

Hari 4 - Array, Slice, Map & Looping

Array

Array di dalam go adalah kumpulan data bertipe sama, yang disimpan dalam sebuah variabel. Array memiliki kapasitas yang nilainya ditentukan pada saat pembuatan, menjadikan elemen/data yang disimpan di array tersebut jumlahnya tidak boleh melebihi yang sudah dialokasikan

Berikut ini contoh penerapannya:

```
var names [4]string
names[0] = "John"
names[1] = "Doe"
names[2] = "Frank"
names[3] = "Jack"

fmt.Println(names[0], names[1], names[2], names[3])
```

Variabel `names` dideklarasikan sebagai `array string` dengan alokasi elemen `4` slot. Cara mengisi slot elemen array bisa dilihat di kode di atas, yaitu dengan langsung mengakses elemen menggunakan indeks, lalu mengisinya.

1. Inisialisasi Nilai Awal Array

seperti yang kita lihat pada contoh diatas array nya di deklarasikan terlebih dahulu lalu baru di isi datanya sesuai indeksnya, namun selain itu kita juga dapat mendeklarasikan array sekaligus mengisi nilainya, berikut ini contoh penerapannya:

```
var names = [4]string{"John", "Doe", "Frank", "Jack"}

fmt.Println(names[0], names[1], names[2], names[3])

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

fmt.Println(fruits)
```

2. Inisialisasi Nilai Awal Array Tanpa Jumlah Elemen

Deklarasi array yang nilainya diset di awal, boleh tidak dituliskan jumlah lebar array-nya, cukup ganti dengan tanda 3 titik (...). Jumlah elemen akan dikalkulasi secara otomatis menyesuaikan data elemen yang diisikan.

```
var numbers = [...]int{2, 3, 2, 4, 3}

fmt.Println("data array \t:", numbers)
fmt.Println("jumlah elemen \t:", len(numbers))
```

Variabel `numbers` akan secara otomatis memiliki jumlah elemen `5`, karena pada saat deklarasi disiapkan 5 buah elemen.

3. Array Multidimensi

rray multidimensi adalah array yang tiap elemennya juga berupa array (dan bisa seterusnya, tergantung kedalaman dimensinya).

Cara deklarasi array multidimensi secara umum sama dengan cara deklarasi array biasa, dengan cara menuliskan data array dimensi selanjutnya sebagai elemen array dimensi sebelumnya.

Khusus untuk array yang merupakan sub dimensi atau elemen, boleh tidak dituliskan jumlah datanya. Contohnya bisa dilihat pada deklarasi variabel `numbers2` di kode berikut.

```
fmt.Println("numbers1", numbers1)
fmt.Println("numbers2", numbers2)
```

Kedua array di atas memiliki elemen yang sama.

berikut ini hasil dari contoh kode diatas:

```
numbers1 [[3 2 3] [3 4 5]]
numbers2 [[3 2 3] [3 4 5]]
```

4. Alokasi Elemen Array Menggunakan Keyword **make**

Deklarasi sekaligus alokasi data array juga bisa dilakukan lewat keyword **make**.

```
var fruits = make([]string, 2)
fruits[0] = "apple"
fruits[1] = "manggo"

fmt.Println(fruits) // [apple manggo]
```

Parameter pertama keyword **make** diisi dengan tipe data elemen array yang diinginkan, parameter kedua adalah jumlah elemennya. Pada kode di atas, variabel **fruits** tercetak sebagai array string dengan alokasi 2 slot.

Slice

Slice adalah *reference* elemen array. Slice bisa dibuat, atau bisa juga dihasilkan dari manipulasi sebuah array ataupun slice lainnya. Karena merupakan data *reference*, menjadikan perubahan data di tiap elemen slice akan berdampak pada slice lain yang memiliki alamat memori yang sama.

1. Inisialisasi Slice

Cara pembuatan slice mirip seperti pembuatan array, bedanya tidak perlu mendefinisikan jumlah elemen ketika awal deklarasi. Pengaksesan nilai elemen-nya juga sama. Contoh pembuatan slice:

```
var fruits = []string{"apple", "grape", "banana", "melon"}
fmt.Println(fruits[0]) // "apple"
```

Salah satu perbedaan slice dan array bisa diketahui pada saat deklarasi variabel-nya, jika jumlah elemen tidak dituliskan, maka variabel tersebut adalah slice.

```
var fruitsA = []string{"apple", "grape"} // slice
var fruitsB = [2]string{"banana", "melon"} // array
var fruitsC = [...]string{"papaya", "grape"} // array
```

2. Hubungan Slice Dengan Array & Operasi Slice

Kalau perbedaannya hanya di penentuan alokasi pada saat inisialisasi, kenapa tidak menggunakan satu istilah saja? atau adakah perbedaan lainnya?

