

---



# Go-Lang Dasar

Eko Kurniawan Khannedy

---

# License

- Dokumen ini boleh Anda gunakan atau ubah untuk keperluan non komersial
- Tapi Anda wajib mencantumkan sumber dan pemilik dokumen ini
- Untuk keperluan komersial, silahkan hubungi pemilik dokumen ini

# Eko Kurniawan Khannedy

- Technical architect at one of the biggest ecommerce company in Indonesia
- 10+ years experiences
- [youtube.com/c/ProgrammerZamanNow](https://youtube.com/c/ProgrammerZamanNow)



---

# Pengenalan Go-Lang

---

# Sejarah Go-Lang

- Dibuat di Google menggunakan bahasa pemrograman C
- Di Rilis ke public sebagai open source pada tahun 2009
- Go-Lang populer sejak digunakan untuk membuat Docker pada tahun 2011
- Saat ini mulai banyak teknologi baru yang dibuat menggunakan bahasa Go-Lang dibanding bahasa C, seperti Kubernetes, Prometheus, CockroachDB, dan lain-lain
- Saat ini mulai populer untuk pembuatan Backend API di Microservices

---

# Kenapa Belajar Go-Lang?

- Bahasa Go-Lang sangat sederhana, tidak butuh waktu lama untuk mempelajarinya
- Go-Lang mendukung baik concurrency programming, dimana saat ini kita hidup di zaman multicore processor
- Go-Lang mendukung garbage collector, sehingga tidak butuh melakukan management memory secara manual seperti di bahasa C
- Salah satu bahasa pemrograman yang sedang naik daun

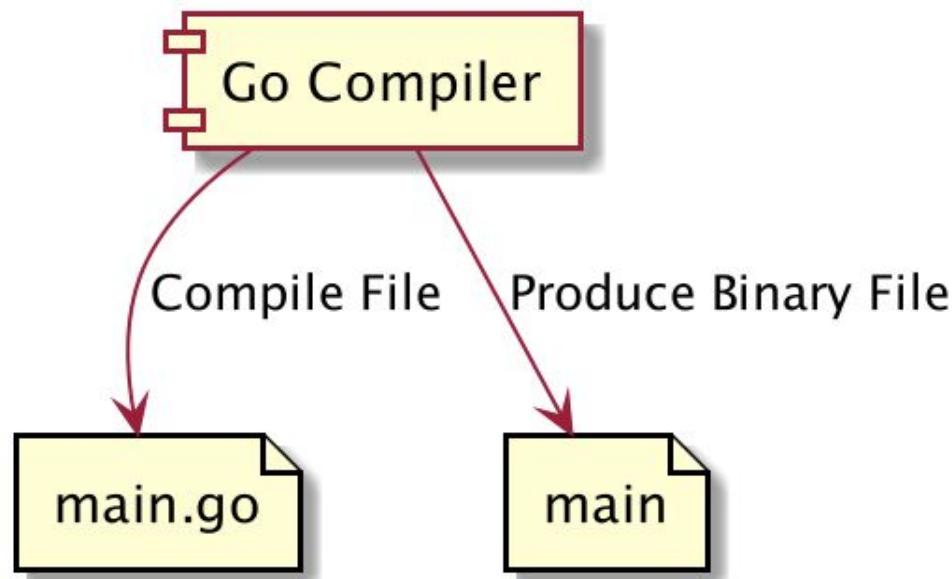
---

# Software Development Kit

- <https://golang.org/>
- Download Compiler Go-Lang
- Install Compiler Go-Lang
- Cek menggunakan perintah : go version

---

# Proses Development Program Go-Lang



---

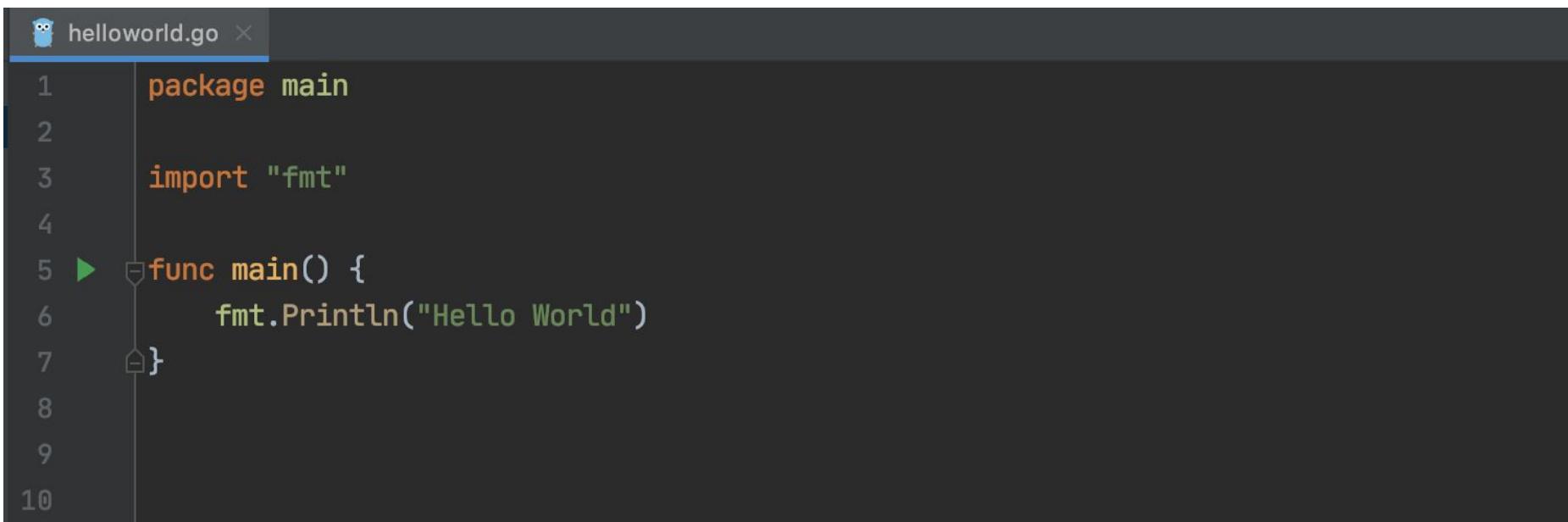
## **Text Editor atau IDE**

- Visual Studio Code
- JetBrains GoLand

---

# Program Hello World

# Program Hello World



The image shows a screenshot of a code editor with a dark theme. The title bar of the window is labeled "helloworld.go". The code editor displays the following Go program:

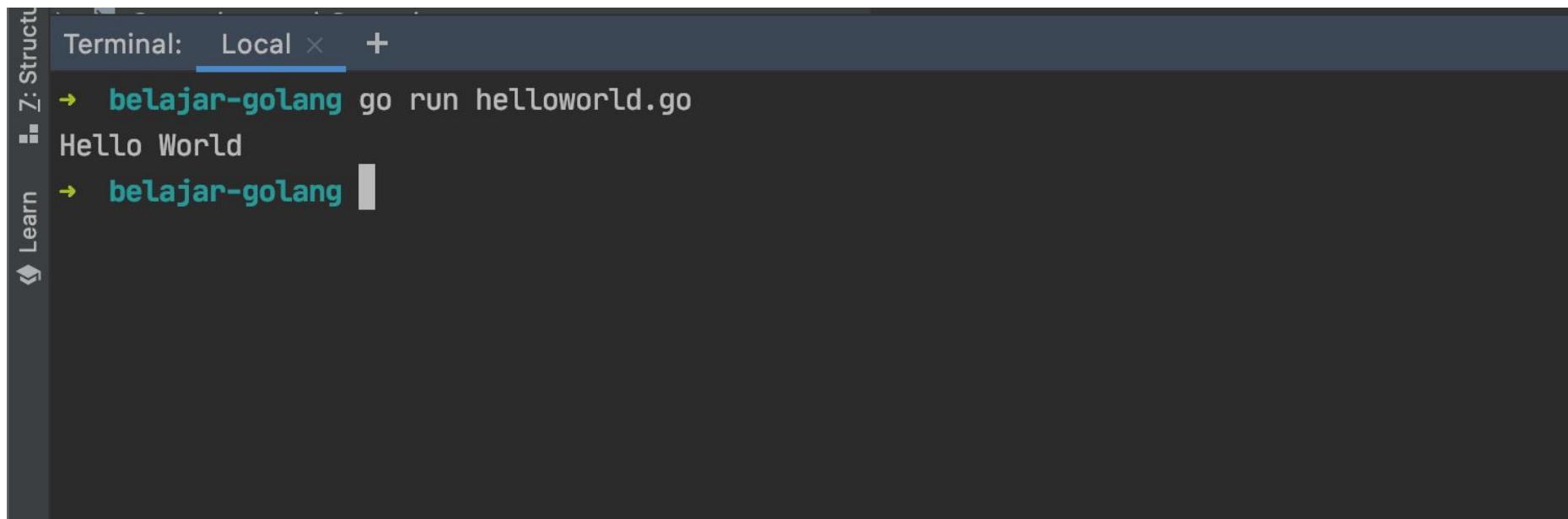
```
1 package main
2
3 import "fmt"
4
5 ► 5 func main() {
6     fmt.Println("Hello World")
7 }
```

The code is numbered from 1 to 10 on the left side. A green play button icon is positioned next to line 5, indicating the current line of execution. The word "func" is highlighted in orange.

# Kompilasi File Go-Lang

```
Terminal: Local × +  
→ belajar-golang go build helloworld.go  
→ belajar-golang ls -l  
total 4256  
-rwxr-xr-x 1 khannedy staff 2174008 May  2 14:29 helloworld  
-rw-r--r-- 1 khannedy staff         76 May  2 14:28 helloworld.go  
→ belajar-golang
```

# Menjalankan Tanpa Kompilasi



The screenshot shows a terminal window titled "Terminal: Local". The command "belajar-golang go run helloworld.go" is entered and executed, resulting in the output "Hello World". The terminal interface includes a sidebar with "Learn" and "Z: Structu" items.

```
Terminal: Local
→ belajar-golang go run helloworld.go
Hello World
→ belajar-golang
```

---

# Tipe Data Number

---

# Tipe Data Number

- Ada dua jenis tipe data Number, yaitu :
  - Integer
  - Floating Point

---

## Tipe Data Integer (1)

Tipe Data	Nilai Minimum	Nilai Maksimum
int8	-128	127
int16	-32768	32767
int32	-2147483648	2147483647
int64	-9223372036854775808	9223372036854775807

---

## Tipe Data Integer (2)

Tipe Data	Nilai Minimum	Nilai Maksimum
uint8	0	255
uint16	0	65535
uint32	0	4294967295
uint64	0	18446744073709551615

---

# Tipe Data Floating Point

Tipe Data	Nilai Minimum	Nilai Maksimum
float32	$1.18 \times 10^{-38}$	$3.4 \times 10^{38}$
float64	$2.23 \times 10^{-308}$	$1.80 \times 10^{308}$
complex64	complex numbers with float32 real and imaginary parts.	
complex128	complex numbers with float64 real and imaginary parts.	

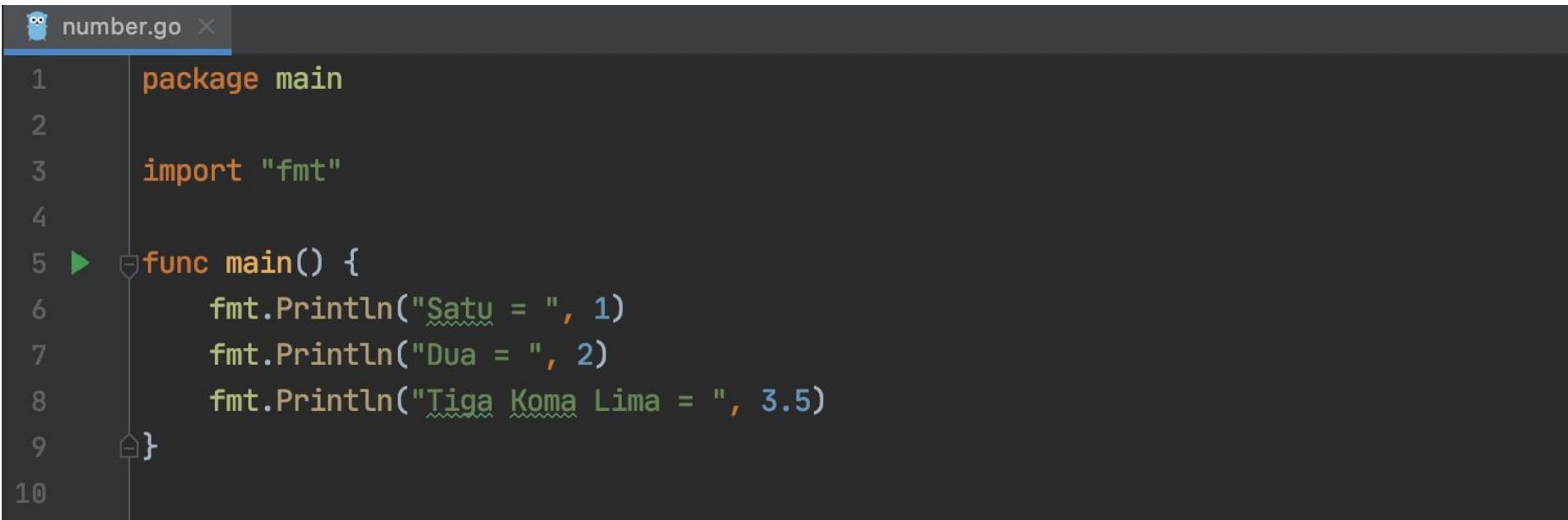


# Alias

Tipe Data	Alias untuk
byte	uint8
rune	int32
int	Minimal int32
uint	Minimal uint32



# Kode Program Number



The screenshot shows a code editor window with a dark theme. The title bar says "number.go". The code editor displays the following Go program:

```
1 package main
2
3 import "fmt"
4
5 ► func main() {
6     fmt.Println("Satu = ", 1)
7     fmt.Println("Dua = ", 2)
8     fmt.Println("Tiga Koma Lima = ", 3.5)
9 }
10
```

The code uses color-coded syntax highlighting. The file is numbered from 1 to 10 on the left. A green arrow icon is next to line 5, indicating it is the current line of execution.

