardware"
   errs[1] = "Segmentation fault"
   fmt.Println(errs, " size =", len(errs))
   errs[0] = "Firmware fault"
   fmt.Println(errs, " size =", len(errs))
   fmt.Println("The errorcode '0' is a ", errs[0])
}
```

```
[Module06]$ go run ex06-02.go
map[1:Segmentation fault 0:Hardware]  size = 2
map[0:Firmware fault 1:Segmentation fault]  size = 2
The errorcode '0' is a  Firmware fault
```

Empty versus nil Maps

```
// Example 06-03 Empty versus nil maps
...

var errs map[string]string

func main() {
   fmt.Println("Check for nil before make() ", errs == nil)
   fmt.Println("Length of errs ", len(errs))

   errs = make(map[string]string)

   fmt.Println("Check for nil after make() ", errs == nil)
   fmt.Println("Length of errs ", len(errs))
}
```

```
[Module06]$ go run ex06-03.go
Check for nil before make()  true
Length of errs  0
Check for nil after make()  false
Length of errs  0
```

Map with Initial Capacity

```
// Example 06-04 Map capacity
...

var errs map[string]string

func main() {
   errs = make(map[string]string, 200)
   fmt.Println("Length of errs ", len(errs))
}
```

```
[[Module06]$ go run ex06-4.go
Length of errs  0
```

Working with Map Entries

Working with Map Entries

1. map[key] always returns a value.
2. If the key does not exist, the default zero of the value type is returned.
3. The more complete form is

```
value, bool = map[key]
```

4. The bool value is often assigned to the variable “ok”.
5. If “ok” is false, the key is not in the map.
6. Attempting to delete a non-existent element causes a panic.

The Comma ok Idiom

```
// Example 06-05  comma ok update
...

func update(m map[int]int, key int, val int) {
   _, ok := m[key]
   if ok {
      m[key] = val
   }
}

func main() {
   data := map[int]int{1: 100, 3: 300}
   update(data, 1, -1)
   update(data, 2, 200)
   fmt.Println(data)
}
```

```
[Module06]$ go run ex06-05.go
map[1:-1 3:300]
```

Map Iteration

Map Iteration

```
// Example 06-06 Iteration
...

func main() {

   dotw := map[string]int{
      "Sun": 1, "Mon": 2, "Tue": 3, "Wed": 4,
      "Thu": 5, "Fri": 6, "Sat": 7}
   for day, num := range dotw {
      fmt.Printf("(%s : %d) ", day, num)
   }
}
```

```
[Module06]$ go run ex06-0c.go
(Thu : 5) (Fri : 6) (Sat : 7) (Sun : 1) (Mon : 2) (Tue : 3) (Wed : 4)
```

Lab #6: Maps