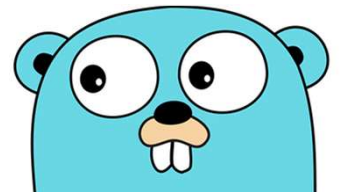


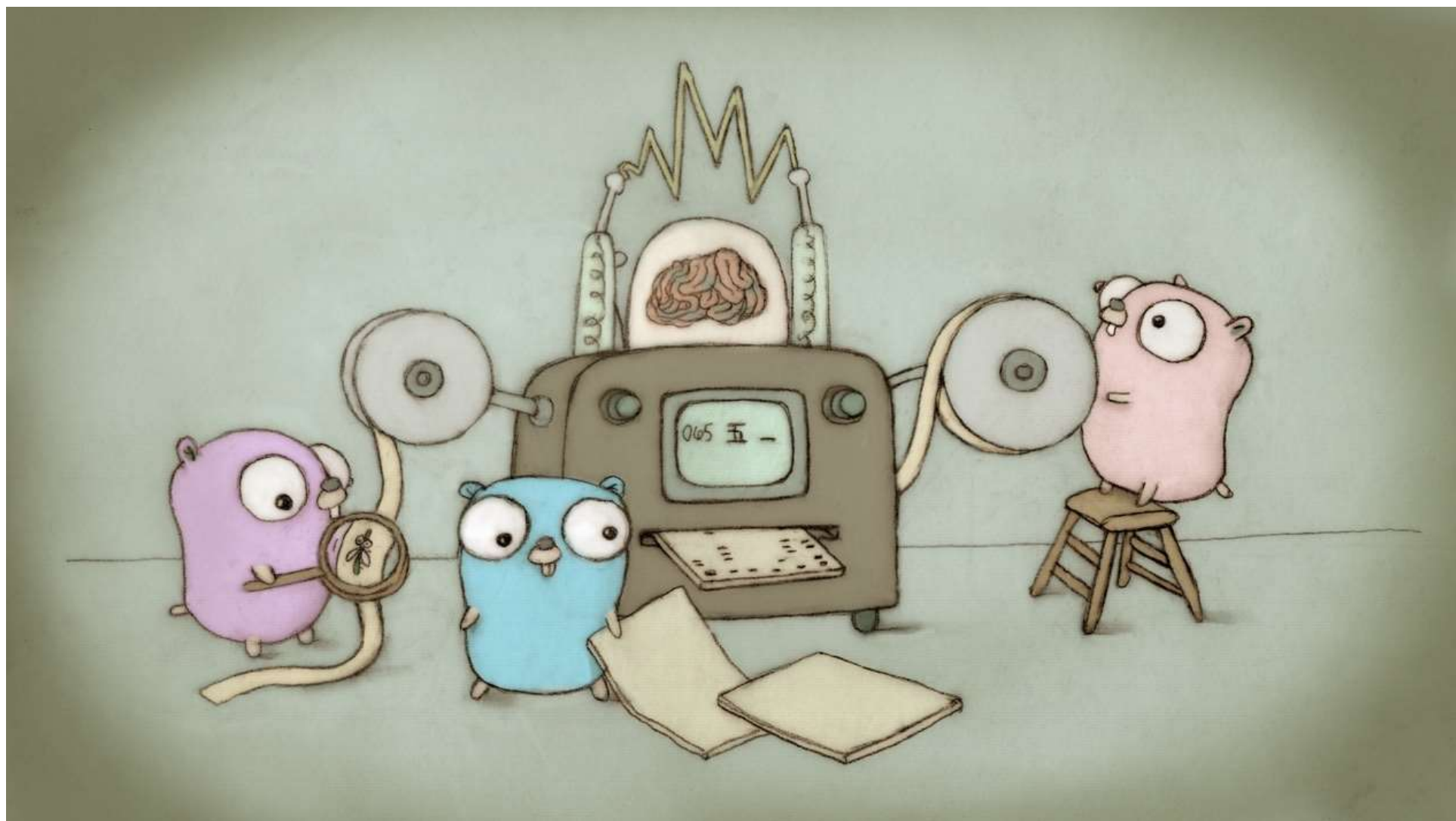
Go.Next()