---

# Tipe Data Boolean

---

# Tipe Data Boolean

- Tipe data boolean adalah tipe data yang memiliki nilai dua nilai, yaitu benar atau salah
- Di Go-Lang, tipe data boolean direpresentasikan menggunakan kata kunci bool

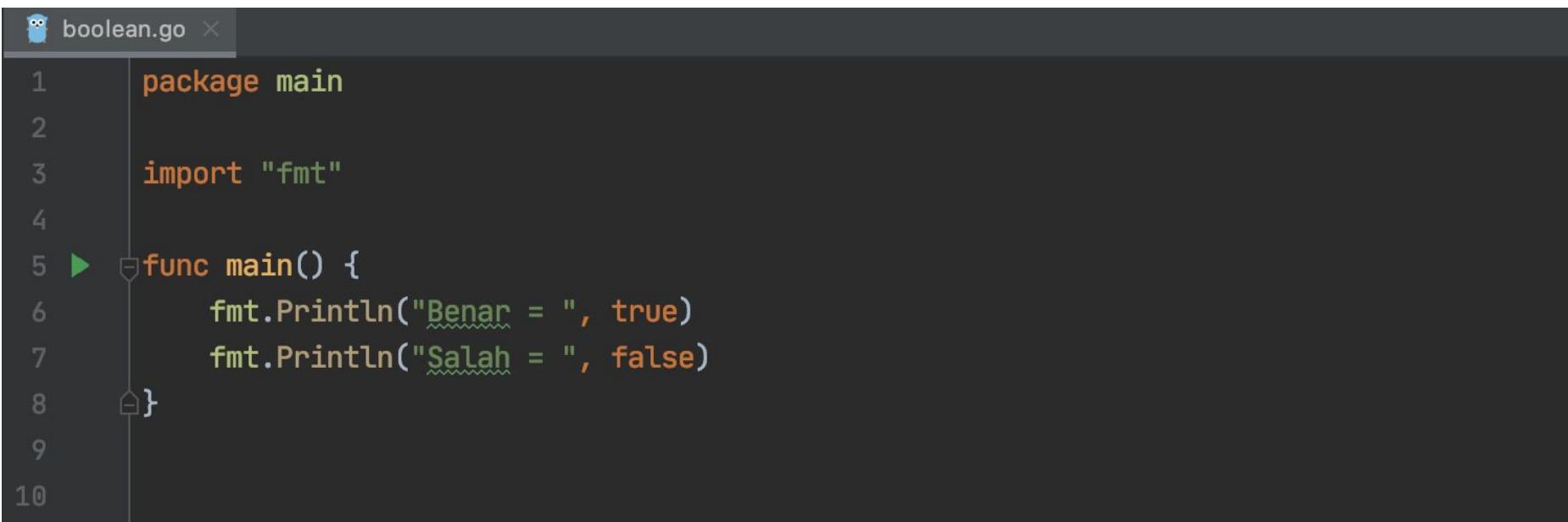


# Boolean

Nilai Boolean	Keterangan
true	Benar
false	Salah

---

# Kode Program Boolean



A screenshot of a code editor showing a file named `boolean.go`. The code is a simple Go program that prints "Benar" and "Salah" to the console using the `fmt.Println` function. The code editor has a dark theme and shows line numbers from 1 to 10.

```
1 package main
2
3 import "fmt"
4
5 ► 6 func main() {
6     fmt.Println("Benar = ", true)
7     fmt.Println("Salah = ", false)
8 }
```

---

# Tipe Data String

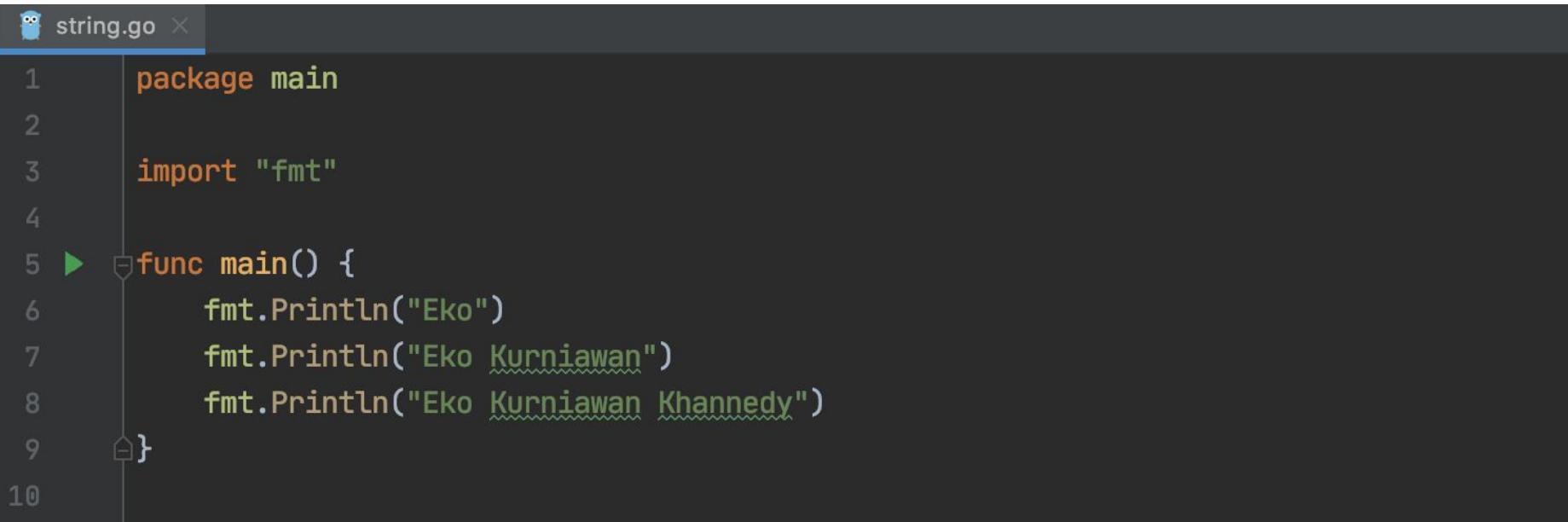
---

# Tipe Data String

- String ada tipe data kumpulan karakter
- Jumlah karakter di dalam String bisa nol sampai tidak terhingga
- Tipe data String di Go-Lang direpresentasikan dengan kata kunci string
- Nilai data String di Go-Lang selalu diawali dengan karakter “ (petik dua) dan diakhiri dengan karakter ” (petik dua)



# Kode Program String



A screenshot of a code editor showing a Go program named `string.go`. The code consists of the following lines:

```
1 package main
2
3 import "fmt"
4
5 ► □ func main() {
6     fmt.Println("Eko")
7     fmt.Println("Eko Kurniawan")
8     fmt.Println("Eko Kurniawan Khannedy")
9 }
10
```

The file icon shows an owl, and the line numbers are on the left. A green arrow points to the first line of the `func main()` block.

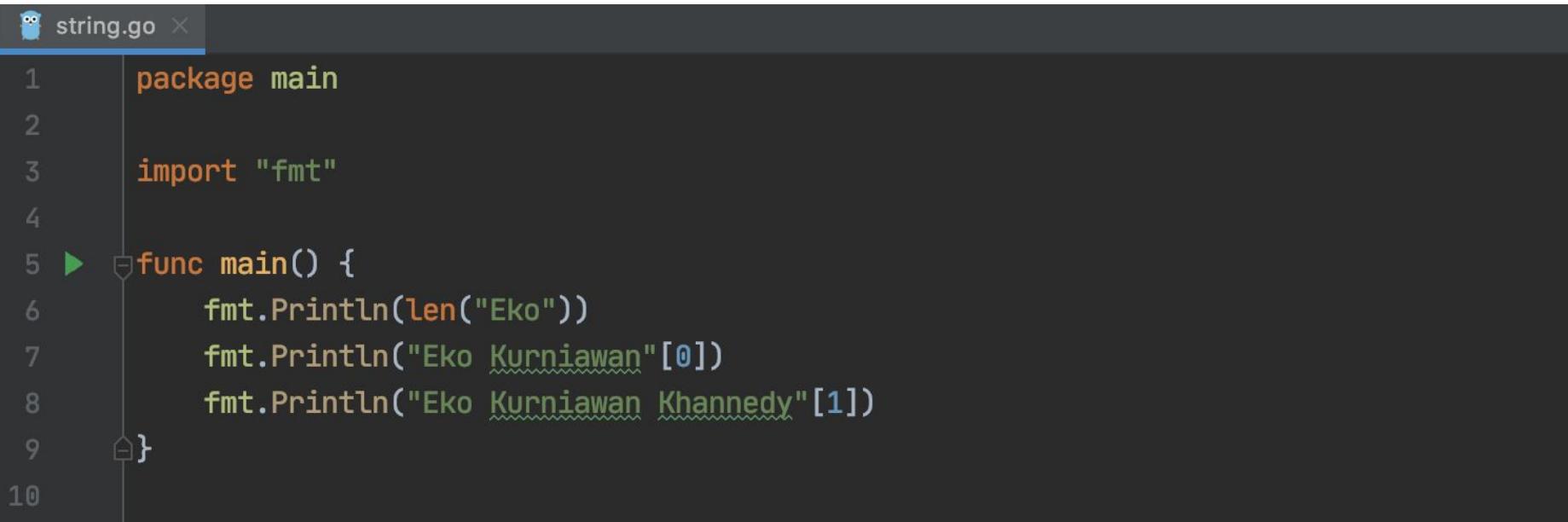
---

# Function untuk String

Function	Keterangan
len("string")	Menghitung jumlah karakter di String
"string"[number]	Mengambil karakter pada posisi yang ditentukan



# Kode Program String



A screenshot of a code editor showing a Go program named `string.go`. The code prints the length of the string "Eko" and its first character.

```
1 package main
2
3 import "fmt"
4
5 ► func main() {
6     fmt.Println(len("Eko"))
7     fmt.Println("Eko Kurniawan"[0])
8     fmt.Println("Eko Kurniawan Khannedy"[1])
9 }
10
```

---

# Variable

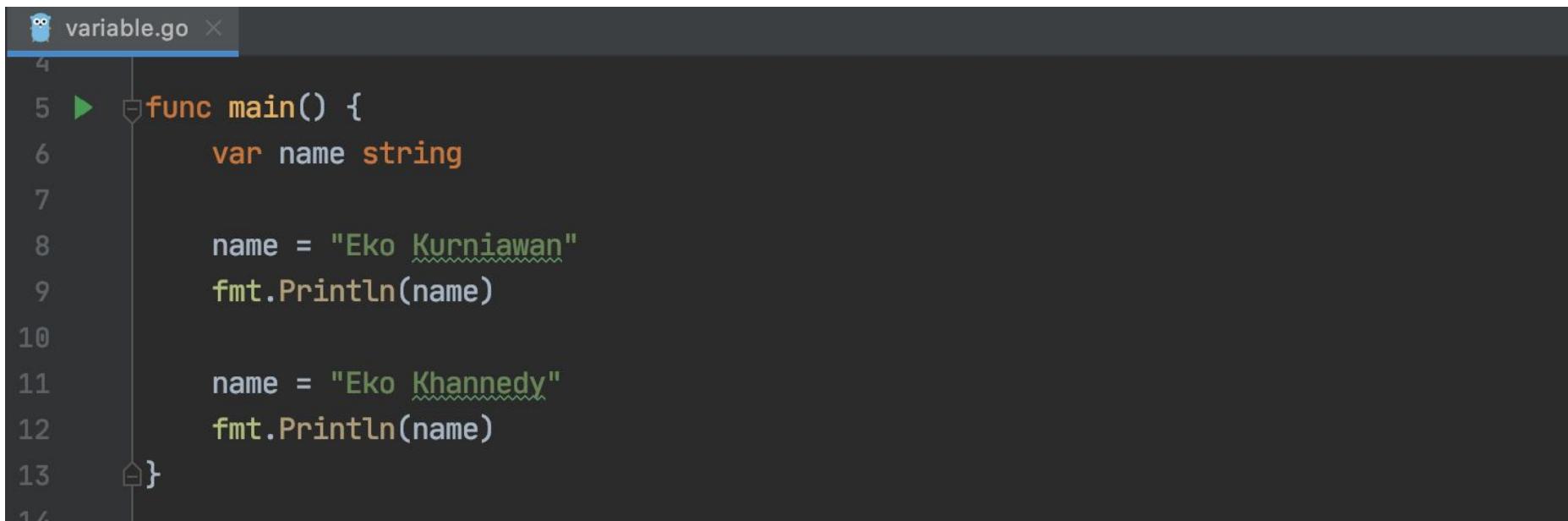
---

# Variable

- Variable adalah tempat untuk menyimpan data
- Variable digunakan agar kita bisa mengakses data yang sama dimanapun kita mau
- Di Go-Lang Variable hanya bisa menyimpan tipe data yang sama, jika kita ingin menyimpan data yang berbeda-beda jenis, kita harus membuat beberapa variable
- Untuk membuat variable, kita bisa menggunakan kata kunci var, lalu diikuti dengan nama variable dan tipe datanya



