goroutine

goroutine melakukan proses eksekusi pada thread operating sistem dan sangat ringan.

```
package main
import "fmt"
func f(from string) {
    for i := 0; i < 3; i++ {
        fmt.Println(from, ":", i)
    }
}
func main() {
    f("direct")

    go f("goroutine")

    go func(msg string) {
        fmt.Println(msg)
    }("going")
    var input string
    fmt.Scanln(&input)
    fmt.Println("done")
}
```

Dalam contoh ini dibuat fungsi call `f(s)`. Ini adalah cara untuk menjalankan fungsi sehingga berjalan synchronous.

Untuk memanggil fungsi tadi dalam gorouting, gunakan `go f(s)`. goroutine akan melakukan proses eksekusi secara reatime (concurrent).

Juga dapat menggunakan goroutine untuk pemanggilan fungsi yang bersifat anonymous seperti baris berikut,

Kedua fungsi tadi berjalan asyncrounous pada goroutine yang berbeda, baris ini adalah akhir dari proses eksekusi, sehingga ketika anda memasukkan input karakter apapun di keyboard, program akan otomatis menutup. `Scanln digunakan` untuk memasukan input sebelum program ditutup.

Ketika anda jalankan program, anda akan melihat output hasil dari eksekusi fungsi pertama f(“direct”) kemudian output dari kedua fungsi goroutine. Fungsi yang menggunakan “go” berjalan secara reatime. Anda bisa melihat hasilnya, setelah **goroutine 0**, lalu muncul **going** pada output program.

Channel

Channels adalah pipes yang terhubung dengan concurrent pada goroutines. Anda dapat mengirimkan nilai (value) ke channel dari satu goroutine dan memasukkan nilai tersebut ke goroutine yang lain..

```
package main

import "fmt"

func main() {

    messages := make(chan string)

    go func() { messages <- "ping" }()

    msg := <-messages
    fmt.Println(msg)
}
```

Pada baris ini terlihat variable messages membuat channel dengan type string `make(chan val-type)`..

Mengirimkan nilai pada channel dengan menggunakan syntax `<-` . Pada baris ini kita mengirimka kata “ping” ke channel messages dari goroutine baru.

Syntax <- pada channel merepresentasikan bahwa messages memberikan masukan nilai pada variable msg (msg menerima nilai “ping”). Hasil nya lalu di printout ke terminal.

Ketika menjalankan program tersebut pesan ping sukses melewati satu goroutine ke goroutine yang lain melalui channel yang kita buat.

Select

Select dapat dioperasikan pada multi channel, dimana fungsinya mirip seperti *case*. Dapat dikombinasikan dengan goroutine dan channel.

```
package main

import "time"

import "fmt"

func main() {

    channel1 := make(chan string)

    channel2 := make(chan string)

    go func() {

        time.Sleep(time.Second * 4)
```

Sebagai contoh kita membuat 2 channel disini.

Setiap channel akan menerima nilai setelah beberapa waktu yang kita tentukan, dalam contoh ini adalah 2

detik dan 4 detik.

Menggunakan select untuk menunnggu nilai yang sudah kita tentukan tapi diterima secara simultan (bersamaan)

```
        channel1 <- "satu"
    }()
    go func() {
        time.Sleep(time.Second * 2)
        channel2 <- "dua"
    }()
    for i := 0; i < 2; i++ {
        select {
        case msg1 := <-channel1:
            fmt.Println("menerima",
msg1)
        case msg2 := <-channel2:
            fmt.Println("menerima",
msg2)
        }
    }
}
```

Anda akan memperoleh hasil dimana dua akan terlebih dahulu muncul karena terpaut 2 detik dengan satu.

```
$ time go run select.go
received dua
received satu
```