Radical Rakhman Wahid & Amir Muhammad Hakim



Go Programming Course #2
October 25th 2020 / Rabbi'ul Awwal 8st 1442





Bahasan hari ini

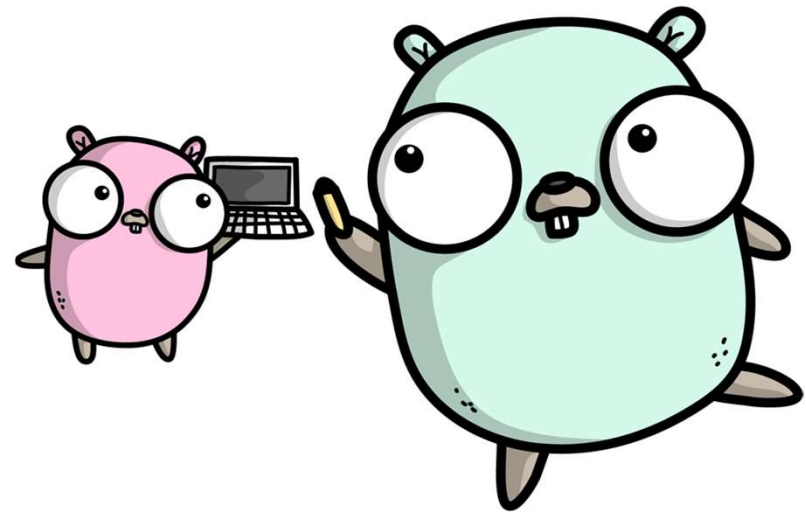
Percabangan

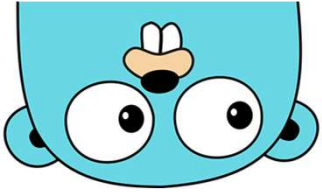
- Operasi perbandingan & Operasi logika
- if statements (if | else if | else)
- switch-case

Collections

- Arrays
- Slices
- Maps

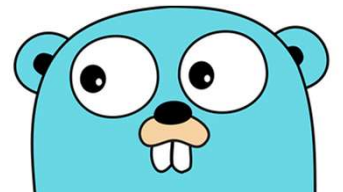
Range (For-loop khusus collections)





Percabangan

if statements & switch-case



Operasi perbandingan

- Operasi perbandingan adalah operasi untuk **membandingkan dua buah data**
- Operasi perbandingan selalu **menghasilkan nilai boolean** (true or false)
- Jika operasi **benar**, hasilnya adalah **true**
- Jika operasi **salah**, hasilnya adalah **false**

Operator	Penjelasan	Contoh & Hasil	
==	Sama dengan	2 == 2	true
!=	Tidak sama dengan	3 != 2	true
<	Kurang dari	2 < 3	true
<=	Kurang dari atau sama dengan	3 <= 3	true
>	Lebih dari	3 > 4	false
>=	Lebih dari atau sama dengan	5 >= 4	true

Operasi perbandingan

```
// Membandingkan 2 string
const name1 = "udin"
const name2 = "udin"

const isEqual = name1 == name2

fmt.Println(isEqual) // true
```

```
// Membandingkan 2 buah integer
const num1 = 2
const num2 = 3

fmt.Println(num1 > num2) // false
fmt.Println(num1 < num2) // true
fmt.Println(num1 == num2) // false
fmt.Println(num1 != num2) // true
fmt.Println()
```

```
// Membandingkan 2 boolean
const bool1 = true
const bool2 = false

fmt.Println(bool1 == bool2) // false
fmt.Println(bool1 != bool2) // true
```

Operasi logika

- Operasi logika adalah **operasi logika untuk tipe data boolean**, meliputi:
 - Operasi perbandingan antar boolean (w/ AND atau OR)
 - Operasi Negasi dari nilai boolean
- Operasi logika selalu **menghasilkan nilai boolean** (true or false)
- Jika operasi **benar**, hasilnya adalah **true**
- Jika operasi **salah**, hasilnya adalah **false**