Sebenarnya slice dan array tidak bisa dibedakan karena merupakan sebuah kesatuan. Array adalah kumpulan nilai atau elemen, sedang slice adalah referensi tiap elemen tersebut.

Slice bisa dibentuk dari array yang sudah didefinisikan, caranya dengan memanfaatkan teknik **2 index** untuk mengambil elemen-nya. Contoh bisa dilihat pada kode berikut.

```
var fruits = []string{"apple", "grape", "banana", "melon"}
var newFruits = fruits[0:2]

fmt.Println(newFruits) // ["apple", "grape"]
```

Kode `fruits[0:2]` maksudnya adalah pengaksesan elemen dalam slice `fruits` yang **dimulai dari indeks ke-0, hingga elemen sebelum indeks ke-2**. Elemen yang memenuhi kriteria tersebut akan didapat, untuk kemudian disimpan pada variabel lain sebagai slice baru. Pada contoh di atas, `newFruits` adalah slice baru yang tercetak dari slice `fruits`, dengan isi 2 elemen, yaitu "apple" dan "grape".

```
C:\Workspaces\Materi-Golang\Slice>_
```

Ketika mengakses elemen array menggunakan satu buah indeks (seperti `data[2]`), nilai yang didapat merupakan hasil **copy** dari referensi aslinya. Berbeda dengan pengaksesan elemen menggunakan 2 indeks (seperti `data[0:2]`), nilai yang didapat adalah *reference* elemen atau slice.

Tabel berikut adalah list operasi operasi menggunakan teknik 2 indeks yang bisa dilakukan.

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

Kode	Output	Penjelasan
<code>fruits[0:2]</code>	<code>[apple, grape]</code>	semua elemen mulai indeks ke-0, hingga sebelum indeks ke-2
<code>fruits[0:4]</code>	<code>[apple, grape, banana, melon]</code>	semua elemen mulai indeks ke-0, hingga sebelum indeks ke-4
<code>fruits[0:0]</code>	<code>[]</code>	menghasilkan slice kosong, karena tidak ada elemen sebelum indeks ke-0
<code>fruits[4:4]</code>	<code>[]</code>	menghasilkan slice kosong, karena tidak ada elemen yang dimulai dari indeks ke-4
<code>fruits[4:0]</code>	<code>[]</code>	error, pada penulisan <code>fruits[a,b]</code> nilai <code>a</code> harus lebih besar atau sama dengan <code>b</code>
<code>fruits[:]</code>	<code>[apple, grape, banana, melon]</code>	semua elemen
<code>fruits[2:]</code>	<code>[banana, melon]</code>	semua elemen mulai indeks ke-2
<code>fruits[:2]</code>	<code>[apple, grape]</code>	semua elemen hingga sebelum indeks ke-2

3. Slice Merupakan Tipe Data Reference

Slice merupakan tipe data *reference* atau referensi. Artinya jika ada slice baru yang terbentuk dari slice lama, maka data elemen slice yang baru akan memiliki alamat memori yang sama dengan elemen slice lama. Setiap perubahan yang terjadi di elemen slice baru, akan berdampak juga pada elemen slice lama yang memiliki referensi yang sama.

Program berikut merupakan pembuktian tentang teori yang baru kita bahas. Kita akan mencoba mengubah data elemen slice baru, yang terbentuk dari slice lama.