# Kode Program Variable



The screenshot shows a code editor window with a dark theme. The title bar displays the file name "variable.go". The code editor contains the following Go code:

```
variable.go
4
5 ► func main() {
6     var name string
7
8     name = "Eko Kurniawan"
9     fmt.Println(name)
10
11    name = "Eko Khannedy"
12    fmt.Println(name)
13
14 }
```

The code defines a main function that declares a string variable named "name". It then initializes "name" to "Eko Kurniawan" and prints it using the `fmt.Println` function. Subsequently, it changes the value of "name" to "Eko Khannedy" and prints it again. The code editor interface includes a status bar at the bottom.

---

# Tipe Data Variable

- Saat kita membuat variable, maka kita wajib menyebutkan tipe data variable tersebut
- Namun jika kita langsung menginisialisasikan data pada variable nya, maka kita tidak wajib menyebutkan tipe data variable nya

---

# Kode Program Variable

```
2
3     import "fmt"
4
5 ►  func main() {
6         var name = "Eko Kurniawan"
7         fmt.Println(name)
8
9         name = "Eko Khannedy"
10        fmt.Println(name)
11    }
12
```

---

# Kata Kunci Var

- Di Go-Lang, kata kunci var saat membuat variable tidak lah wajib.
- Asalkan saat membuat variable kita langsung menginisialisasi datanya
- Agar tidak perlu menggunakan kata kunci var, kita perlu menggunakan kata kunci := saat menginisialisasikan data pada variable tersebut

---

# Kode Program Variable

```
2
3     import "fmt"
4
5 ►  func main() {
6         name := "Eko Kurniawan"
7         fmt.Println(name)
8
9         name = "Eko Khannedy"
10        fmt.Println(name)
11    }
12
```

---

# Deklarasi Multiple Variable

- Di Go-Lang kita bisa membuat variable secara sekaligus banyak
- Code yang dibuat akan lebih bagus dan mudah dibaca

---

# Kode Program Variable

```
3 import "fmt"
4
5 ► func main() {
6     var (
7         firstName = "Eko Kurniawan"
8         lastName = "Khannedy"
9     )
10
11    fmt.Println(firstName)
12    fmt.Println(lastName)
13 }
```

---

# Constant

---

# Constant

- Constant adalah variable yang nilainya tidak bisa diubah lagi setelah pertama kali diberi nilai
- Cara pembuatan constant mirip dengan variable, yang membedakan hanya kata kunci yang digunakan adalah const, bukan var
- Saat pembuatan constant, kita wajib langsung menginisialisasikan datanya



# Kode Program Constant

```
2
3 ►  func main() {
4     const firstName string = "Eko"
5     const lastName = "Khannedy"
6
7     // error
8     firstName = "Tidak Bisa Diubah"
9     lastName = "Tidak Bisa Diubah"
10
11
12 }
```

---

# Deklarasi Multiple Constant

- Sama seperti variable, di Go-Lang juga kita bisa membuat constant secara sekaligus banyak



# Kode Program Constant

```
3 ►  func main() {
4  	const (
5    	firstName string = "Eko"
6    	lastName  = "Khannedy"
7  	)
8
9  	// error
10 	firstName = "Tidak Bisa Diubah"
11 	lastName  = "Tidak Bisa Diubah"
12 }
13 }
```

---

# Konversi Tipe Data

---

# Konversi Tipe Data

- Di Go-Lang kadang kita butuh melakukan konversi tipe data dari satu tipe ke tipe lain
- Misal kita ingin mengkonversi tipe data int32 ke int63, dan lain-lain

---

# Kode Program Konversi Tipe Data (1)

```
4
5 ►  func main() {
6      var nilai32 int32 = 32768
7      var nilai64 int64 = int64(nilai32)
8
9      var nilai16 int16 = int16(nilai32)
10
11     fmt.Println(nilai32)
12     fmt.Println(nilai64)
13     fmt.Println(nilai16)
14 }
15
```

---

## Kode Program Konversi Tipe Data (2)

```
14
15     var name = "Eko Kurniawna"
16     var e = name[0]
17     var eString = string(e)
18
19     fmt.Println(name)
20     fmt.Println(eString)
21 }
22
```

---

# Type Declarations

---

# Type Declarations

- Type Declarations adalah kemampuan membuat ulang tipe data baru dari tipe data yang sudah ada
- Type Declarations biasanya digunakan untuk membuat alias terhadap tipe data yang sudah ada, dengan tujuan agar lebih mudah dimengerti



# Kode Program Type Declarations

```
2
3     import "fmt"
4
5 ► 5 func main() {
6
7     type NoKTP string
8
9     var ktpEko NoKTP = "1111111111"
10    fmt.Println(ktpEko)
11    fmt.Println(NoKTP("2222222222"))
12 }
```

---

# Operasi Matematika

---

# Operasi Matematika

Operator	Keterangan
+	Penjumlahan
-	Pengurangan
*	Perkalian
/	Pembagian
%	Sisa Pembagian

---

# Kode Program Operasi Matematika

```
2
3     import "fmt"
4
5 ► 5 func main() {
6         var a = 10
7         var b = 10
8         var c = a + b
9         fmt.Println(c)
10    }
11
```

---

# Augmented Assignments

Operasi Matematika	Augmented Assignments
$a = a + 10$	$a += 10$
$a = a - 10$	$a -= 10$
$a = a * 10$	$a *= 10$
$a = a / 10$	$a /= 10$
$a = a \% 10$	$a \%= 10$

---

# Kode Program Augmented Assignments

```
10
11
12     var i = 10
13     i += 10
14
15     fmt.Println(i)
16 }
17
```

---

# Unary Operator

Operator	Keterangan
<code>++</code>	$a = a + 1$
<code>--</code>	$a = a - 1$
<code>-</code>	Negative
<code>+</code>	Positive
<code>!</code>	Boolean kebalikan

---

# Kode Program Unary Operator

```
16
17     var j = 1
18     j++
19     j++
20
21     fmt.Println(j)
22 }
23
```

---

# Operasi Perbandingan

---

# Operasi Perbandingan

- Operasi perbandingan adalah operasi untuk membandingkan dua buah data
- Operasi perbandingan adalah operasi yang menghasilkan nilai boolean (benar atau salah)
- Jika hasil operasinya adalah benar, maka nilainya adalah true
- Jika hasil operasinya adalah salah, maka nilainya adalah false

---

# Operator Perbandingan

Operator	Keterangan
>	Lebih Dari
<	Kurang Dari
>=	Lebih Dari Sama Dengan
<=	Kurang Dari Sama Dengan
==	Sama Dengan
!=	Tidak Sama Dengan

---

# Kode Program Operasi Perbandingan

```
3 import "fmt"
4
5 ► func main() {
6     var name1 = "Eko"
7     var name2 = "Eko"
8
9     var result bool = name1 == name2
10
11    fmt.Println(result)
12
13 }
```

---

# Operasi Boolean

---

# Operasi Boolean

Operator	Keterangan
&&	Dan
	Atau
!	Kebalikan

---

# Operasi &&

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

---

# Operasi ||

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

---

# Operasi !

Operator	Nilai 2	Hasil
!	true	false
!	false	true

---

# Kode Program Operasi Boolean

```
5 ►  func main() {  
6      var nilaiAkhir = 90  
7      var absensi = 80  
8  
9      var lulusNilaiAkhir bool = nilaiAkhir > 80  
10     var lulusAbsensi bool = absensi > 80  
11  
12     var lulus bool = lulusNilaiAkhir && lulusAbsensi  
13  
14     fmt.Println(lulus)  
15 }
```

---

# Tipe Data Array

---

# Tipe Data Array

- Array adalah tipe data yang berisikan kumpulan data dengan tipe yang sama
- Saat membuat array, kita perlu menentukan jumlah data yang bisa ditampung oleh Array tersebut
- Daya tampung Array tidak bisa bertambah setelah Array dibuat



# Index di Array

Index	Data
0	Eko
1	Kurniawan
2	Khannedy

---

# Kode Program Array

```
5 ► 5 func main() {  
6     var names [3]string  
7     names[0] = "Eko"  
8     names[1] = "Kurniawan"  
9     names[2] = "Khannedy"  
10  
11     fmt.Println(names[0])  
12     fmt.Println(names[1])  
13     fmt.Println(names[2])  
14 }  
15
```

---

# Membuat Array Langsung

- Di Go-Lang kita juga bisa membuat Array secara langsung saat deklarasi variable

---

# Kode Program Array

```
14
15     var values = [3]int{
16         90,
17         80,
18         95,
19     }
20     fmt.Println(values)
21 }
22 |
```

---

# Function Array

Operasi	Keterangan
len(array)	Untuk mendapatkan panjang Array
array[index]	Mendapat data di posisi index
array[index] = value	Mengubah data di posisi index

---

# Kode Program Array

```
15     ↗  var values = [...]int{  
16         90,  
17         80,  
18         95,  
19     }  
20     ↘  fmt.Println(values)  
21     ↘  fmt.Println(len(values))  
22     ↘  values[0] = 100  
23     ↘  fmt.Println(values)  
24     ↗ }  
25
```

---

# Tipe Data Slice

---

# Tipe Data Slice

- Tipe data Slice adalah potongan dari data Array
- Slice mirip dengan Array, yang membedakan adalah ukuran Slice bisa berubah
- Slice dan Array selalu terkoneksi, dimana Slice adalah data yang mengakses sebagian atau seluruh data di Array

---

## Detail Tipe Data Slice

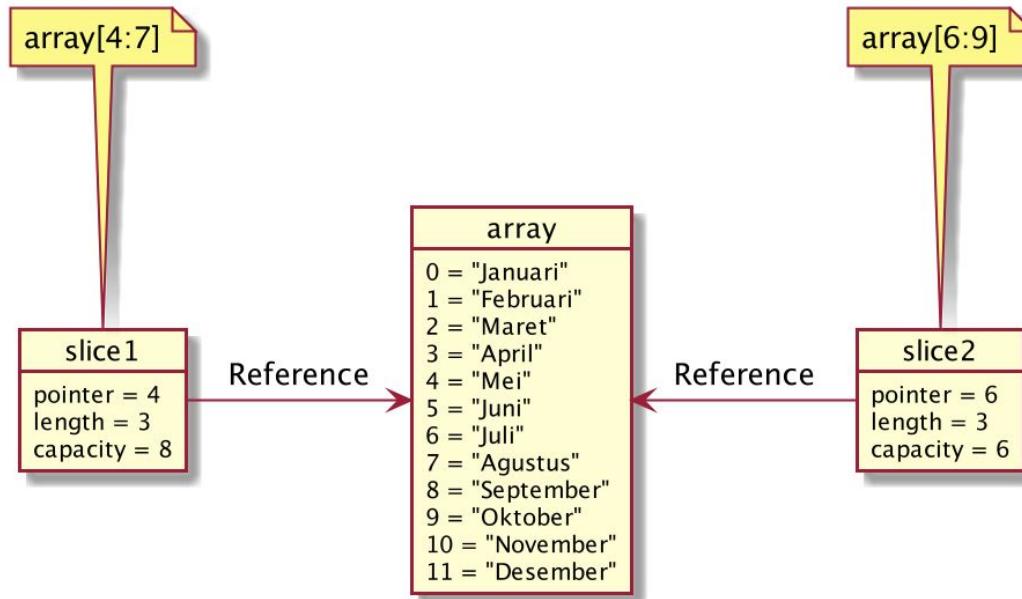
- Tipe Data Slice memiliki 3 data, yaitu pointer, length dan capacity
- Pointer adalah penunjuk data pertama di array para slice
- Length adalah panjang dari slice, dan
- Capacity adalah kapasitas dari slice, dimana length tidak boleh lebih dari capacity