Operator	Penjelasan	Contoh & hasil
&&	AND	<i>Nanti</i>
	OR	<i>Nanti</i>
!	Negasi	!true → false

Operasi Logika

Operator && (AND)

Nilai 1	Operator	Nilai 2	Hasil
true	&&	true	true
true	&&	false	false
false	&&	true	false
false	&&	false	false

Operasi Logika

Operator || (OR)

Nilai 1	Operator	Nilai 2	Hasil
true		true	true
true		false	true
false		true	true
false		false	false

Operasi Logika

```
// Operasi logika 2 buah boolean  
fmt.Println(false && true) // false  
fmt.Println(false || true) // true  
fmt.Println(!true) // false
```

```
// Operasi logika 3 buah boolean  
fmt.Println(false && true || true) // true
```

```
// Kombinasi operasi logika & perbandingan  
const nilai = 80  
const absensi = 85  
  
const lulus = nilai >= 80 && absensi >= 90  
  
fmt.Println("Lulus:", lulus) // Lulus: false
```

if statements

Keyword if

```
if kondisi {  
    blok ini dieksekusi ketika hasil dari kondisi bernilai true  
}
```

- If adalah salah satu keyword yang **digunakan untuk melakukan percabangan**
- Percabangan maksudnya adalah **menentukan blok kode mana akan dieksekusi berdasarkan suatu kondisi** (sebagai acuannya)
- Adapun kondisi yang dijadikan acuan untuk melakukan percabangan adalah nilai bertipe bool (boolean)

if statements (keyword if)

```
const name = "udin"

if name == "udin" {
  fmt.Println("Hello udin 😊")
}

// Output:
// Hello udin 😊
```

```
const num = 1

if num == 2 {
  fmt.Println("num bernilai 2")
}

// Output:
//
```

```
const nilai = 80
const absensi = 85

if nilai >= 80 && absensi >= 75 {
  fmt.Println("Selamat, anda lulus!")
}

// Output:
// Selamat, anda lulus!
```

if statements

Keyword if-else

```
if kondisi {  
    blok ini dieksekusi ketika kondisi bernilai true  
}  
else {  
    blok ini dieksekusi ketika kondisi bernilai false  
}
```

- else adalah salah satu keyword yang digunakan untuk melakukan percabangan
- else akan dieksekusi ketika kondisi if bernilai false / tidak terpenuhi

If statements (keyword if-else)

```
const lulus = false

if lulus == true {
  fmt.Println("Anda lulus! 😊")
} else {
  fmt.Println("Anda tidak lulus! 😞")
}

// Output:
// Anda tidak lulus! 😞
```

```
const lulus = false

if lulus {
  fmt.Println("Anda lulus! 😊")
} else {
  fmt.Println("Anda tidak lulus! 😞")
}

// Output:
// Anda tidak lulus! 😞
```



if statements

Keyword `if`, `else if`, `else`

```
if kondisi_1 {  
    blok ini dieksekusi ketika kondisi_1 bernilai true  
}  
else if kondisi_2 {  
    blok ini dieksekusi ketika kondisi_1 tidak terpenuhi & kondisi_2 bernilai true  
}  
else {  
    blok ini dieksekusi ketika blok if & else if di atas tidak ada yang terpenuhi  
}
```

if statements (keyword if, else if, else)

```
const yourName = "udin"

if yourName == "kacong" {
  fmt.Println("Hello Kacong")
} else if yourName == "udin" {
  fmt.Println("Hello Udin")
} else {
  fmt.Println("Ga kenal 😊")
}

// Output:
// Hello Udin
```


if statements

Keyword if dengan inisialisasi variabel

```
if inisialisasi_variabel; kondisi {  
    blok ini dieksekusi ketika kondisi bernilai true  
    kita bisa akses variabel yang diinisialisasi disini  
}
```