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

var aFruits = fruits[0:3]
var bFruits = fruits[1:4]

var aaFruits = aFruits[1:2]
var baFruits = bFruits[0:1]

fmt.Println(fruits)    // [apple grape banana melon]
fmt.Println(aFruits)   // [apple grape banana]
fmt.Println(bFruits)   // [grape banana melon]
fmt.Println(aaFruits)  // [grape]
fmt.Println(baFruits)  // [grape]

fmt.Println()
fmt.Println(`Buah "grape" diubah menjadi "pineapple"`)
fmt.Println()
baFruits[0] = "pineapple"

fmt.Println(fruits)    // [apple pineapple banana melon]
fmt.Println(aFruits)   // [apple pineapple banana]
fmt.Println(bFruits)   // [pineapple banana melon]
fmt.Println(aaFruits)  // [pineapple]
fmt.Println(baFruits)  // [pineapple]
```

Sekilas bisa kita lihat bahwa setelah slice yang isi datanya adalah `grape` di-ubah menjadi `pineapple`, semua slice pada 4 variabel lainnya juga ikut berubah.

Variabel `aFruits`, `bFruits` merupakan slice baru yang terbentuk dari variabel `fruits`. Dengan menggunakan dua slice baru tersebut, diciptakan lagi slice lainnya, yaitu `aaFruits`, dan `baFruits`. Kelima slice tersebut ditampilkan nilainya.

Selanjutnya, nilai dari `baFruits[0]` diubah, dan 5 slice tadi ditampilkan lagi. Hasilnya akan ada banyak slice yang elemennya ikut berubah. Yaitu elemen-elemen yang referensi-nya sama dengan referensi elemen `baFruits[0]`.

```
[apple grape banana]
[grape banana melon]
[grape]
[grape]

Buah "grape" diubah menjadi "pineapple"

[apple pineapple banana melon]
[apple pineapple banana]
[pineapple banana melon]
[pineapple]
[pineapple]

C:\Workspaces\Materi-Golang\Slice>
```

Bisa dilihat pada output di atas, elemen yang sebelumnya bernilai "grape" pada variabel `fruits`, `aFruits`, `bFruits`, `aaFruits`, dan `baFruits`; kesemuanya berubah menjadi "pineapple", karena memiliki referensi yang sama.

4. Fungsi `len()`

Fungsi `len()` digunakan untuk menghitung jumlah elemen slice yang ada. Sebagai contoh jika sebuah variabel adalah slice dengan data 4 buah, maka fungsi ini pada variabel tersebut akan mengembalikan angka **4**.

```
var fruits = []string{"apple", "grape", "banana", "melon"}
fmt.Println(len(fruits)) // 4
```

5. Fungsi `cap()`

Fungsi `cap()` digunakan untuk menghitung lebar atau kapasitas maksimum slice. Nilai kembalian fungsi ini untuk slice yang baru dibuat pasti sama dengan `len`, tapi bisa berubah seiring operasi slice yang dilakukan. Agar lebih jelas, silakan disimak kode berikut.

```
var fruits = []string{"apple", "grape", "banana", "melon"}
fmt.Println(len(fruits)) // len: 4
fmt.Println(cap(fruits)) // cap: 4

var aFruits = fruits[0:3]
fmt.Println(len(aFruits)) // len: 3
fmt.Println(cap(aFruits)) // cap: 4

var bFruits = fruits[1:4]
fmt.Println(len(bFruits)) // len: 3
fmt.Println(cap(bFruits)) // cap: 3
```

Variabel `fruits` disiapkan di awal dengan jumlah elemen 4, fungsi `len(fruits)` dan `cap(fruits)` pasti hasilnya 4.

Variabel `aFruits` dan `bFruits` merupakan slice baru berisikan 3 buah elemen milik slice `fruits`. Variabel `aFruits` mengambil elemen index 0, 1, 2; sedangkan `bFruits` 1, 2, 3.

Fungsi `len()` menghasilkan angka 3, karena jumlah elemen kedua slice ini adalah 3. Tetapi `cap(aFruits)` menghasilkan angka yang berbeda, yaitu 4 untuk `aFruits` dan 3 untuk `bFruits`. kenapa? jawabannya bisa dilihat pada tabel berikut.

Kode	Output	len ()	cap()
<code>fruits[0:4]</code>	<code>[buah buah buah buah]</code>	4	4
<code>aFruits[0:3]</code>	<code>[buah buah buah --]</code>	3	4
<code>bFruits[1:3]</code>	<code>--- - [buah buah buah]</code>	3	3

Kita analogikan slicing 2 index menggunakan **x** dan **y**.