---

# Membuat Slice Dari Array

Membuat Slice	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

# Slice dan Array





# Kode Program Slice

```
1 package main
2
3 import "fmt"
4
5 ► □ func main() {
6     names := [...]string{"Eko", "Kurniawan", "Khannedy", "Joko", "Budi", "Nugraha"}
7     slice := names[4:6]
8
9     fmt.Println(slice[0])
10    fmt.Println(slice[1])
11 }
```

---

# Function Slice

Operasi	Keterangan
len(slice)	Untuk mendapatkan panjang
cap(slice)	Untuk mendapat kapasitas
append(slice, data)	Membuat slice baru dengan menambah data ke posisi terakhir slice, jika kapasitas sudah penuh, maka akan membuat array baru
make([]TypeData, length, capacity)	Membuat slice baru
copy(destination, source)	Menyalin slice dari source ke destination



# Kode Program Append Slice

```
days := [...]string{"Senin", "Selasa", "Rabu", "Kamis", "Jumat", "Sabtu", "Minggu"}  
daySlice1 := days[5:]  
daySlice1[0] = "Sabtu Baru"  
daySlice1[1] = "Minggu Baru"  
fmt.Println(days) // [Senin, Selasa, Rabu, Kamis, Jumat, Sabtu Baru, Minggu Baru]  
  
daySlice2 := append(daySlice1, "Libur Baru")  
daySlice2[0] = "Ups"  
fmt.Println(daySlice2) // [Ups, Minggu Baru, Libur Baru]  
fmt.Println(days) // [Senin, Selasa, Rabu, Kamis, Jumat, Sabtu Baru, Minggu Baru]
```

---

# Kode Program Make Slice

```
17
18     newSlice := make([]string, 2, 5)
19     newSlice[0] = "Eko"
20     newSlice[1] = "Eko"
21
22     fmt.Println(newSlice)
23     fmt.Println(len(newSlice))
24     fmt.Println(cap(newSlice))
25 }
26
```

---

# Kode Program Copy Slice

```
56
37     fromSlice := days[:]
38     toSlice := make([]string, len(fromSlice), cap(fromSlice))
39
40     copy(toSlice, fromSlice)
41
42     fmt.Println(toSlice)
43 }
44
```

---

## Hati-Hati Saat Membuat Array

- Saat membuat Array, kita harus berhati-hati, jika salah, maka yang kita buat bukanlah Array, melainkan Slice

---

# Kode Program Array vs Slice

```
12 //https://play.golang.org/p/200L100y
13
14     iniArray := [...]int{1, 2, 3, 4, 5}
15     iniSlice := []int{1, 2, 3, 4, 5}
16
17     fmt.Println(iniArray)
18     fmt.Println(iniSlice)
19
20 }
```

---

# Tipe Data Map

---

# Tipe Data Map

- Pada Array atau Slice, untuk mengakses data, kita menggunakan index Number dimulai dari 0
- Map adalah tipe data lain yang berisikan kumpulan data yang sama, namun kita bisa menentukan jenis tipe data index yang akan kita gunakan
- Sederhananya, Map adalah tipe data kumpulan key-value (kata kunci - nilai), dimana kata kuncinya bersifat unik, tidak boleh sama
- 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 akan diganti dengan data baru



# Kode Program Map

```
5 ►  ↴func main() {  
6  
7     ↴    person := map[string]string{  
8         "name":      "Eko",  
9         "address":   "Subang",  
10    }  
11  
12    fmt.Println(person)  
13    fmt.Println(person["name"])  
14    fmt.Println(person["address"])  
15
```

---

# Function Map

Operasi	Keterangan
len(map)	Untuk mendapatkan jumlah data di map
map[key]	Mengambil data di map dengan key
map[key] = value	Mengubah data di map dengan key
make(map[TypeKey]TypeValue)	Membuat map baru
delete(map, key)	Menghapus data di map dengan key

---

# Kode Program Map

```
15  
16     book := make(map[string]string)  
17     book["title"] = "Buku Go-Lang"  
18     book["author"] = "Eko Kurniawan"  
19     book["wrong"] = "Ups"  
20  
21     delete(book, "wrong")  
22  
23     fmt.Println(book)  
24  
25 }
```

---

# If Expression

---

# If Expression

- If adalah salah satu kata kunci yang digunakan untuk percabangan
- Percabangan artinya kita bisa mengeksekusi kode program tertentu ketika suatu kondisi terpenuhi
- Hampir di semua bahasa pemrograman mendukung if expression



# Kode Program If

```
6 ►  func main() {  
7     name := "Eko"  
8  
9     if name == "Eko" {  
10        fmt.Println("Hello Eko")  
11    }  
12 }  
13  
14
```

---

# Else Expression

- Blok if akan dieksekusi ketika kondisi if bernilai true
- Kadang kita ingin melakukan eksekusi program tertentu jika kondisi if bernilai false
- Hal ini bisa dilakukan menggunakan else expression

---

# Kode Program Else

```
6 ► func main() {  
7     name := "Eko"  
8  
9     if name == "Eko" {  
10        fmt.Println("Hello Eko")  
11    } else {  
12        fmt.Println("Hi, Boleh Kenalan?)  
13    }  
14}  
15  
16
```

---

# Else If Expression

- Kadang dalam If, kita butuh membuat beberapa kondisi
- Kasus seperti ini, kita bisa menggunakan Else If expression

---

# Kode Program If Else

```
6 ► func main() {  
7     name := "Eko"  
8  
9     if name == "Eko" {  
10        fmt.Println("Hello Eko")  
11    } else if name == "Joko" {  
12        fmt.Println("Hello Joko")  
13    } else {  
14        fmt.Println("Hi, Boleh Kenalan?")  
15    }  
16 }
```

---

## If dengan Short Statement

- If mendukung short statement sebelum kondisi
- Hal ini sangat cocok untuk membuat statement yang sederhana sebelum melakukan pengecekan terhadap kondisi

---

# Kode Program If Short Statement

```
16
17     if length := len(name); length > 5 {
18         fmt.Println("Nama terlalu panjang")
19     } else {
20         fmt.Println("Nama sudah benar")
21     }
22 }
```

---

# Switch Expression

---

# Switch Expression

- Selain if expression, untuk melakukan percabangan, kita juga bisa menggunakan Switch Expression
- Switch expression sangat sederhana dibandingkan if
- Biasanya switch expression digunakan untuk melakukan pengecekan ke kondisi dalam satu variable



# Kode Program Switch

```
5 ► func main() {  
6     name := "Eko"  
7  
8     switch name {  
9     case "Eko":  
10        fmt.Println("Hello Eko")  
11    case "Joko":  
12        fmt.Println("Hello Joko")  
13    default:  
14        fmt.Println("Hi, Boleh Kenalan?")  
15    }  
}
```

---

## **Switch dengan Short Statement**

- Sama dengan If, Switch juga mendukung short statement sebelum variable yang akan di cek kondisinya

---

# Kode Program Switch dengan Short Statement

```
17
18     switch length := len(name); length > 5 {
19         case true:
20             fmt.Println("Nama Terlalu Panjang")
21         case false:
22             fmt.Println("Nama Sudah Benar")
23     }
24 }
25 |
```

---

# Switch Tanpa Kondisi

- Kondisi di switch expression tidak wajib
- Jika kita tidak menggunakan kondisi di switch expression, kita bisa menambahkan kondisi tersebut di setiap case nya

---

# Kode Program Switch Tanda Kondisi

```
25      length := len(name)
26      switch {
27          case length > 10:
28              fmt.Println("Nama Terlalu Panjang")
29          case length > 5:
30              fmt.Println("Nama Lumayan Panjang")
31          default:
32              fmt.Println("Nama Sudah Benar")
33      }
34  }
```

---

# For Loops



# For

- Dalam bahasa pemrograman, biasanya ada fitur yang bernama perulangan
- Salah satu fitur perulangan adalah for loops

---

# Kode Program For

```
6 ► func main() {  
7     counter := 1  
8  
9     for counter <= 10 {  
10        fmt.Println("Perulangan ke ", counter)  
11        counter++  
12    }  
13 }  
14 }
```

---

# For dengan Statement

- Dalam for, kita bisa menambahkan statement, dimana terdapat 2 statement yang bisa tambahkan di for
- Init statement, yaitu statement sebelum for di eksekusi
- Post statement, yaitu statement yang akan selalu dieksekusi di akhir tiap perulangan

---

# Kode Program For dengan Statement

```
13
14     for counter := 1; counter <= 10; counter++ {
15         fmt.Println("Perulangan ke", counter)
16     }
17 }
18 |
```

---

# For Range

- For bisa digunakan untuk melakukan iterasi terhadap semua data collection
- Data collection contohnya Array, Slice dan Map

---

# Kode Program For Range

```
17  
18     names := []string{"Eko", "Kurniawan", "Khannedy"}  
19     for index, name := range names {  
20         fmt.Println("index", index, "=", name)  
21     }  
22 }  
23
```

---

# Break & Continue

---

# Break & Continue

- Break & continue adalah kata kunci yang bisa digunakan dalam perulangan
- Break digunakan untuk menghentikan seluruh perulangan
- Continue adalah digunakan untuk menghentikan perulangan yang berjalan, dan langsung melanjutkan ke perulangan selanjutnya

---

# Kode Program Break

```
5 ►  func main() {  
6     for i := 0; i < 10; i++ {  
7         if i == 5 {  
8             break  
9         }  
10    }  
11    fmt.Println("Perulangan ke", i)  
12 }  
13 }  
14 }
```

---

# Kode Program Continue

```
13
14     for i := 0; i < 10; i++ {
15         if i%2 == 0 {
16             continue
17         }
18
19         fmt.Println("Perulangan ke", i)
20     }
21 }
```

---

# Function

---

# Function

- Sebelumnya kita sudah mengenal sebuah function yang wajib dibuat agar program kita bisa berjalan, yaitu function main
- 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

---

# Kode Program Function

```
4
5  func sayHello() {
6      fmt.Println("Hello")
7  }
8
9 ▶ func main() {
10    sayHello()
11}
12
```

---

# Function Parameter

---

# Function Parameter

- Saat membuat function, kadang-kadang kita membutuhkan data dari luar, atau kita sebut 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

---

# Kode Program Function Parameter

```
4
5  func sayHelloTo(firstName string, lastName string) {
6      fmt.Println("Hello", firstName, lastName)
7  }
8
9 ▶ func main() {
10    sayHelloTo("Eko", "Khannedy")
11}
12
```

---

# Function Return Value

---

# Function Return Value

- Function bisa mengembalikan data
- Untuk memberitahu bahwa function mengembalikan data, kita harus menuliskan tipe data kembalian 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

---

# Kode Program Function Return Value

```
4
5  func getHello(name string) string {
6      return "Hello " + name
7  }
8
9 ▶ func main() {
10    result := getHello("Eko")
11    fmt.Println(result)
12}
13
```

---

# Returning Multiple Values

---

# Returning Multiple Values

- 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

---

# Kode Program Returning Multiple Values

```
4
5  func getFullName() (string, string) {
6      return "Eko", "Khannedy"
7  }
8
9 ▶ func main() {
10    firstName, lastName := getFullName()
11    fmt.Println(firstName, lastName)
12}
13|
```

---

# Menghiraukan Return Value

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

---

# Kode Program Menghiraukan Return Value

```
4
5  func getFullName() (string, string) {
6      return "Eko", "Khannedy"
7  }
8
9 ▶ func main() {
10    firstName, _ := getFullName()
11    fmt.Println(firstName)
12}
13
```

---

# Named Return Values

---

## Named Return Values

- 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



# Kode Program Named Return Values

```
5  func getCompleteName() (firstName, middleName, lastName string) {
6      firstName = "Eko"
7      middleName = "Kurniawan"
8      lastName = "Khannedy"
9
10     return firstName, middleName, lastName
11 }
12
13 ► func main() {
14     firstName, middleName, lastName := getCompleteName()
15     fmt.Println(firstName, middleName, lastName)
```

---

# Variadic Function

---

# Variadic Function

- Parameter yang berada di posisi terakhir, memiliki kemampuan dijadikan sebuah varargs
- Varargs artinya datanya bisa menerima lebih dari satu input, atau anggap saja semacam Array.
- Apa bedanya dengan parameter biasa dengan tipe data Array?
  - Jika parameter tipe Array, kita wajib membuat array terlebih dahulu sebelum mengirimkan ke function
  - Jika parameter menggunakan varargs, kita bisa langsung mengirim data nya, jika lebih dari satu, cukup gunakan tanda koma

---

# Kode Program Variadic Function

```
5  func sumAll(numbers ...int) int {
6      total := 0
7      for _, number := range numbers {
8          total += number
9      }
10     return total
11 }
12
13 ► func main() {
14     total := sumAll(10, 10, 10, 10, 10, 10)
15     fmt.Println(total)
```

---

# Slice Parameter

- Kadang ada kasus dimana kita menggunakan Variadic Function, namun memiliki variable berupa slice
- Kita bisa menjadikan slice sebagai vararg parameter