- Kode menjadi lebih rapi
- Jika nilai variabel didapat dari sebuah komputasi, perhitungan tidak perlu dilakukan di dalam masing-masing blok if
- Scope variabel yang diinisialisasi hanya bisa diakses di dalam if statement itu saja

if statements (keyword if dengan inisialisasi variabel)

```
const myName = "Amir Muhammad Hakim"

const length = len(myName)


if length > 10 {
|   fmt.Printf("Nama dengan %d huruf terlalu panjang!", length)
}

// Output:
// Nama dengan 19 huruf terlalu panjang!
```

```
const myName = "Amir Muhammad Hakim"

if length := len(myName); length > 10 {
|   fmt.Printf("Nama dengan %d huruf terlalu panjang!", length)
}

// Output:
// Nama dengan 19 huruf terlalu panjang!
```



Switch-case

Basic form

```
switch nilai {  
    case nilai_1:  
        Blok kode ini dieksekusi jika nilai == nilai_1  
    case nilai_2:  
        Blok kode ini dieksekusi jika nilai == nilai_2  
    default:  
        Blok kode dieksekusi jika case-case di atas tidak ada yang terpenuhi  
}
```

- Sama seperti if. Di Go, switch-case juga melakukan percabangan
- switch-case lebih sederhana dibandingkan if statements
- Biasanya switch-case digunakan untuk mengecek 1 variabel yang digunakan untuk melakukan percabangan

Switch-case (Basic form)

```
const myAge = 21

switch myAge {
case 20:
|   fmt.Println("Umur saya dua puluh tahun")
case 21:
|   fmt.Println("Umur saya dua puluh satu tahun")
|   fmt.Println("Hari ini saya tambah umur 🥳")
default:
|   fmt.Println("Umur saya tidak diketahui")
}

// Output:
// Umur anda dua puluh satu tahun
// Hari ini saya tambah umur 🥳
```

Switch-case

Sebuah case dengan banyak kondisi

```
switch nilai {  
  case nilai_1:  
    Blok kode ini dieksekusi jika nilai == nilai_1  
  case nilai_2, nilai_3, nilai_4:  
    Blok kode ini dieksekusi jika nilai sama dengan nilai_2 atau nilai_3 atau nilai_4  
  default:  
    Blok kode dieksekusi jika case-case di atas tidak ada yang terpenuhi  
}
```

Switch-case (Sebuah case dengan banyak kondisi)

```
const yourAge = 22

switch yourAge {
  case 20:
    | fmt.Println("Umur kamu 21 tahun")
  case 21, 22, 23:
    | fmt.Println("Umur kamu 21 / 22 / 23 tahun")
  default:
    | fmt.Println("Umur kamu tidak diketahui")
}

// Output:
// Umur anda 21 / 22 / 23 tahun
```

Switch-case

switch-case dengan gaya if statements

```
switch {  
  case kondisi_1:  
    Blok kode ini dieksekusi jika kondisi_1 bernilai true  
  case kondisi_2:  
    Blok kode ini dieksekusi jika kondisi_2 bernilai true  
  default:  
    Blok kode dieksekusi jika case-case di atas tidak ada yang terpenuhi  
}
```

Switch-case (switch-case dengan gaya if statements)

```
name := "Amir Muhammad Hakim"

length := len(name)

switch {
case length > 10:
|   fmt.Printf("Nama dengan %d huruf terlalu panjang!", length)
case length > 5:
|   fmt.Printf("Nama dengan %d huruf masih terlalu panjang!", length)
}

// Output:
// Nama dengan 19 huruf masih terlalu panjang!
```


Switch-case

switch-case dengan inisialisasi variabel

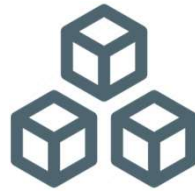
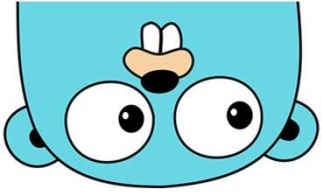
```
switch inisialisasi_variabel; kondisi {  
    case kondisi_1:  
        Blok kode ini dieksekusi jika kondisi_1 bernilai true  
    case kondisi_2:  
        Blok kode ini dieksekusi jika kondisi_2 bernilai true  
    default:  
        Blok kode dieksekusi jika case-case di atas tidak ada yang terpenuhi  
}
```