```
fruits[x:y]
```

Slicing yang dimulai dari indeks **0** hingga **y** akan mengembalikan elemen-elemen mulai indeks **0** hingga sebelum indeks **y**, dengan lebar kapasitas adalah sama dengan slice aslinya.

6. Fungsi `append()`

Fungsi `append()` digunakan untuk menambahkan elemen pada slice. Elemen baru tersebut diposisikan setelah indeks paling akhir. Nilai balik fungsi ini adalah slice yang sudah ditambahkan nilai barunya. Contoh penggunaannya bisa dilihat di kode berikut.

```
var fruits = []string{"apple", "grape", "banana"}
var cFruits = append(fruits, "papaya")

fmt.Println(fruits) // ["apple", "grape", "banana"]
fmt.Println(cFruits) // ["apple", "grape", "banana", "papaya"]
```

Ada 3 hal yang perlu diketahui dalam penggunaan fungsi ini.

- Ketika jumlah elemen dan lebar kapasitas adalah sama (`len(fruits) == cap(fruits)`), maka elemen baru hasil `append()` merupakan referensi baru.
- Ketika jumlah elemen lebih kecil dibanding kapasitas (`len(fruits) < cap(fruits)`), elemen baru tersebut ditempatkan kedalam cakupan kapasitas, menjadikan semua elemen slice lain yang referensi-nya sama akan berubah nilainya.

Agar lebih jelas silakan perhatikan contoh berikut.

```
var fruits = []string{"apple", "grape", "banana"}
var bFruits = fruits[0:2]

fmt.Println(cap(bFruits)) // 3
fmt.Println(len(bFruits)) // 2

fmt.Println(fruits) // ["apple", "grape", "banana"]
fmt.Println(bFruits) // ["apple", "grape"]

var cFruits = append(bFruits, "papaya")

fmt.Println(fruits) // ["apple", "grape", "papaya"]
fmt.Println(bFruits) // ["apple", "grape"]
fmt.Println(cFruits) // ["apple", "grape", "papaya"]
```

Pada contoh di atas bisa dilihat, elemen indeks ke-2 slice `fruits` nilainya berubah setelah ada penggunaan keyword `append()` pada `bFruits`. Slice `bFruits` kapasitasnya adalah **3** sedang datanya hanya **2**. Karena `len(bFruits) < cap(bFruits)`, maka elemen baru yang dihasilkan, terdeteksi sebagai perubahan nilai pada referensi yang lama (referensi elemen indeks ke-2 slice `fruits`), membuat elemen yang referensinya sama, nilainya berubah.

7. Fungsi `copy()`

Fungsi `copy()` digunakan untuk men-copy elements slice pada `src` (parameter ke-2), ke `dst` (parameter pertama).

```
copy(dst, src)
```

Jumlah element yang di-copy dari `src` adalah sejumlah lebar slice `dst` (atau `len(dst)`). Jika jumlah slice pada `src` lebih kecil dari `dst`, maka akan ter-copy semua. Lebih jelasnya silakan perhatikan contoh berikut.

```
dst := make([]string, 3)
src := []string{"watermelon", "pinapple", "apple", "orange"}
n := copy(dst, src)

fmt.Println(dst) // watermelon pinapple apple
fmt.Println(src) // watermelon pinapple apple orange
fmt.Println(n) // 3
```

Pada kode di atas variabel slice `dst` dipersiapkan dengan lebar adalah 3 elements. Slice `src` yang isinya 4 elements, di-copy ke `dst`. Menjadikan isi slice `dst` sekarang adalah 3 buah elements yang sama dengan 3 buah elements `src`, hasil dari operasi `copy()`.

Yang ter-copy hanya 3 buah (meski `src` memiliki 4 elements) hal ini karena `copy()` hanya meng-copy elements sebanyak `len(dst)`.

```
Fungsi copy() mengembalikan informasi angka, representasi dari jumlah element yang berhasil di-copy.
```

Pada contoh kedua berikut, `dst` merupakan slice yang sudah ada isinya, 3 buah elements. Variabel `src` yang juga merupakan slice dengan isi dua elements, di-copy ke `dst`. Karena operasi `copy()` akan meng-copy sejumlah `len(dst)`, maka semua elements `src` akan ter-copy **karena jumlahnya dibawah atau sama dengan lebar dst**.