# Kode Program Slice Parameter

```
12  
13 ►  func main() {  
14     total := sumAll(10, 10, 10, 10, 10, 10)  
15     fmt.Println(total)  
16  
17     numbers := []int{10, 10, 10, 10, 10}  
18     total = sumAll(numbers...)  
19     fmt.Println()  
20 }  
21
```

---

# Function Value

---

# Function Value

- Function adalah first class citizen
- Function juga merupakan tipe data, dan bisa disimpan di dalam variable

---

# Kode Program Function Value

```
5  func getGoodBye(name string) string {
6      return "Good Bye " + name
7  }
8
9 ► func main() {
10    goodbye := getGoodBye
11    fmt.Println(goodbye("Eko"))
12}
13
```

---

# Function as Parameter

---

# Function as Parameter

- Function tidak hanya bisa kita simpan di dalam variable sebagai value
- Namun juga bisa kita gunakan sebagai parameter untuk function lain

---

# Kode Program Function as Parameter (1)

```
5  func sayHelloWithFilter(name string, filter func(string) string) {
6      fmt.Println("Hello ", filter(name))
7  }
8
9  func spamFilter(name string) string {
10     if name == "Anjing" {
11         return "..."
12     } else {
13         return name
14     }
15 }
```

---

## Kode Program Function as Parameter (2)

```
16
17 ► func main() {
18     sayHelloWithFilter("Eko", spamFilter)
19
20     filter := spamFilter
21     sayHelloWithFilter("Anjing", filter)
22 }
23
```

---

# Function Type Declarations

- Kadang jika function terlalu panjang, agak ribet untuk menuliskannya di dalam parameter
- Type Declarations juga bisa digunakan untuk membuat alias function, sehingga akan mempermudah kita menggunakan function sebagai parameter

---

# Kode Program Function Type Declarations

```
4
5     type Filter func(string) string
6
7     func sayHelloWithFilter(name string, filter Filter) {
8         fmt.Println("Hello ", filter(name))
9     }
10
11    func spamFilter(name string) string {
12        if name == "Anjing" {
13            return "..."
14        } else {
15            return name
```

---

# Anonymous Function

---

# Anonymous Function

- Sebelumnya setiap membuat function, kita akan selalu memberikan sebuah nama pada function tersebut
- Namun kadang ada kalanya lebih mudah membuat function secara langsung di variable atau parameter tanpa harus membuat function terlebih dahulu
- Hal tersebut dinamakan anonymous function, atau function tanpa nama

---

# Kode Program Anonymous Function (1)

```
4
5     type Blacklist func(string) bool
6
7     func registerUser(name string, blacklist Blacklist) {
8         if blacklist(name) {
9             fmt.Println("You Are Blocked", name)
10        } else {
11            fmt.Println("Welcome", name)
12        }
13    }
14
15 ► func main() {
```

---

## Kode Program Anonymous Function (2)

```
14
15 ► func main() {
16     blacklist := func(name string) bool {
17         return name == "anjing"
18     }
19
20     registerUser("eko", blacklist)
21     registerUser("anjing", func(name string) bool {
22         return name == "anjing"
23     })
24 }
25 }
```

---

# Recursive Function

---

# Recursive Function

- Recursive function adalah function yang memanggil function dirinya sendiri
- Kadang dalam pekerjaan, kita sering menemui kasus dimana menggunakan recursive function lebih mudah dibandingkan tidak menggunakan recursive function
- Contoh kasus yang lebih mudah diselesaikan menggunakan recursive adalah Factorial

---

# Kode Program Factorial For Loop

```
1
2
3
4
5  ↗ func factorialLoop(value int) int {
6
7    ↗ for i := value; i > 0; i-- {
8      result *= i
9    }
10   return result
11 }
12
13 ➤ ↗ func main() {
14   fmt.Println(factorialLoop(10))
15 }
```

---

# Kode Program Factorial Recursive

```
12
13     func factorialRecursive(value int) int {
14         if value == 1 {
15             return 1
16         } else {
17             ⚡ return value * factorialRecursive(value-1)
18         }
19     }
20
21 ➜ func main() {
22     fmt.Println(factorialRecursive(10))
23 }
```

---

# Closure

---

# Closures

- Closure adalah kemampuan sebuah function berinteraksi dengan data-data disekitarnya dalam scope yang sama
- Harap gunakan fitur closure ini dengan bijak saat kita membuat aplikasi



# Kode Program Function

```
5 ►  ↗ func main() {  
6     counter := 0  
7     ↗ increment := func() {  
8         fmt.Println("Increment")  
9         counter++  
10    }  
11  
12    increment()  
13    increment()  
14    fmt.Println(counter)  
15 }
```

---

# Defer, Panic dan Recover

---

# Defer

- Defer function adalah function yang bisa kita jadwalkan untuk dieksekusi setelah sebuah function selesai di eksekusi
- Defer function akan selalu dieksekusi walaupun terjadi error di function yang dieksekusi

# Kode Program Defer

```
5  func logging() {
6      fmt.Println("Selesai memanggil function")
7  }
8
9  func runApplication() {
10     defer logging()
11     fmt.Println("Run Application")
12 }
13
14 ► func main() {
15     runApplication()
```

---

# Panic

- Panic function adalah function yang bisa kita gunakan untuk menghentikan program
- Panic function biasanya dipanggil ketika terjadi error pada saat program kita berjalan
- Saat panic function dipanggil, program akan terhenti, namun defer function tetap akan dieksekusi



# Kode Program Panic

```
4
5      ↗ func endApp() {
6          ↗ fmt.Println("End App")
7      }
8
9      ↗ func runApp(error bool) {
10         ↗ defer endApp()
11         ↗ if error {
12             ↗     ↗ panic("ERROR")
13         }
14     }
15 }
```



# Recover

- Recover adalah function yang bisa kita gunakan untuk menangkap data panic
- Dengan recover proses panic akan terhenti, sehingga program akan tetap berjalan

---

# Kode Program Recover Salah

```
7  }
8
9  func runApp(error bool) {
10    defer endApp()
11    if error {
12      panic("ERROR")
13    }
14    message := recover()
15    fmt.Println("Terjadi Error", message)
16
17 }
```

# Kode Program Recover Benar

```
5  func endApp() {
6      fmt.Println("End App")
7      message := recover()
8      fmt.Println("Terjadi Error", message)
9  }
10
11 func runApp(error bool) {
12     defer endApp()
13     if error {
14         panic("ERROR")
15 }
```

---

# Komentar

---

# Komentar

- Komentar terbaik pada kode adalah kode itu sendiri
- Saat membuat kode, kita perlu membuat kode semudah mungkin untuk dibaca
- Namun kadang juga kita butuh menambahkan komentar di kode kita

---

# Kode Program Komentar

```
1 package main
2
3 /**
4  * This is multi line comment
5  * You can add a lot of comments here
6  */
7 > func main() {
8     // this is single line comment
9 }
10
```

---

# Struct

---

# Struct

- Struct adalah sebuah template data yang digunakan untuk menggabungkan nol atau lebih tipe data lainnya dalam satu kesatuan
- Struct biasanya representasi data dalam program aplikasi yang kita buat
- Data di struct disimpan dalam field
- Sederhananya struct adalah kumpulan dari field



# Kode Program Struct

```
6
7
8 type Customer struct {
9     Name, Address string
10    Age          int
11 }
12
13
14
15
16
17
```

---

# Membuat Data Struct

- Struct adalah template data atau prototype data
- Struct tidak bisa langsung digunakan
- Namun kita bisa membuat data/object dari struct yang telah kita buat



# Kode Program Membuat Data Struct

```
10
11 ►  func main() {
12     var eko Customer
13     eko.Name = "Eko Kurniawan"
14     eko.Address = "Indonesia"
15     eko.Age = 30
16
17     fmt.Println(eko)
18 }
19
```

---

# Struct Literals

- Sebelumnya kita telah membuat data dari struct, namun sebenarnya ada banyak cara yang bisa kita gunakan untuk membuat data dari struct

---

# Kode Program Struct Literals

```
18  
19     joko := Customer{  
20         Name:      "Joko",  
21         Address:   "Indonesia",  
22         Age:       30,  
23     }  
24     fmt.Println(joko)  
25  
26     budi := Customer{"Budi", "Indonesia", 30}  
27     fmt.Println(budi)  
28 }  
29 |
```

---

# Struct Method

---

# Struct Method

- Struct adalah tipe data seperti tipe data lainnya, dia bisa digunakan sebagai parameter untuk function
- Namun jika kita ingin menambahkan method ke dalam structs, sehingga seakan-akan sebuah struct memiliki function
- Method adalah function

# Kode Program Struct Method

```
8 }  
9  
10 func (customer Customer) sayHello() {  
11     fmt.Println("Hello, My Name is", customer.Name)  
12 }  
13  
14 ► func main() {  
15     rully := Customer{Name: "Rully"}  
16     rully.sayHello()  
17  
18 }
```

---

# Interface



# Interface

- Interface adalah tipe data Abstract, dia tidak memiliki implementasi langsung
- Sebuah interface berisikan definisi-definisi method
- Biasanya interface digunakan sebagai kontrak



# Kode Program Interface

```
4
5 ①↓ ⌂ type HasName interface {
6    ①↓   GetName() string
7    ⌂ }
8
9   ⌂ func SayHello(hasName HasName) {
10      fmt.Println("Hello", hasName.GetName())
11    ⌂ }
12
13
14
```

---

# Implementasi Interface

- Setiap tipe data yang sesuai dengan kontrak interface, secara otomatis dianggap sebagai interface tersebut
- Sehingga kita tidak perlu mengimplementasikan interface secara manual
- Hal ini agak berbeda dengan bahasa pemrograman lain yang ketika membuat interface, kita harus menyebutkan secara eksplisit akan menggunakan interface mana

---

# Kode Program Implementasi Interface (1)

```
13 i↑ type Person struct {
14     Name string
15 }
16
17 i↑ func (person Person) GetName() string {
18     return person.Name
19 }
20
21 ► func main() {
22     person := Person{Name: "Eko"}
23     SayHello(person)
```

## Kode Program Implementasi Interface (2)

```
21 ① type Animal struct {
22     Name string
23 }
24
25 ① func (animal Animal) GetName() string {
26     return animal.Name
27 }
28
29 ► func main() {
30     animal := Animal{Name: "Kucing"}
31     SayHello(animal)
```

---

# Interface Kosong

---

# Interface Kosong

- Go-Lang bukanlah bahasa pemrograman yang berorientasi objek
- Biasanya dalam pemrograman berorientasi objek, ada satu data parent di puncak yang bisa dianggap sebagai semua implementasi data yang ada di bahasa pemrograman tersebut
- Contoh di Java ada `java.lang.Object`
- Untuk menangani kasus seperti ini, di Go-Lang kita bisa menggunakan interface kosong
- Interface kosong adalah interface yang tidak memiliki deklarasi method satupun, hal ini membuat secara otomatis semua tipe data akan menjadi implementasi nya

---

# Penggunaan Interface Kosong di Go-Lang

- Ada banyak contoh penggunaan interface kosong di Go-Lang, seperti :
  - fmt.Println(a ...interface{})
  - panic(v interface{})
  - recover() interface{}
  - dan lain-lain

---

# Kode Program Interface Kosong

```
4
5  func Ups() interface{} {
6      // return 1
7      // return true
8      return "Ups"
9  }
10
11 func main() {
12     kosong := Ups()
13     fmt.Println(kosong)
14 }
15
```

---

Nil

---

# Nil

- Biasanya di dalam bahasa pemrograman lain, object yang belum diinisialisasi maka secara otomatis nilainya adalah null atau nil
- Berbeda dengan Go-Lang, di Go-Lang saat kita buat variable dengan tipe data tertentu, maka secara otomatis akan dibuatkan default value nya
- Namun di Go-Lang ada data nil, yaitu data kosong
- Nil sendiri hanya bisa digunakan di beberapa tipe data, seperti interface, function, map, slice, pointer dan channel

---

# Kode Program Nil (1)

```
4
5     func NewMap(name string) map[string]string {
6         if name == "" {
7             return nil
8         } else {
9             return map[string]string{
10                 "name": name,
11             }
12         }
13     }
14 }
```

---

## Kode Program Nil (2)

```
14
15 ► | func main() {
16 |     data := NewMap("")
17 |     if data == nil {
18 |         fmt.Println("Data Kosong")
19 |     } else {
20 |         fmt.Println(data)
21 |     }
22 | }
23
```

---

# error Interface



# error Interface

- Go-Lang memiliki interface yang digunakan sebagai kontrak untuk membuat error, nama interface nya adalah error

```
// The error built-in interface type is the conventional interface for
// representing an error condition, with the nil value representing no error.
type error interface {
    Error() string
}
```

---

# Membuat Error

- Untuk membuat error, kita tidak perlu manual.
- Go-Lang sudah menyediakan library untuk membuat helper secara mudah, yang terdapat di package errors (Package akan kita bahas secara detail di materi tersendiri)