Switch-case (switch-case dengan gaya if statements)

```
const firstName = "Amir"

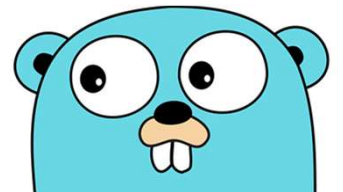
switch length := len(firstName); length > 10 {
case true:
|   fmt.Printf("Nama dengan %d huruf terlalu panjang!\n", length)
case false:
|   fmt.Println("Nama sudah pas 👍")
}

// Output:
// Nama sudah pas 👍
```



Collections

Arrays, Slices & Maps



Arrays

```
nama_array := [banyak_data]tipe_data{data_1, data_2, ....}
```

```
scores := [3]int{100, 80, 90}
```

- Array merupakan tipe data yang dapat menyimpan **kumpulan data dengan tipe yang seragam**
- Saat membuat sebuah array, kita perlu menentukan banyak data yang dapat ditampung oleh array tersebut
- Daya tampung array tidak dapat dirubah setelah array dibuat (tidak dapat ditambah / dikurangi)
- Untuk mengakses data array satu-persatu, kita dapat menggunakan index dari elemen array dipilih
- Ingat, index array dimulai dari 0!

Arrays

Inisialisasi variabel array

Index	0	1	2
Value	100	80	90

```
scores := [3]int{100, 80, 90}  
  
fmt.Println(scores) // [100 80 90]
```

```
nums := [...]int{100, 80, 90}  
  
fmt.Println(nums) // [100 80 90]
```

```
var values [3]int  
  
values[0] = 10  
values[1] = 20  
values[2] = 30  
  
fmt.Println(values)  
fmt.Println()
```

Arrays

Mengakses nilai suatu elemen array

Index	0	1	2
Value	100	80	90

```
scores := [3]int{100, 80, 90}

fmt.Println(scores[0]) // 100
fmt.Println(scores[1]) // 80
fmt.Println(scores[2]) // 90
```

Mengubah nilai suatu elemen array

```
scores = [3]int{100, 80, 90}

scores[0] = 95

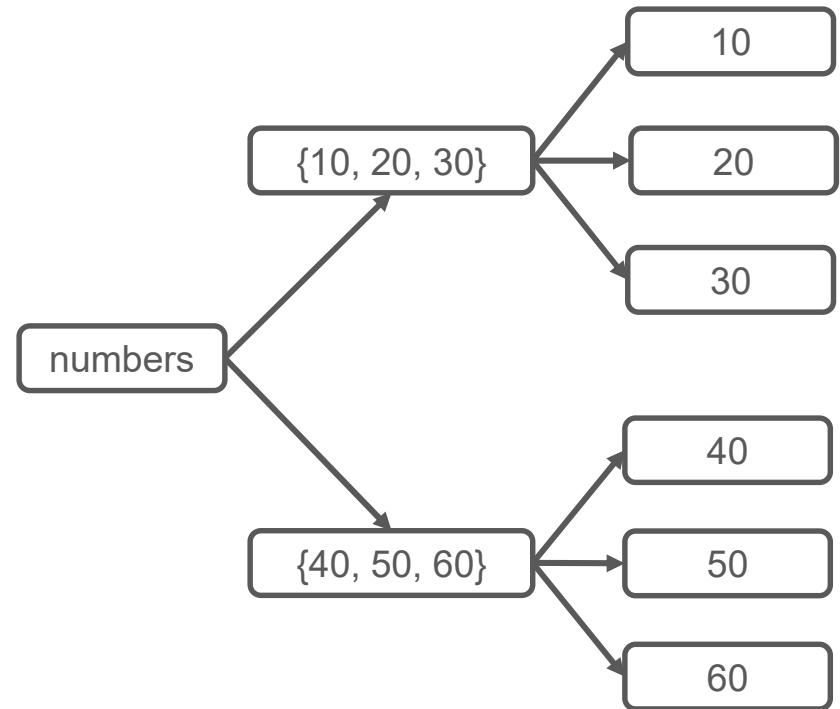
fmt.Println(scores) // [95 80 90]
```