```
dst := []string{"potato", "potato", "potato"}
src := []string{"watermelon", "pinapple"}
n := copy(dst, src)

fmt.Println(dst) // watermelon pinapple potato
fmt.Println(src) // watermelon pinapple
fmt.Println(n) // 2
```


8. Pengaksesan Elemen Slice Dengan 3 Indeks

3 index adalah teknik slicing elemen yang sekaligus menentukan kapasitasnya. Cara menggunakannya yaitu dengan menyisipkan angka kapasitas di belakang, seperti `fruits[0:1:1]`. Angka kapasitas yang diisikan tidak boleh melebihi kapasitas slice yang akan di slicing.

Berikut merupakan contoh penerapannya.

```
var fruits = []string{"apple", "grape", "banana"}
var aFruits = fruits[0:2]
var bFruits = fruits[0:2:2]

fmt.Println(fruits)      // ["apple", "grape", "banana"]
fmt.Println(len(fruits)) // len: 3
fmt.Println(cap(fruits)) // cap: 3

fmt.Println(aFruits)     // ["apple", "grape"]
fmt.Println(len(aFruits)) // len: 2
fmt.Println(cap(aFruits)) // cap: 3

fmt.Println(bFruits)     // ["apple", "grape"]
fmt.Println(len(bFruits)) // len: 2
fmt.Println(cap(bFruits)) // cap: 2
```

Map

Map adalah tipe data asosiatif yang ada di Go, berbentuk *key-value pair*. Untuk setiap data (atau value) yang disimpan, disiapkan juga key-nya. Key harus unik, karena digunakan sebagai penanda (atau identifier) untuk pengaksesan value yang bersangkutan.

Kalau dilihat, `map` mirip seperti slice, hanya saja indeks yang digunakan untuk pengaksesan bisa ditentukan sendiri tipe-nya (indeks tersebut adalah key).

1. Penggunaan Map

Cara menggunakan map cukup dengan menuliskan keyword `map` diikuti tipe data key dan value-nya. Agar lebih mudah dipahami, silakan perhatikan contoh di bawah ini.

```
var chicken map[string]int
chicken = map[string]int{}

chicken["januari"] = 50
chicken["februari"] = 40

fmt.Println("januari", chicken["januari"]) // januari 50
fmt.Println("mei",      chicken["mei"])    // mei 0
```

Variabel `chicken` dideklarasikan sebagai map, dengan tipe data key adalah `string` dan value-nya `int`. Dari kode tersebut bisa dilihat bagaimana cara penggunaan keyword `map`.

Kode `map[string]int` maknanya adalah, tipe data `map` dengan key bertipe `string` dan value bertipe `int`.

Default nilai variabel `map` adalah `nil`. Oleh karena itu perlu dilakukan inisialisasi nilai default di awal, caranya cukup dengan tambahkan kurung kurawal pada akhir tipe, contoh seperti pada kode di atas: `map[string]int{}`.

Cara menge-set nilai pada sebuah map adalah dengan menuliskan variabel-nya, kemudian disisipkan `key` pada kurung siku variabel (mirip seperti cara pengaksesan elemen slice), lalu isi nilainya. Contohnya seperti `chicken["februari"] = 40`. Sedangkan cara pengambilan value adalah cukup dengan menyisipkan `key` pada kurung siku variabel.

Pengisian data pada map bersifat **overwrite**, ketika variabel sudah memiliki item dengan key yang sama, maka value lama akan ditimpa dengan value baru.

Pada pengaksesan item menggunakan key yang belum tersimpan di map, akan dikembalikan nilai default tipe data value-nya. Contohnya seperti pada kode di atas, `chicken["mei"]` menghasilkan nilai 0 (nilai default tipe `int`), karena belum ada item yang tersimpan menggunakan key `"mei"`.

2. Inisialisasi Nilai Map

Zero value dari map adalah `nil`, maka tiap variabel bertipe map harus di-inisialisasi secara eksplisit nilai awalnya (agar tidak `nil`).

```
data = map[string]int{}
data["one"] = 1
// tidak ada error
```

Nilai variabel bertipe map bisa didefinisikan di awal, caranya dengan menambahkan kurung kurawal setelah tipe data, lalu menuliskan key dan value didalamnya. Cara ini sekilas mirip dengan definisi nilai array/slice namun dalam bentuk key-value.