---

# Kode Program error Interface (1)

```
3 import (
4     "errors"
5     "fmt"
6 )
7
8 func Pembagian(nilai int, pembagi int) (int, error) {
9     if pembagi == 0 {
10         return 0, errors.New("Pembagian Dengan NOL")
11     } else {
12         return nilai / pembagi, nil
13     }
}
```

---

## Kode Program error Interface (2)

```
15
16 ► func main() {
17     hasil, err := Pembagian(100, 10)
18     if err == nil {
19         fmt.Println("Hasil", hasil)
20     } else {
21         fmt.Println("Error", err.Error())
22     }
23 }
24 |
```

---

# Type Assertions

---

# Type Assertions

- Type Assertions merupakan kemampuan merubah tipe data menjadi tipe data yang diinginkan
- Fitur ini sering sekali digunakan ketika kita bertemu dengan data interface kosong

---

# Kode Program Type Assertions

```
5  func random() interface{} {
6      return "OK"
7  }
8
9 ▶ func main() {
10    result := random()
11    resultString := result.(string)
12    fmt.Println(resultString)
13
14    resultInt := result.(int) // panic
15    fmt.Println(resultInt)
```

---

# Type Assertions Menggunakan Switch

- Saat salah menggunakan type assertions, maka bisa berakibat terjadi panic di aplikasi kita
- Jika panic dan tidak ter-recover, maka otomatis program kita akan mati
- Agar lebih aman, sebaiknya kita menggunakan switch expression untuk melakukan type assertions

---

# Kode Program Type Assertions Switch

```
9 ► func main() {
10     result := random()
11     switch value := result.(type) {
12         case string:
13             fmt.Println("String", value)
14         case int:
15             fmt.Println("Int", value)
16         default:
17             fmt.Println("Unknown")
18     }
19 }
```

---

# Pointer

---

# Pass by Value

- Secara default di Go-Lang semua variable itu di passing by value, bukan by reference
- Artinya, jika kita mengirim sebuah variable ke dalam function, method atau variable lain, sebenarnya yang dikirim adalah duplikasi value nya

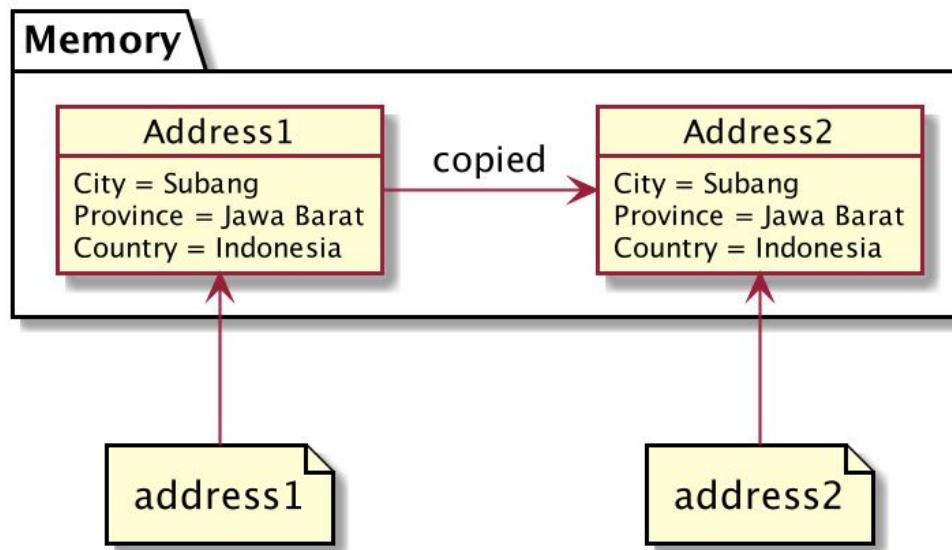
---

# Kode Program Pass by Value

```
8
9 ► func main() {
10    address1 := Address{"Subang", "Jawa Barat", "Indonesia"}
11    address2 := address1
12
13    address2.City = "Bandung"
14
15    fmt.Println(address1) // address1 tidak berubah
16    fmt.Println(address2)
17 }
18
```

---

# Penjelasan Detail Pass by Value



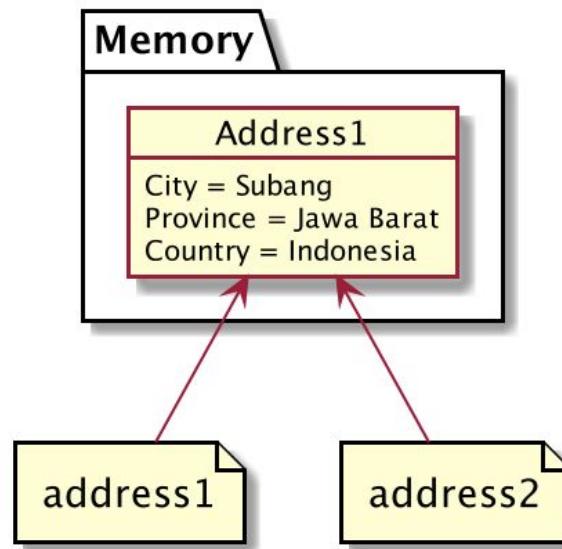


# Pointer

- Pointer adalah kemampuan membuat reference ke lokasi data di memory yang sama, tanpa menduplikasi data yang sudah ada
- Sederhananya, dengan kemampuan pointer, kita bisa membuat pass by reference

---

# Pass by Reference dengan Pointer





# Operator &

- Untuk membuat sebuah variable dengan nilai pointer ke variable yang lain, kita bisa menggunakan operator & diikuti dengan nama variable nya



# Kode Program Operator &

```
8
9 ►  func main() {
10    address1 := Address{"Subang", "Jawa Barat", "Indonesia"}
11    address2 := &address1
12
13    address2.City = "Bandung"
14
15    fmt.Println(address1)
16    fmt.Println(address2)
17 }
18
```

---

# Operator \*

- Saat kita mengubah variable pointer, maka yang berubah hanya variable tersebut.
- Semua variable yang mengacu ke data yang sama tidak akan berubah
- Jika kita ingin mengubah seluruh variable yang mengacu ke data tersebut, kita bisa menggunakan operator \*

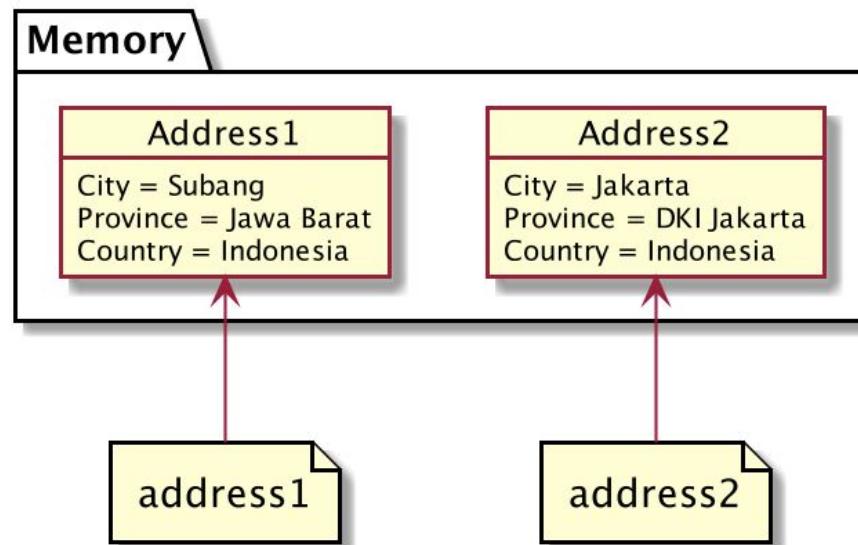
---

# Kode Program Operator \* (1)

```
8
9 ► func main() {
10    address1 := Address{"Subang", "Jawa Barat", "Indonesia"}
11    address2 := &address1
12
13    address2.City = "Bandung"
14
15    address2 = &Address{"Jakarta", "DKI Jakarta", "Indonesia"}
16
17    fmt.Println(address1) // address 1 tidak berubah
18    fmt.Println(address2)
19 }
```

---

# Tanpa Operator \*



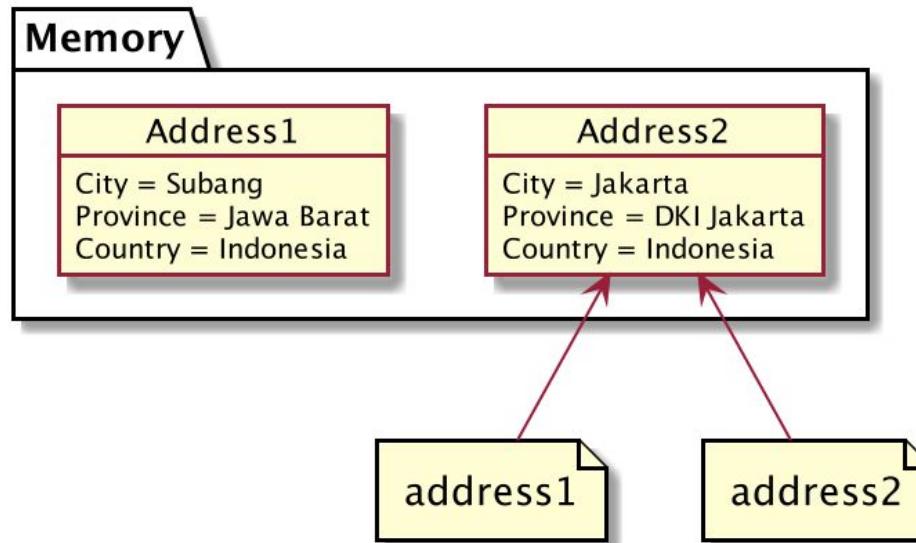
---

## Kode Program Operator \* (2)

```
9 ►  func main() {  
10     address1 := Address{"Subang", "Jawa Barat", "Indonesia"}  
11     address2 := &address1  
12  
13     address2.City = "Bandung"  
14  
15     *address2 = Address{"Jakarta", "DKI Jakarta", "Indonesia"}  
16  
17     fmt.Println(address1) // address 1 berubah!  
18     fmt.Println(address2)  
19 }
```

---

# Dengan Operator \*



---

# Function new

- Sebelumnya untuk membuat pointer dengan menggunakan operator &
- Go-Lang juga memiliki function new yang bisa digunakan untuk membuat pointer
- Namun function new hanya mengembalikan pointer ke data kosong, artinya tidak ada data awal

---

# Kode Program Function new

```
19
20     alamat1 := new(Address)
21     alamat2 := alamat1
22
23     alamat2.Country = "Indonesia"
24
25     fmt.Println(alamat1) // alamat 1 berubah
26     fmt.Println(alamat2)
27
28 }
29
```

---

# Pointer di Function

---

# Pointer di Function

- Saat kita membuat parameter di function, secara default adalah pass by value, artinya data akan di copy lalu dikirim ke function tersebut
- Oleh karena itu, jika kita mengubah data di dalam function, data yang aslinya tidak akan pernah berubah.
- Hal ini membuat variable menjadi aman, karena tidak akan bisa diubah
- Namun kadang kita ingin membuat function yang bisa mengubah data asli parameter tersebut
- Untuk melakukan ini, kita juga bisa menggunakan pointer di function
- Untuk menjadikan sebuah parameter sebagai pointer, kita bisa menggunakan operator \* di parameternya

# Kode Program Pointer di Function (1)

```
6
9  func ChangeAddressToIndonesia(address Address) {
10    address.Country = "Indonesia"
11 }
12
13 ► func main() {
14   address := Address{"Subang", "Jawa Barat", ""}
15   ChangeAddressToIndonesia(address)
16
17   fmt.Println(address) // tidak berubah
18 }
19
```

## Kode Program Pointer di Function (2)

```
6
9  func ChangeAddressToIndonesia(address *Address) {
10    address.Country = "Indonesia"
11 }
12
13 ► func main() {
14   address := Address{"Subang", "Jawa Barat", ""}
15   ChangeAddressToIndonesia(&address)
16
17   fmt.Println(address) // berubah
18 }
19
```

---

# Pointer di Method

---

# Pointer di Method

- Walaupun method akan menempel di struct, tapi sebenarnya data struct yang diakses di dalam method adalah pass by value
- Sangat direkomendasikan menggunakan pointer di method, sehingga tidak boros memory karena harus selalu diduplikasi ketika memanggil method

---

# Kode Program Pointer di Method (1)

```
5  type Man struct {
6      Name string
7  }
8
9  func (man Man) Married() {
10     man.Name = "Mr. " + man.Name
11 }
12
13 ►  func main() {
14     eko := Man{"Eko"}
15     eko.Married()
```

---

## Kode Program Pointer di Method (2)

```
5  type Man struct {
6      Name string
7  }
8
9  func (man *Man) Married() {
10     man.Name = "Mr. " + man.Name
11 }
12
13 ►  func main() {
14     eko := Man{"Eko"}
15     eko.Married()
```