Arrays

Array multidimensi

```
numbers := [2][3]int{
  {10, 20, 30},
  {40, 50, 60},
}

fmt.Println(numbers) // [[10 20 30] [40 50 60]]
fmt.Println(numbers[0]) // [10 20 30]
fmt.Println(numbers[0][1]) // 20
```



Arrays

Mendapatkan panjang array

```
scores = [3]int{100, 80, 90}  
fmt.Println(len(scores)) // 3
```

```
numbers = [2][3]int{  
    {10, 20, 30},  
    {40, 50, 60},  
}  
  
fmt.Println(len(numbers)) // 2
```


Slices

```
nama_slice := []tipe_data{data_1, data_2, ....}
```

```
scores := []int{100, 80, 90}
```

- Jika dilihat, **slice seperti array**
- **Slice adalah *reference* ke suatu array**, karena slice dibangun di atas array
- Slice bisa dibuat secara mandiri / bisa dibuat berdasarkan dari array yang sudah ada (*reference*)
- Karena slice merupakan data *reference*, **perubahan data di tiap elemen slice akan berdampak pada slice lain yang memiliki alamat memori yang sama**
- Tipe data slice di balik layar membawa 3 data: pointer, length & capacity
 - Pointer adalah penunjuk data pertama ke array
 - Length adalah panjang slice (banyak elemen di slice)
 - Capacity adalah kapasitas dari slice, dimana length tidak boleh lebih dari capacity

Slices

Membuat slice / inisialisasi slice

```
friends := []string{"Udin", "Sohib", "Jafar"}  
  
fmt.Println(friends)    // [Udin Sohib Jafar]  
fmt.Println(friends[1]) // Sohib  
fmt.Println()
```

Slices

Membuat slice dari array yang sudah ada

Keyword membuat slice dari array	Keterangan
array[low:high]	Membuat slice dari array dimulai index low sampai index sebelum high
array[low:]	Membuat slide dari array dimulai index low sampai index akhir di array
array[:high]	Membuat slice dari array dimulai index 0 sampai index sebelum high
array[:]	Membuat slice dari array dimulai index 0 sampai index akhir di array

Slices

Membuat slice dari array yang sudah ada

```
days := []string{
    "Senin",
    "Selasa",
    "Rabu",
    "Kamis",
    "Jumat",
    "Sabtu",
    "Minggu",
}

sliceDays := days[1:4]

fmt.Println(sliceDays)      // [Selasa Rabu Kamis]
fmt.Println(len(sliceDays)) // 3
fmt.Println(cap(sliceDays)) // 6
```

Slices

Membuat slice dari array – di balik layar

```
slice1 := array[1:4]
```

slice1	
Pointer	Index ke 1
Length	3
Capacity	6

Reference

array	
0	"Senin"
1	"Selasa"
2	"Rabu"
3	"Kamis"
4	"Jumat"
5	"Sabtu"
6	"Minggu"

Reference

```
slice2 := array[3:6]
```

slice2	
Pointer	Index ke 3
Length	3
Capacity	4

Slices

Fungsi bawaan yang dapat digunakan untuk memproses slice

Keyword membuat slice dari array	Keterangan
<code>len(slice)</code>	Mendapatkan panjang slice (banyak elemen)
<code>cap(slice)</code>	Mendapatkan kapasitas slice
<code>append(slice, data)</code>	Membuat slice baru dengan menambahkan data ke posisi terakhir slice, jika capacity slice sudah penuh, maka akan membuat array baru
<code>make([]tipe_data, length, capacity)</code>	Membuat slice baru
<code>copy(destination, source)</code>	Menyalin slice dari source ke destination

Maps

```
Nama_map := map[tipe_data_key]tipe_data_value{  
    "key1": "value",  
    "key2": "value",  
}
```

```
nums := map[int]string{  
    1: "satu",  
    2: "dua",  
}
```