```
// cara horizontal
var chicken1 = map[string]int{"januari": 50, "februari": 40}

// cara vertical
var chicken2 = map[string]int{
    "januari": 50,
    "februari": 40,
}
```

Key dan value dituliskan dengan pembatas tanda titik dua (:). Sedangkan tiap itemnya dituliskan dengan pembatas tanda koma (,). Khusus deklarasi dengan gaya vertikal, tanda koma perlu dituliskan setelah item terakhir.

Variabel **map** bisa di-inisialisasi dengan tanpa nilai awal, caranya menggunakan tanda kurung kurawal, contoh: **map[string]int{}**. Atau bisa juga dengan menggunakan keyword **make** dan **new**. Contohnya bisa dilihat pada kode berikut. Ketiga cara di bawah ini intinya adalah sama.

```
var chicken3 = map[string]int{}
var chicken4 = make(map[string]int)
var chicken5 = *new(map[string]int)
```

Khusus inisialisasi data menggunakan keyword **new**, yang dihasilkan adalah data pointer. Untuk mengambil nilai aslinya bisa dengan menggunakan tanda asterisk (*). Topik pointer akan dibahas lebih detail di materi selanjutnya

4. Menghapus Item Map

Fungsi **delete()** digunakan untuk menghapus item dengan key tertentu pada variabel map. Cara penggunaannya, dengan memasukan objek map dan key item yang ingin dihapus sebagai parameter.

```
var chicken = map[string]int{"januari": 50, "februari": 40}

fmt.Println(len(chicken)) // 2
fmt.Println(chicken)

delete(chicken, "januari")

fmt.Println(len(chicken)) // 1
fmt.Println(chicken)
```

Item yang memiliki key **"januari"** dalam variabel **chicken** akan dihapus.

Fungsi **len()** jika digunakan pada map akan mengembalikan jumlah item.

5. Deteksi Keberadaan Item Dengan Key Tertentu

Ada cara untuk mengetahui apakah dalam sebuah variabel map terdapat item dengan key tertentu atau tidak, yaitu dengan memanfaatkan 2 variabel sebagai penampung nilai kembalian pengaksesan item. Return value ke-2 ini adalah opsional, isinya nilai **bool** yang menunjukkan ada atau tidaknya item yang dicari.

```
var chicken = map[string]int{"januari": 50, "februari": 40}
var value, isExist = chicken["mei"]

if isExist {
    fmt.Println(value)
} else {
    fmt.Println("item is not exists")
}
```

Looping

Looping adalah proses mengulang-ulang eksekusi blok kode tanpa henti, selama kondisi yang dijadikan acuan terpenuhi. Biasanya disiapkan variabel untuk iterasi atau variabel penanda kapan perulangan akan diberhentikan.

Di Go keyword perulangan hanya **for** saja, tetapi meski demikian, kemampuannya merupakan gabungan **for**, **foreach**, dan **while** ibarat bahasa pemrograman lain.

1. Perulangan Menggunakan Keyword **for**

```
for i := 0; i < 5; i++ {
    fmt.Println("Angka", i)
}
```

Perulangan di atas hanya akan berjalan ketika variabel `i` bernilai dibawah `5`, dengan ketentuan setiap kali perulangan, nilai variabel `i` akan di-iterasi atau ditambahkan 1 (`i++` artinya ditambah satu, sama seperti `i = i + 1`). Karena `i` pada awalnya bernilai 0, maka perulangan akan berlangsung 5 kali, yaitu ketika `i` bernilai 0, 1, 2, 3, dan 4.

2. Penggunaan Keyword `for` Dengan Argumen Hanya Kondisi

Cara ke-2 adalah dengan menuliskan kondisi setelah keyword `for` (hanya kondisi). Deklarasi dan iterasi variabel counter tidak dituliskan setelah keyword, hanya kondisi perulangan saja. Konsepnya mirip seperti `while` milik bahasa pemrograman lain.

Kode berikut adalah contoh `for` dengan argumen hanya kondisi (seperti `if`), output yang dihasilkan sama seperti penerapan for cara pertama.

```
var i = 0

for i < 5 {
    fmt.Println("Angka", i)
    i++
}
```

3. Penggunaan Keyword `for` Tanpa Argumen

Cara ke-3 adalah `for` ditulis tanpa kondisi. Dengan ini akan dihasilkan perulangan tanpa henti (sama dengan `for true`). Pemberhentian perulangan dilakukan dengan menggunakan keyword `break`.