---

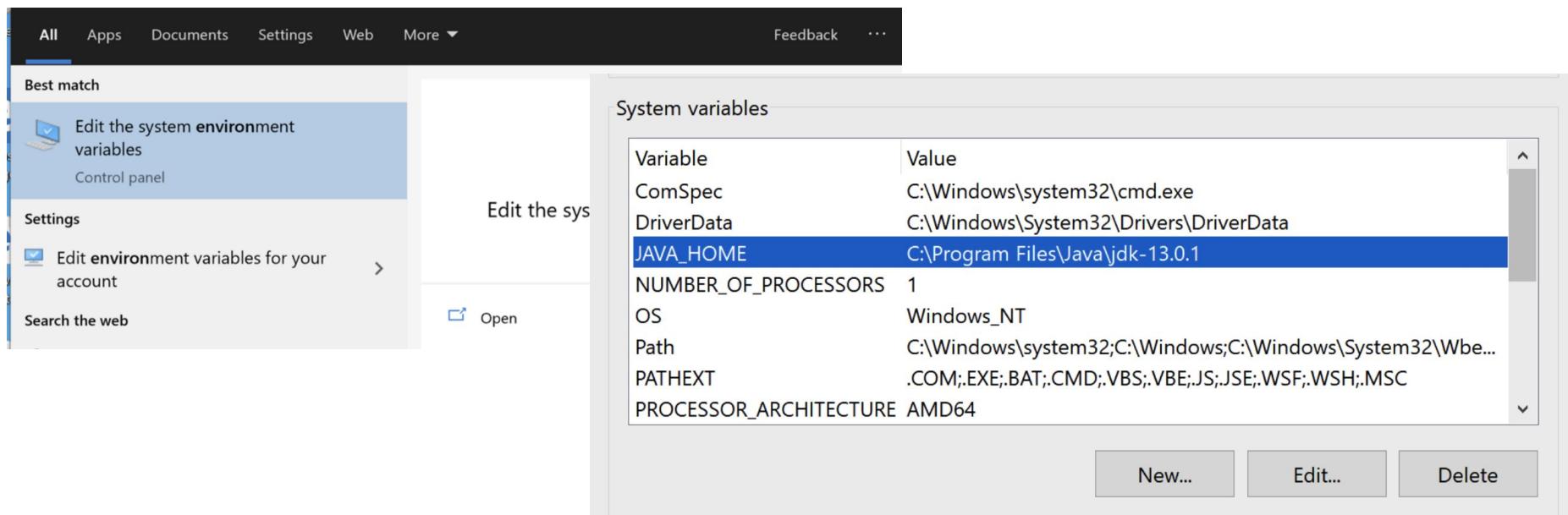
# GOPATH

---

# GOPATH

- Sebelumnya saat kita membuat project aplikasi Go-Lang kita belum membahas tentang GOPATH
- GOPATH adalah environment variable yang berisikan lokasi tempat kita menyimpan project dan library Go-Lang
- GOPATH wajib di buat ketika kita mulai membuat aplikasi lebih dari satu file atau butuh menggunakan library dari luar

# Setting GOPATH di Windows

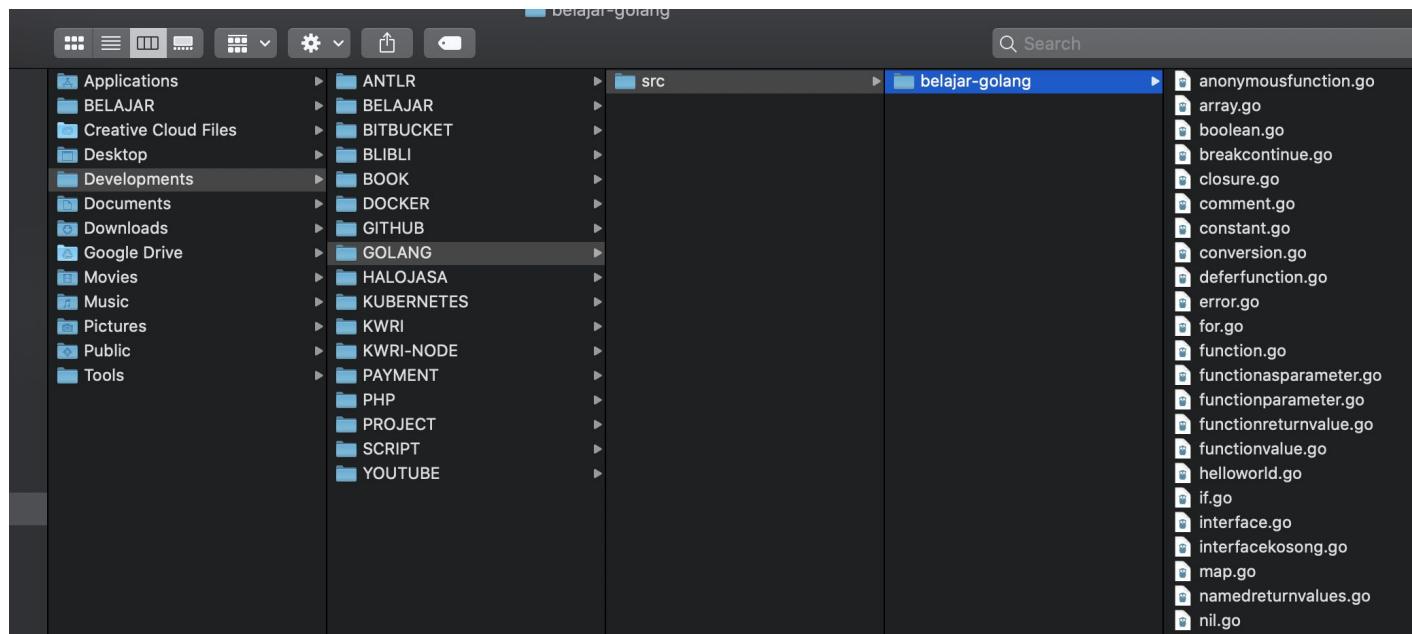


---

# Setting GOPATH di Linux atau Mac

```
# Add to .bashrc or .zshrc or .profile  
  
export GOPATH="/Users/khannedy/Developments/GOLANG"
```

# Struktur Direktori GOPATH



---

# Package & Import

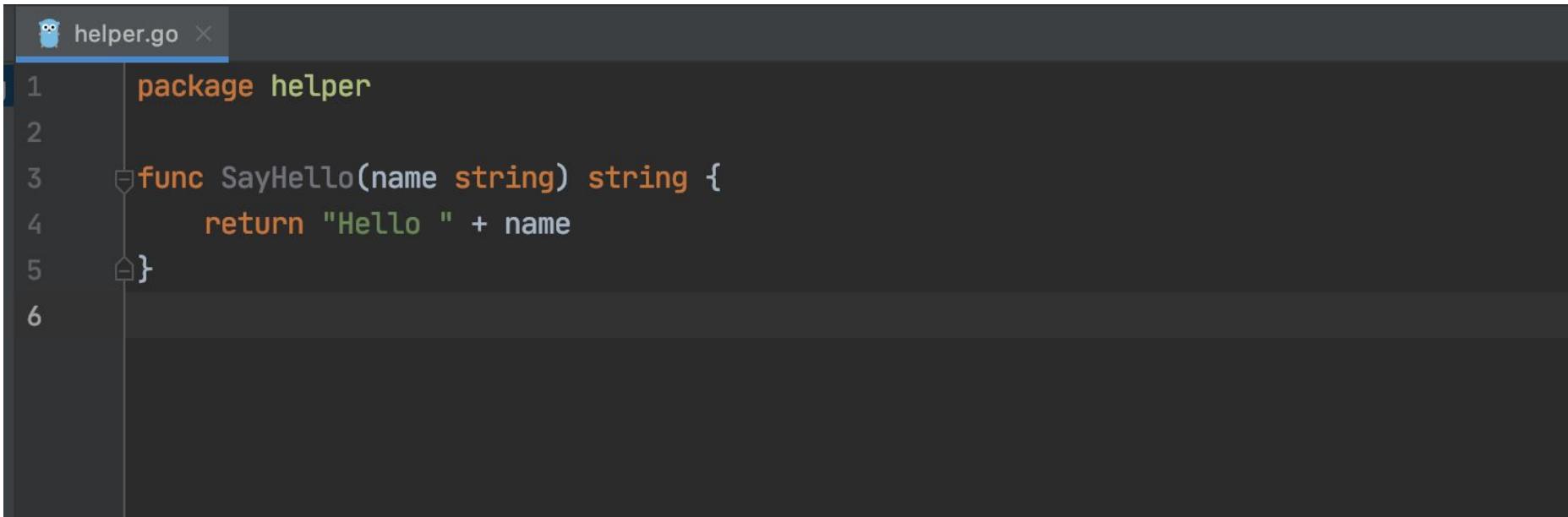
---

# Package

- Package adalah tempat yang bisa digunakan untuk mengorganisir kode program yang kita buat di Go-Lang
- Dengan menggunakan package, kita bisa merapikan kode program yang kita buat
- Package sendiri sebenarnya hanya direktori folder di sistem operasi kita

---

# Kode Program Package



A screenshot of a code editor showing a single file named `helper.go`. The file contains the following Go code:

```
1 package helper
2
3 func SayHello(name string) string {
4     return "Hello " + name
5 }
6
```

---

# Import

- Secara standar, file Go-Lang hanya bisa mengakses file Go-Lang lainnya yang berada dalam package yang sama
- Jika kita ingin mengakses file Go-Lang yang berada diluar package, maka kita bisa menggunakan Import



# Kode Program Import

```
2
3 import (
4     "belajar-golang/helper"
5     "fmt"
6 )
7
8 ► func main() {
9     result := helper.SayHello("Eko")
10    fmt.Println(result)
11 }
12 |
```

---

# Access Modifier

---

# Access Modifier

- Di bahasa pemrograman lain, biasanya ada kata kunci yang bisa digunakan untuk menentukan access modifier terhadap suatu function atau variable
- Di Go-Lang, untuk menentukan access modifier, cukup dengan nama function atau variable
- Jika nama nya diawali dengan hurup besar, maka artinya bisa diakses dari package lain, jika dimulai dengan hurup kecil, artinya tidak bisa diakses dari package lain

---

# Kode Program Access Modifier

```
1 package helper
2
3 var version = "1.0.0" // tidak bisa diakses dari luar
4 var Application = "golang"
5
6 // Tidak bisa diakses dari luar package
7 func sayGoodBye(name string) string {
8     return "Hello " + name
9 }
10
11 func SayHello(name string) string {
```

---

# Package Initialization

---

# Package Initialization

- Saat kita membuat package, kita bisa membuat sebuah function yang akan diakses ketika package kita diakses
- Ini sangat cocok ketika contohnya, jika package kita berisi function-function untuk berkomunikasi dengan database, kita membuat function inisialisasi untuk membuka koneksi ke database
- Untuk membuat function yang diakses secara otomatis ketika package diakses, kita cukup membuat function dengan nama init

---

# Kode Program Package Initialization (1)

```
1 package database\n\n2\n3     var connection string\n4\n5     func init() {\n6         connection = "MySQL"\n7     }\n8\n9     func GetDatabase() string {\n10        return connection\n11    }
```

---

# Kode Program Package Initialization (2)

```
1 package main
2
3 import (
4     "belajar-golang/database"
5     "fmt"
6 )
7
8 func main() {
9     fmt.Println(database.GetDatabase())
10 }
11 |
```

---

# Blank Identifier

- Kadang kita hanya ingin menjalankan init function di package tanpa harus mengeksekusi salah satu function yang ada di package
- Secara default, Go-Lang akan komplen ketika ada package yang di import namun tidak digunakan
- Untuk menangani hal tersebut, kita bisa menggunakan blank identifier (\_) sebelum nama package ketika melakukan import

---

# Package os

---

# Package os

- Go-Lang telah menyediakan banyak sekali package bawaan, salah satunya adalah package os
- Package os berisikan fungsionalitas untuk mengakses fitur sistem operasi secara independen (bisa digunakan disemua sistem operasi)
- <https://golang.org/pkg/os/>

---

# Kode Program Package os (1)

```
1 package main
2
3 import (
4     "fmt"
5     "os"
6 )
7
8 ►  func main() {
9     args := os.Args
10    fmt.Println(args)
11 }
```

---

## Kode Program Package os (2)

```
11  
12     hostname, err := os.Hostname()  
13     if err == nil {  
14         fmt.Println(hostname)  
15     } else {  
16         fmt.Println("Error", err.Error())  
17     }  
18 }  
19
```

---

# Package flag

---

# Package flag

- Package flag berisikan fungsionalitas untuk memarsing command line argument
- <https://golang.org/pkg/flag/>



# Kode Program Package flag

```
7
8 ►  func main() {
9      host := flag.String("host", "localhost", "Put your database host")
10     username := flag.String("username", "root", "Put your database username")
11     password := flag.String("password", "root", "Put your database password")
12
13     flag.Parse()
14
15     fmt.Println(*host, *username, *password)
16 }
17 |
```

