Strings In Depth

Introduction

Kali ini kita akan membahas lebih jauh tentang tipe data **string** pada Go.

Tipe data **string** pada Go terbentuk dari kumpulan tipe data byte yang di letakkan di dalam slice atau bisa kita sebut dengan **slice of bytes**.

Tipe data **byte** pada Go merupakan tipe data alias dari tipe data **uint8**.

Ketika kita melakukan indexing terhadap suatu string, maka kita akan mendapat nilai representasi dari byte nya.

Looping Over String (byte-by-byte)

Ketika kita melakukan indexing terhadap suatu string, maka kita akan mendapat nilai representasi dari byte nya.

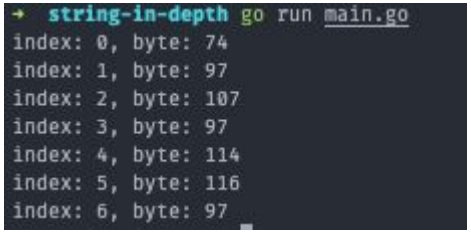
Perhatikan contoh seperti pada gambar dibawah.

```go
package main

import "fmt"

func main() {
    city := "Jakarta"

    for i := 0; i < len(city); i++ {
        fmt.Printf("index: %d, byte: %d\n", i, city[i])
    }
}
```

Pada looping tersebut, kita melakukannya dengan pendekatan byte-per-byte, atau dapat diartikan kita melooping string “Jakarta” berdasarkan byte nya. Kode city[i] tidak akan menghasilkan karakternya, melainkan byte nya.

Looping Over String (byte-by-byte)

Ketika kita eksekusi dengan perintah **go run main.go**, maka hasilnya akan seperti pada gambar dibawah ini



Bisa dilihat, huruf **J** berubah menjadi **74**, huruf **a** menjadi **97**, dan seterusnya. Angka-angka tersebut merupakan representasi dari angka desimal ASCII Code. Kalian dapat mengunjungi link dibawah ini untuk mengetahui lebih detail tentang ASCII Code.

https://www.asciitable.com/

Looping Over String (byte-by-byte)

Setelah mengetahui hasilnya pada terminal, maka kita dapat juga membuat string "Jakarta" dengan menggunakan slice of byte, karena sudah kita bahas diawal bahwa tipe data string pada Go merupakan kumpulan byte yang berada di dalam slice/slice of bytes.

Perhatikan gambar dibawah ini.

```
1  package main
2
3  import "fmt"
4
5  func main() {
6      var city []byte = []byte{74, 97, 107, 97, 114, 116, 97}
7
8      fmt.Println(string(city))
9  }
```

Variable city menampung slice of byte yang berisikan angka-angka decimal yang membentuk string "Jakarta".

Kemudian agar slice of byte tersebut dapat mencetak nilai string nya, maka kita perlu meletekannya dalam function string() seperti yang terdapat pada baris ke 8.

Looping Over String (byte-by-byte)

Jika di eksekusi dengan perintah **go run main.go**, maka hasilnya akan seperti pada gambar dibawah.

Looping Over String (rune-by-rune)

Ketika kita ingin mendapatkan panjang karakter dengan menggunakan function **len()**, maka sebetulnya kita tidak sedang mendapatkan panjang dari string berdasarkan karakternya, namun kita mendapatkan jumlah byte pada **string**.

Perhatikan contoh dibawah ini.

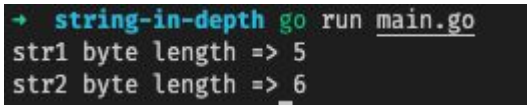
```go
package main

import "fmt"

func main() {
    str1 := "makan"

    str2 := "mânca"

    fmt.Printf("str1 byte length ⇒ %d\n", len(str1))
    fmt.Printf("str2 byte length ⇒ %d\n", len(str2))
}
```

Variable **str1** mengandung string "**makan**", sedangkan variable **str2** mengandung string "**mânca**" yang merupakan "makan" dalam bahasa Romania.

Jika kita perhatikan, string "**makan**", dan string "**mânca**" memiliki 5 karakter, dan ketika kita mencari jumlah karakter dari kedua string tersebut, maka seharusnya hasilnya adalah **5**.

Looping Over String (rune-by-rune)

Jika kita eksekusi dengan perintah **go run main.go**, maka hasilnya seperti pada gambar dibawah ini.



String “**mânca**” sebetulnya memiliki 5 karakter, namun angka 6 pada gambar diatas bukan bermaksud untuk menunjukkan panjan karakter dari string “**mânca**”,namun bermaksud untuk menunjukkan jumlah byte nya.

Jika kita perhatikan pada string “**mânca**”, terdapat sebuah karakter yang tidak didukung pada table ASCII Code, yaitu pada karakter “**â**” yang merupakan sebuah *accented-character*.

Karakter “**â**” memiliki 2 byte, inilah yang menyebabkan keluarnya angka **6** pada terminal ketika kita mencoba untuk menampilkan jumlah byte pada string “**mânca**”.

Looping Over String (rune-by-rune)

Ketika kita hendak mencari jumlah karakter nya, dan bukan jumlah bytenya, maka kita perlu merubah string tersebut menjadi rune terlebih dahulu.

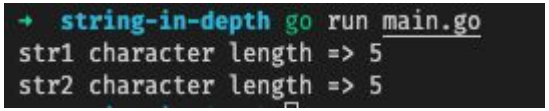
Tipe data rune merupakan tipe data alias dari int32.

Kita dapat menggunakan function RuneCountInString dari package utf8 untuk merubah string menjadi rune sekaligus mencari panjang karakternya.

```go
package main

import (
    "fmt"
    "unicode/utf8"
)

func main() {
    str1 := "makan"

    str2 := "mânca"

    fmt.Printf("str1 character length ⇒ %d\n", utf8.RuneCountInString(str1))
    fmt.Printf("str2 character length ⇒ %d\n", utf8.RuneCountInString(str2))
}
```

Looping Over String (rune-by-rune)

Ketika di eksekusi dengan perintah **go run main.go**, maka hasilnya akan seperti pada gambar dibawah ini.



Looping Over String (rune-by-rune)

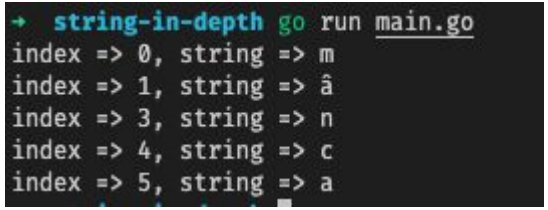
Kita juga dapat melooping sebuah string dengan secara rune-per-rune dengan menggunakan **range loop**.

Dengan menggunakan **range loop**, maka kita secara otomatis melooping string secara rune-per-rune.

```go
package main

import "fmt"

func main() {
    str := "mânca"

    for i, s := range str {
        fmt.Printf("index ⇒ %d, string ⇒ %s\n", i, string(s))
    }
}
```

Looping Over String (rune-by-rune)

Ketika kita eksekusi dengan perintah **go run main.go**, maka hasilnya akan seperti pada gambar dibawah ini.



Perhatikan urutan indexnya, ketika index **ke-1** telah ditampilkan, ia seolah-olah langsung lompat kepada index ke-3. Hal ini disebabkan karakter “â” terdiri dari 2 byte, sehinga index ke-1 dan ke-2 telah diambil alih oleh karakter “â”.

Lompatan index tersebut telah diatur oleh **range loop**, karena **range loop** melooping sebuah string secara rune-per-rune.