```
var i = 0

for {
    fmt.Println("Angka", i)

    i++
    if i == 5 {
        break
    }
}
```

Dalam perulangan tanpa henti di atas, variabel `i` yang nilai awalnya `0` di-inkrementasi. Ketika nilai `i` sudah mencapai `5`, keyword `break` digunakan, dan perulangan akan berhenti.

4. Perulangan Elemen Array Menggunakan Keyword `for - range`

Ada cara yang lebih sederhana me-looping data array, dengan menggunakan keyword `for - range`. Contoh pengaplikasiannya bisa dilihat di kode berikut.

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

for i, fruit := range fruits {
    fmt.Printf("elemen %d : %s\n", i, fruit)
}
```

Array `fruits` diambil elemen-nya secara berurutan. Nilai tiap elemen ditampung variabel oleh `fruit` (tanpa huruf s), sedangkan indeks nya ditampung variabel `i`.

Output program di atas, sama dengan output program sebelumnya, hanya cara yang digunakan berbeda.

5. Penggunaan Variabel Underscore `_` Dalam `for - range`

Kadang kala ketika *looping* menggunakan `for - range`, ada kemungkinan dimana data yang dibutuhkan adalah elemen-nya saja, indeks-nya tidak. Sedangkan kode di atas, `range` mengembalikan 2 data, yaitu indeks dan elemen.

Seperti yang sudah diketahui, bahwa di Go tidak memperbolehkan adanya variabel yang menganggur atau tidak dipakai. Jika dipaksakan, error akan muncul, contohnya seperti kode berikut.

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

for i, fruit := range fruits {
    fmt.Printf("nama buah : %s\n", fruit)
}
```

Hasil dari kode program di atas:

Disinilah salah satu kegunaan variabel pengangguran, atau underscore (`_`). Tampung saja nilai yang tidak ingin digunakan ke underscore.


```
fmt.Printf("nama buah : %s\n", fruit)
}
```

Pada kode di atas, yang sebelumnya adalah variabel `i` diganti dengan `_`, karena kebetulan variabel `i` tidak digunakan.

Jika yang dibutuhkan hanya indeks elemen-nya saja, bisa gunakan 1 buah variabel setelah keyword `for`.

```
for i, _ := range fruits { }
// atau
for i := range fruits { }
```

6. Penggunaan Keyword `break` & `continue`

Keyword `break` digunakan untuk menghentikan secara paksa sebuah perulangan, sedangkan `continue` dipakai untuk memaksa maju ke perulangan berikutnya.

Berikut merupakan contoh penerapan `continue` dan `break`. Kedua keyword tersebut dimanfaatkan untuk menampilkan angka genap berurutan yang lebih besar dari 0 dan dibawah 8.

```
for i := 1; i <= 10; i++ {
    if i % 2 == 1 {
        continue
    }

    if i > 8 {
        break
    }

    fmt.Println("Angka", i)
}
```

Kode di atas akan lebih mudah dicerna jika dijelaskan secara berurutan. Berikut adalah penjelasannya.

1. Dilakukan perulangan mulai angka 1 hingga 10 dengan `i` sebagai variabel iterasi.
2. Ketika `i` adalah ganjil (dapat diketahui dari `i % 2`, jika hasilnya `1`, berarti ganjil), maka akan dipaksa lanjut ke perulangan berikutnya.
3. Ketika `i` lebih besar dari 8, maka perulangan akan berhenti.
4. Nilai `i` ditampilkan.

Referensi Video:

- [Array](#) (Programmer Zaman Now)
- [Slice](#) (Programmer Zaman Now)
- [Map](#) (Programmer Zaman Now)
- [Loop](#) (Programmer Zaman Now)
- [Break & Continue](#) (Programmer Zaman Now)

Referensi Tulisan:

- <https://dasarpemrogramangolang.novalagung.com/A-array.html>
- <https://dasarpemrogramangolang.novalagung.com/A-slice.html>
- <https://dasarpemrogramangolang.novalagung.com/A-map.html>
- <https://dasarpemrogramangolang.novalagung.com/A-perulangan.html>

Rating - Feedback

Berikan Rating pada posting ini:



Berikan kritik dan saran..

Submit