---

# Package strings

---

# Package strings

- Package strings adalah package yang berisikan function-function untuk memanipulasi tipe data String
- Ada banyak sekali function yang bisa kita gunakan
- <https://golang.org/pkg/strings/>

---

# Beberapa Function di Package strings

Function	Kegunaan
strings.Trim(string, cutset)	Memotong cutset di awal dan akhir string
strings.ToLower(string)	Membuat semua karakter string menjadi lower case
strings.ToUpper(string)	Membuat semua karakter string menjadi upper case
strings.Split(string, separator)	Memotong string berdasarkan separator
strings.Contains(string, search)	Mengecek apakah string mengandung string lain
strings.ReplaceAll(string, from, to)	Mengubah semua string dari from ke to



# Kode Program Package strings

```
7
8 ►  func main() {
9      fmt.Println(strings.Contains("Eko Kurniawan", "Eko"))
10     fmt.Println(strings.Split("Eko Kurniawan", " "))
11     fmt.Println(strings.ToLower("Eko Kurniawan"))
12     fmt.Println(strings.ToUpper("Eko Kurniawan"))
13     fmt.Println(strings.Trim("    Eko Kurniawan    ", " "))
14     fmt.Println(strings.ReplaceAll("Eko Eko Eko Eko", "Eko", "Budi"))
15 }
16
```

---

# Package strconv

---

# Package strconv

- Sebelumnya kita sudah belajar cara konversi tipe data, misal dari int32 ke int34
- Bagaimana jika kita butuh melakukan konversi yang tipe datanya berbeda? Misal dari int ke string, atau sebaliknya
- Hal tersebut bisa kita lakukan dengan bantuan package strconv (string conversion)
- <https://golang.org/pkg/strconv/>

---

# Beberapa Function di Package strconv

Function	Kegunaan
strconv.parseBool(string)	Mengubah string ke bool
strconv.parseFloat(string)	Mengubah string ke float
strconv.parseInt(string)	Mengubah string ke int64
strconv.FormatBool(bool)	Mengubah bool ke string
strconv.FormatFloat(float, ... )	Mengubah float64 ke string
strconv.FormatInt(int, ... )	Mengubah int64 ke string

---

# Kode Program Package strconv

```
7
8 ►  func main() {
9      boolean, err := strconv.ParseBool("true")
10     if err == nil {
11         fmt.Println(boolean)
12     } else {
13         fmt.Println("Error", err.Error())
14     }
15 }
16 |
```

---

# Package math

---

# Package math

- Package math merupakan package yang berisikan constant dan fungsi matematika
- <https://golang.org/pkg/math/>

---

# Beberapa Function di Package math

Function	Kegunaan
math.Round(float64)	Membulatkan float64 keatas atau kebawah, sesuai dengan yang paling dekat
math.Floor(float64)	Membulatkan float64 kebawah
math.Ceil(float64)	Membulatkan float64 keatas
math.Max(float64, float64)	Mengembalikan nilai float64 paling besar
math.Min(float64, float64)	Mengembalikan nilai float64 paling kecil

---

# Kode Program Package math

```
3  import (
4      "fmt"
5      "math"
6  )
7
8 ▶ func main() {
9     fmt.Println(math.Ceil(1.40))
10    fmt.Println(math.Floor(1.60))
11    fmt.Println(math.Round(1.60))
12    fmt.Println(math.Max(1, 2))
13    fmt.Println(math.Min(1, 2))
```

---

# Package container/list

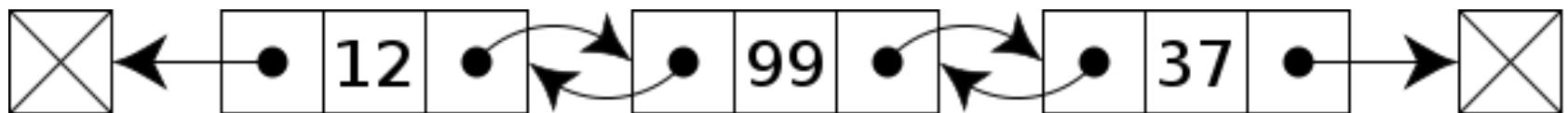
---

# Package container/list

- Package container/list adalah implementasi struktur data double linked list di Go-Lang
- <https://golang.org/pkg/container/list/>

---

## Struktur Data Double Linked List



---

# Kode Program Package container/list

```
8 ►  func main() {
9     data := List.New()
10    data.PushBack("Eko")
11    data.PushBack("Kurniawan")
12    data.PushBack("Khannedy")
13
14    for e := data.Front(); e != nil; e = e.Next() {
15        fmt.Println(e.Value)
16    }
17 }
18 }
```

---

# Package container/ring

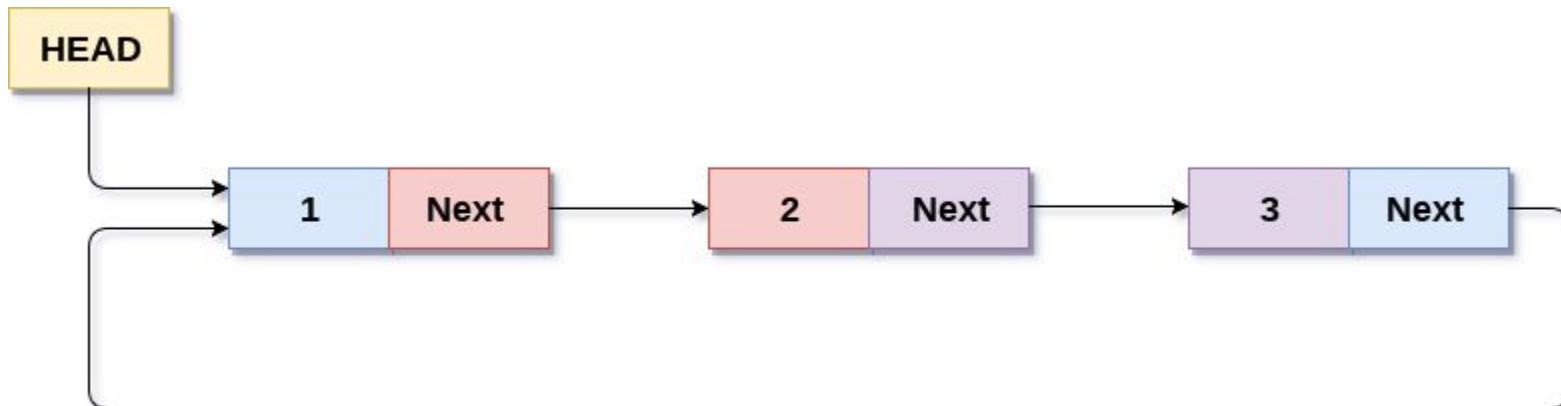
---

# Package container/ring

- Package container/ring adalah implementasi struktur data circular list
- Circular list adalah struktur data ring, dimana diakhir element akan kembali ke element awal (HEAD)
- <https://golang.org/pkg/container/ring/>

---

# Struktur Data Circular List



# Kode Program Package container/ring

```
o
9 ►  func main() {
10    data := ring.New(5)
11    for i := 0; i < data.Len(); i++ {
12        data.Value = "Value " + strconv.FormatInt(int64(i), 10)
13        data = data.Next()
14    }
15
16    data.Do(func(value interface{}) {
17        fmt.Println(value)
18    })
19 }
```

---

# Package sort

---

## Package sort

- Package sort adalah package yang berisikan utilitas untuk proses pengurutan
- Agar data kita bisa diurutkan, kita harus mengimplementasikan kontrak di interface `sort.Interface`
- <https://golang.org/pkg/sort/>

# sort.Interface

```
13     // elements of the collection be enumerated by an integer index.
14 I↓ type Interface interface {
15     // Len is the number of elements in the collection.
16 I↓     Len() int
17     // Less reports whether the element with
18     // index i should sort before the element with index j.
19 I↓     Less(i, j int) bool
20     // Swap swaps the elements with indexes i and j.
21 I↓     Swap(i, j int)
22 }
23 }
```

# Kode Program Package sort (1)

```
2
3     type User struct {
4         Name string
5         Age  int
6     }
7
8 ⚡↑ type UserSlice []User
9
10 ⚡↑ func (userSlice UserSlice) Len() int {
11     return len(userSlice)
12 }
```

## Kode Program Package sort (2)

```
10 ①↑ func (userSlice UserSlice) Len() int {
11     return len(userSlice)
12 }
13
14 ①↑ func (userSlice UserSlice) Less(i, j int) bool {
15     return userSlice[i].Age < userSlice[j].Age
16 }
17
18 ①↑ func (userSlice UserSlice) Swap(i, j int) {
19     userSlice[i], userSlice[j] = userSlice[j], userSlice[i]
20 }
```

---

## Kode Program Package sort (3)

```
27 ► func main() {
28     users := []User{
29         {"Eko", 30},
30         {"Budi", 35},
31         {"Joko", 25},
32         {"Adit", 23},
33     }
34
35     sort.Sort(UserSlice(users))
36
37     fmt.Println(users)
```

---

# Package time



# Package time

- Package time adalah package yang berisikan fungsionalitas untuk management waktu di Go-Lang
- <https://golang.org/pkg/time/>

---

# Beberapa Function di Package time

Function	Kegunaan
time.Now()	Untuk mendapatkan waktu saat ini
time.Date(...)	Untuk membuat waktu
time.Parse(layout, string)	Untuk memarsing waktu dari string



# Kode Program Package time

```
8 ►  func main() {
9      now := time.Now()
10     fmt.Println(now.Local())
11
12     utc := time.Date(2009, time.November, 10, 23, 0, 0, 0, time.UTC)
13     fmt.Println(utc.Local())
14
15     parse, _ := time.Parse(time.RFC3339, "2006-01-02T15:04:05Z")
16     fmt.Println(parse)
17 }
18
```

---

# Package reflect

---

# Package reflect

- Dalam bahasa pemrograman, biasanya ada fitur Reflection, dimana kita bisa melihat struktur kode kita pada saat aplikasi sedang berjalan
- Hal ini bisa dilakukan di Go-Lang dengan menggunakan package reflect
- Fitur ini mungkin tidak bisa dibahas secara lengkap dalam satu video, Anda bisa eksplorasi package reflect ini secara otodidak
- Reflection sangat berguna ketika kita ingin membuat library yang general sehingga mudah digunakan
- <https://golang.org/pkg/reflect/>

---

# Kode Program Package reflect

```
8  type Sample struct {
9      Name string
10 }
11
12 ► func main() {
13     sample := Sample{"Eko"}
14     sampleType := reflect.TypeOf(sample)
15     structField := sampleType.Field(0)
16
17     fmt.Println(structField.Name)
18 }
```

---

# Kode Program StructTag

```
8  type Sample struct {
9      Name string `required:"true" max:"10"`
10 }
11
12 func main() {
13     sample := Sample{"Eko"}
14     sampleType := reflect.TypeOf(sample)
15     structField := sampleType.Field(0)
16     required := structField.Tag.Get("required")
17
18     fmt.Println(required)
```

---

# Kode Program Validation Library

```
24
25     func IsValid(data interface{}) bool {
26         t := reflect.TypeOf(data)
27         for i := 0; i < t.NumField(); i++ {
28             field := t.Field(i)
29             if field.Tag.Get("required") == "true" {
30                 return reflect.ValueOf(data).Field(i).Interface() != ""
31             }
32         }
33     }
34 }
```

---

# Package regexp

---

# Package regexp

- Package regexp adalah utilitas di Go-Lang untuk melakukan pencarian regular expression
- Regular expression di Go-Lang menggunakan library C yang dibuat Google bernama RE2
- <https://github.com/google/re2/wiki/Syntax>
- <https://golang.org/pkg/regexp/>

---

# Beberapa Function di Package regexp

Function	Kegunaan
regexp.MustCompile(string)	Membuat Regexp
Regexp.MatchString(string) bool	Mengecek apakah Regexp match dengan string
Regexp.FindAllString(string, max)	Mencari string yang match dengan maximum jumlah hasil

# Kode Program Package regexp

```
7
8 ►  [-func main() {
9     var regex = regexp.MustCompile(`e([a-z])o`)
10
11    fmt.Println(regex.MatchString("eko"))
12    fmt.Println(regex.MatchString("edo"))
13    fmt.Println(regex.MatchString("eKo"))
14
15    fmt.Println(regex.FindAllString("eko edo egi ego elo eto", 10))
16 }
17 |
```

---

# Materi Selanjutnya?

---

# Materi Selanjutnya?

- Go-Lang Unit Test
- Go-Lang Modules
- Go-Lang Concurrency
- Go-Lang Database
- Go-Lang Web

---

# Eko Kurniawan Khannedy

- Telegram : @khannedy
- Facebook : fb.com/ProgrammerZamanNow
- Instagram : instagram.com/programmerzamannow
- Youtube : youtube.com/c/ProgrammerZamanNow
- Telegram Channel : <https://t.me/ProgrammerZamanNow>
- Email : echo.khannedy@gmail.com