MATERI

Hari 5 - Function

Function

Sebelumnya kita sudah mengenal sebuah function yang wajib dibuat agar program kita bisa berjalan, yaitu function main. lalu apa itu function, function adalah sebuah blok kode yang sengaja dibuat dalam program agar bisa digunakan berulang-ulang. Cara membuat function sangat sederhana, hanya dengan menggunakan kata kunci `func`lalu diikuti dengan nama function nya dan blok kode isi function nya Setelah membuat function, kita bisa mengeksekusi function tersebut dengan memanggilnya menggunakan kata kunci nama function nya diikuti tanda kurung buka, kurung tutup

berikut ini contoh penggunaan function:

```
package main
```

```
import "fmt"

func printHello() {
  fmt.Println("Hello")
}

func main() {
  printHello()
}
```

1. Function Parameter

di dalam function terkadang kita memerlukan data dari luar function yang disebut parameter. Kita bisa menambahkan parameter di function, bisa lebih dari satu Parameter tidaklah wajib, jadi kita bisa membuat function tanpa parameter seperti sebelumnya yang sudah kita buat Namun jika kita menambahkan parameter di function, maka ketika memanggil function tersebut, kita wajib memasukkan data ke parameternya

berikut ini contoh penggunaan parameter pada function:

```
package main

import "fmt"

func printAngka(angka1 int, angka2 int) {
  fmt.Println("angka pertama", angka1)
  fmt.Println("angka kedua", angka2)
}

func main() {
  printAngka(1, 2)
}
```

2. Function Return Value

Function bisa mengembalikan data. Untuk memberitahu bahwa function mengembalikan data, kita harus menuliskan tipe data kembali dari function tersebut Jika function tersebut kita deklarasikan dengan tipe data pengembalian, maka wajib di dalam function nya kita harus mengembalikan data. Untuk mengembalikan data dari function, kita bisa menggunakan kata kunci return, diikuti dengan datanya

berikut ini contoh penggunaan return value pada function:

```
func introduction(name string) string {
  return "Hello My Name Is " + name
}

func main() {
  //langsung panggil didalam print
  fmt.Println(introduction("John"))


  // menggunakan variabel
  result := introduction("Doe")
  fmt.Println(result)

  // function as value
  secondResult := introduction
  fmt.Println(secondResult("Jack"))
}
```

3. Function Return Multiple Value

Function tidak hanya dapat mengembalikan satu value, tapi juga bisa multiple value Untuk memberitahu jika function mengembalikan multiple value, kita harus menulis semua tipe data return value nya di function.

Multiple return value wajib ditangkap semua value nya Jika kita ingin menghiraukan return value tersebut, kita bisa menggunakan tanda _ (garis bawah)

berikut ini contoh penggunaan return multiple value pada function:

```
package main

import "fmt"

func introduction(firstName string, lastName
string) (string, string) {
  introFirstName := "Hello My First Name Is " +
firstName
  introLastName := "Hello My Last Name Is " +
lastName
  return introFirstName, introLastName
}

func main() {
  firstName, lastName := introduction("John",
"Doe")
  fmt.Println(firstName, lastName)

  firstName2, _ := introduction("John", "Doe")
  fmt.Println(firstName2)
}
```

4. Function Predefined return value

Biasanya saat kita memberi tahu bahwa sebuah function mengembalikan value, maka kita hanya mendeklarasikan tipe data return value di function. Namun kita juga bisa membuat variable secara langsung di tipe data return function nya

berikut ini contoh penggunaan return multiple value pada function:

```
// contoh 1
func tambahAngka(firstNumber int, lastNumber int)
(jumlah int) {
  jumlah = firstNumber+lastNumber
  return
}

// contoh 2
func tampilkanKataDanAngka() (firstWord,
secondWord string, number int) {
  firstWord = "Halo"
  secondWord = "Dunia"
  number = 10
  return
}

func main() {
  hasil := tambahAngka(4,5)
  fmt.Println(hasil)

  kataPertama, kataKedua, angka :=
tampilkanKataDanAngka()
  fmt.Println(kataPertama, kataKedua, angka)
}
```

jika di perhatikan pada function tambahAngka di dalamnya terdapat kode `jumlah = firstNumber+lastNumber`, variabel jumlah tidak menggunakan `var`sebelumnya maupun `:=` di karenakan nama jumlah sudah didefinisikan terlebih dahulu pada bersama tipe data yang akan di kembalikan

4. Variadic Function

Go mengadopsi konsep **variadic function** atau pembuatan fungsi dengan parameter sejenis yang tak terbatas. Maksud **tak terbatas** disini adalah jumlah parameter yang disisipkan ketika pemanggilan fungsi bisa berapa saja.