- Map adalah tipe data yang berisikan **kumpulan key-value** (kata kunci - nilai) dimana **kata kuncinya bersifat unik dan tidak boleh sama**
- Untuk mengakses suatu elemen di tipe data map kita harus menggunakan key
- Berbeda dengan Array dan Slice, jumlah **data yang kita masukkan ke dalam Map boleh sebanyak-banyaknya, asalkan kata kunci nya berbeda**, jika kita gunakan kata kunci sama, maka secara otomatis data sebelumnya dengan kata kunci yang sama akan diganti dengan data baru

Maps

Membuat map / inisialisasi map

Key	"apel"	"mangga"	"jeruk"
Value	21	22	31

```
stockBuah := map[string]int{
    "apel": 21,
    "mangga": 22,
    "jeruk": 31,
}

fmt.Println(stockBuah) // map[apel:21 jeruk:31 mangga:22]
```


Maps

Mengakses dan mengubah nilai suatu elemen map

Key	"apel"	"mangga"	"jeruk"
Value	21	22	31

```
stockBuah = map[string]int{
    "apel": 21,
    "mangga": 22,
    "jeruk": 31,
}

fmt.Println(stockBuah["apel"]) // 21
fmt.Println(stockBuah["jeruk"]) // 31

stockBuah["apel"] = 50

fmt.Println(stockBuah["apel"]) // 50
```

Menghapus suatu elemen map

```
stockBuah = map[string]int{
    "apel": 21,
    "mangga": 22,
    "jeruk": 31,
}

delete(stockBuah, "jeruk")

fmt.Println(stockBuah) // map[apel:21 mangga:22]
```

Maps

Membuat map dengan fungsi make

```
stockBuah := map[string]int{  
    "apel": 21,  
    "mangga": 22,  
    "jeruk": 31,  
}  
  
fmt.Println(stockBuah) // map[apel:21 jeruk:31 mangga:22]
```

Maps

Fungsi bawaan untuk melakukan operasi pada maps

Keyword membuat slice dari array	Keterangan
<code>len(map)</code>	Mendapatkan banyak data yang ada di dalam map
<code>map[key]</code>	Mengambil nilai suatu elemen map
<code>map[key] = value</code>	Mengubah nilai suatu elemen map
<code>delete(map, key)</code>	Menghapus suatu elemen map
<code>make(map[tipe_key]tipe_value)</code>	Membuat map baru

Range

- Untuk tipe data collections (arrays, slices, maps), umumnya kita perlu mengakses semua isi elemennya satu-persatu dengan menggunakan for-loop

```
fruits := [4]string{"apple", "grape", "banana", "melon"}

for i := 0; i < len(fruits); i++ {
    fmt.Println(fruits[i])
}

// Output:
// apple
// grape
// banana
// melon
```

- Go menyediakan syntax khusus untuk tipe data collections yang bisa digunakan untuk menangani hal di atas dengan lebih clean, yakni dengan menggunakan for-range

Range

For-range pada tipe data arrays

```
fruits = [4]string{"apple", "grape", "banana", "melon"}  
  
for index, value := range fruits {  
    fmt.Println(index, value)  
}
```

For-range pada tipe data slices

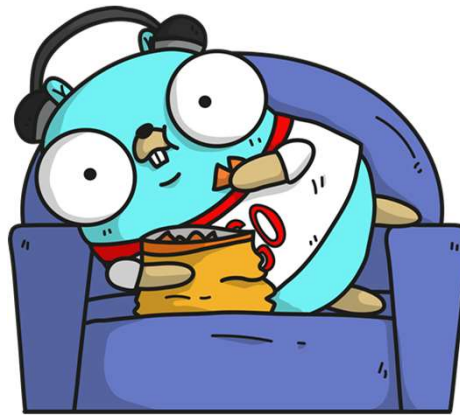
```
numbers := []int{20, 95, 100, 56}  
  
for index, value := range numbers {  
    fmt.Println(index, value)  
}
```

Range

For-range pada tipe data maps

```
stockBuah := map[string]int{
    "apel":    21,
    "mangga": 22,
    "jeruk":   31,
}

for key, value := range stockBuah {
    fmt.Println(key, value)
}
```



Sampun