Parameter variadic memiliki sifat yang mirip dengan slice. Nilai dari parameter-parameter yang disisipkan bertipe data sama, dan ditampung oleh sebuah variabel saja. Cara pengaksesan tiap datanya juga sama, dengan menggunakan index.

Deklarasi parameter variadic sama dengan cara deklarasi variabel biasa, pembedanya adalah pada parameter jenis ini ditambahkan tanda titik tiga kali (`...`) tepat setelah penulisan variabel (sebelum tipe data). Nantinya semua nilai yang disisipkan sebagai parameter akan ditampung oleh variabel tersebut.

Berikut merupakan contoh penggunaanya:

```
package main

import "fmt"

func sum(numbers ...int) int {
  var total int = 0
  for _, number := range numbers {
    total += number
  }
  return total
}

func main() {
  var total = sum(2, 4, 3, 5, 4, 3, 3, 5, 5, 3)
  fmt.Println(total)
}
```

Pengisian Parameter Fungsi Variadic Menggunakan Data Slice

Kadang ada kasus dimana kita menggunakan Variadic Function, namun memiliki variable berupa slice Kita bisa menjadikan **slice** sebagai **parameter**. cara menggunakannya cukup gunakan variabel slice sebagai parameter dan tambahkan tanda titik tiga kali (`...`) setelah nama variabel tersebut.

```
func sum(numbers ...int) int {
  var total int = 0
  for _, number := range numbers {
    total += number
  }
  return total
}

func main() {
  var numbers = []int{2,6,7,8,9,10}
  var total = sum(numbers...)
  fmt.Println(total)
}
```

Function Dengan Parameter Biasa & Variadic

Parameter variadic bisa dikombinasikan dengan parameter biasa, dengan syarat parameter variadic-nya harus diposisikan di akhir. Contohnya bisa dilihat pada kode berikut.

```
package main

import "fmt"

func yourHobbies(name string, hobbies ...string)
{
  fmt.Println("Hello, my name is", name)
  fmt.Println("My hobbies are: ")
  for _, hobby := range hobbies {
    fmt.Println(hobby)
  }
}

func main() {
  yourHobbies("John", "Gaming", "Hiking",
"Reading")
  fmt.Println()

  var hobbies = []string{"Sleeping", "Eating"}
  yourHobbies("Doe", hobbies...)
}
```

5. Closure Function

Definisi **Closure** adalah sebuah fungsi yang bisa disimpan dalam variabel. Dengan menerapkan konsep tersebut, kita bisa membuat fungsi didalam fungsi, atau bahkan membuat fungsi yang mengembalikan fungsi.

Closure merupakan *anonymous function* atau fungsi tanpa nama. Biasa dimanfaatkan untuk membungkus suatu proses yang hanya dipakai sekali atau dipakai pada blok tertentu saja.

Closure Disimpan Sebagai Variabel

Sebuah fungsi tanpa nama bisa disimpan dalam variabel. Variabel yang menyimpan closure memiliki sifat seperti fungsi yang disimpannya. Di bawah ini adalah contoh program sederhana untuk mencari nilai terendah dan tertinggi dari suatu array. Logika pencarian dibungkus dalam closure yang ditampung oleh variabel `getMinMax`.

```
import "fmt"

func main() {
  var getMinMax = func(n []int) (int, int) {
    var min, max int
    for i, e := range n {
      switch {
        case i == 0:
          max, min = e, e
        case e > max:
          max = e
        case e < min:
          min = e
      }
    }
    return min, max
  }
  var numbers = []int{2, 3, 4, 3, 4, 2, 3}
  var min, max = getMinMax(numbers)
  fmt.Println("data :", numbers)
  fmt.Println("min :", min)
  fmt.Println("min :", max)
}
```

Bisa dilihat pada kode di atas bagiamana sebuah closure dibuat dan dipanggil. Sedikit berbeda memang dibanding pembuatan fungsi biasa. Fungsi ditulis tanpa nama, lalu ditampung dalam variabel.

```
var getMinMax = func(n
[]int) (int, int) {
    // ...
}
```

Cara pemanggilannya, dengan menuliskan nama variabel tersebut sebagai fungsi, seperti pemanggilan fungsi biasa.

```
var min, max =
getMinMax(numbers)
```

Output program:

```
C:\Workspaces\Materi-Golang\Function>go run main.go
data : [2 3 4 3 4 2 3]
min : 2
min : 4

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

Closure Sebagai Nilai Kembali

Salah satu keunikan closure lainnya adalah bisa dijadikan sebagai nilai balik fungsi, cukup aneh memang, tapi pada suatu kondisi teknik ini sangat membantu. Di bawah ini disediakan sebuah fungsi bernama `findMax()`, fungsi ini salah satu nilai kembalinya berupa closure.

```
package main

import "fmt"

func findMax(numbers []int, max int) (int, func()
[]int) {
  var res []int
  for _, e := range numbers {
    if e <= max {
      res = append(res, e)
    }
  }
  return len(res), func() []int {
    return res
  }
}
```

Nilai kembali ke-2 pada fungsi di atas adalah closure dengan skema `func() []int`. Bisa dilihat di bagian akhir, ada fungsi tanpa nama yang dikembalikan.

```
}
```

Sedikit tentang fungsi `findMax()`, fungsi ini digunakan untuk mencari banyaknya angka-angka yang nilainya di bawah atau sama dengan angka tertentu. Nilai kembalian pertama adalah jumlah angkanya. Nilai kembalian kedua berupa closure yang mengembalikan angka-angka yang dicari. Berikut merupakan contoh implementasi fungsi tersebut.

```
func main() {
  var max = 3
  var numbers = []int{2, 3, 0, 4, 3, 2, 0, 4, 2,
0, 3}
  var howMany, getNumbers = findMax(numbers, max)
  var theNumbers = getNumbers()
  fmt.Println("numbers\t:", numbers)
  fmt.Printf("find \t: %d\n\n", max)

  fmt.Println("found \t:", howMany) // 9
  fmt.Println("value \t:", theNumbers) // [2 3 0
3 2 0 2 0 3]
}
```

Output program:

```
C:\Workspaces\Materi-Golang\Function>go run main.go
numbers : [2 3 0 4 3 2 0 4 2 0 3]
find    : 3

found   : 9
value   : [2 3 0 3 2 0 2 0 3]

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

6. Function as Parameter

Function tidak hanya bisa kita simpan di dalam variable sebagai value Namun juga bisa kita gunakan sebagai parameter untuk function lain. Berikut ini contoh penggunaannya:

```
package main

import "fmt"

// function yang menggunakan function sebagai
parameternya
func sayHelloWithFilter(name string, filter
func(string) string) {
    nameFiltered := filter(name)
    fmt.Println("Hello", nameFiltered)
}

// function yang digunakan sebagai parameter
func spamFilter(name string) string {
    if name == "Kasar" {
        return "..."
    }else {
        return name
    }
}

func main() {
    sayHelloWithFilter("John", spamFilter)
    sayHelloWithFilter("Kasar", spamFilter)
}
```

Referensi Video:

- Function (Programmer Zaman Now)
- Function Parameter (Programmer Zaman Now)
- Function Returm Value (Programmer Zaman Now)
- Returning Multiple Value (Programmer Zaman Now)
- Named Return Value (Programmer Zaman Now)
- Variadic Function (Programmer Zaman Now)
- Function Value (Programmer Zaman Now)
- Function as Parameter (Programmer Zaman Now)
- Anonymous Function (Programmer Zaman Now)
- Closure (Programmer Zaman Now)

Referensi Tulisan:

- https://dasarpemrogramangolang.novalagung.com/A-fungsi.html
- https://dasarpemrogramangolang.novalagung.com/A-fungsi-multiple-return.html
- https://dasarpemrogramangolang.novalagung.com/A-fungsi-variadic.html
- https://dasarpemrogramangolang.novalagung.com/A-fungsi-closure.html
- https://dasarpemrogramangolang.novalagung.com/A-fungsi-sebagai-parameter.html

Rating - Feedback

Berikan Rating pada posting ini:

0

Berikan kritik dan saran..

Submit