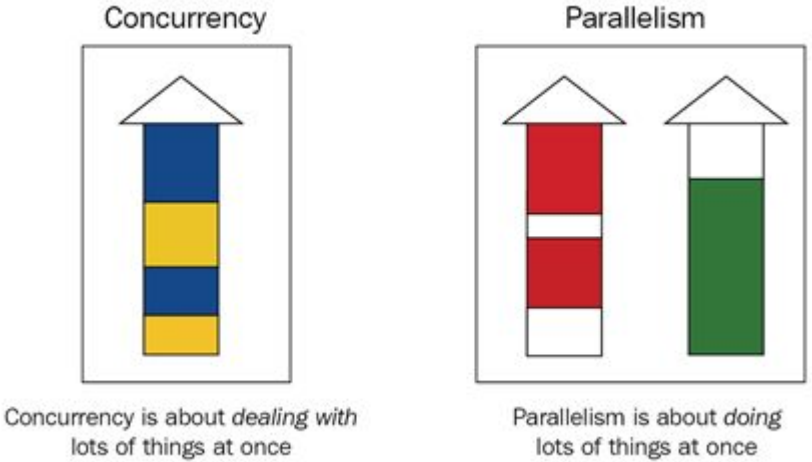
Concurrency - Goroutines

Concurrency



Arti dari concurrency adalah mengeksekusi sebuah proses secara independen atau berhadapan dengan lebih dari satu tugas dalam waktu yang sama. Perlu diingat disini bahwa *concurrency* berbeda dengan parallelism, karena parallelism memiliki arti mengerjakan tugas yang banyak secara bersamaan dari awal hingga akhir. Sedangkan pada *concurrency*, kita tidak akan tahu tentang siapa yang akan menyelesaikan tugasnya terlebih dahulu.

Goroutines

Goroutine merupakan sebuah thread ringan pada bahasa Go untuk melakukan concurrency. Jika dibandingkan dengan thread biasa, *Goroutine* memiliki ukuran thread yang jauh lebih ringan. Pada saat kita mengeksekusi sebuah *Goroutine*, maka satu *Goroutine* hanya membutuhkan 2kb memori saja, sedangkan satu thread biasa dapat menghabiskan 1-2mb memori.

Goroutine bersifat asynchronous sehingga proses nya tidak saling tunggu dengan *Goroutine* lainnya.

Untuk membuat sebuah *Goroutine,* maka kita harus terlebih dahulu membuat sebuah *function.* Lalu ketika kita ingin memanggil *function* tersebut, maka kita perlu menggunakan keyword *go* sebelum kita memanggil *function* tersebut. Contohnya seperti pada gambar dibawah ini.

Bisa dilihat, function bernama *goroutine* dipanggil dengan cara menuliskan keyword *go* terlebih dahulu. Dengan seperti maka maka function *goroutine* secara otomatis menjadi sebuah *Goroutine*.

```go
import "fmt"

func main() {
  go goroutine()
}

func goroutine() {
  fmt.Println("Hello")
}
```

Goroutines (Asynchronous process #1)

Sekarang kita akan mempelajari sifat dari *Goroutine* yang bekerja secara asynchronous. Perhatikan pada gambar disebelah kanan.

Terdapat 2 function bernama *firstProcess* dan *secondProcess*. Kedua *function* tersebut digunakan untuk menampilkan angka dari 1 hingga bilangan yang ditentukan dari parameter yang diterima dengan melakukan looping.

Kemudian pada line 23, function *firstProcess* dijadikan sebagai sebuah *Goroutine* karena dipanggil dengan menggunakan keyword *go*. Lalu pada line 27, kita menggunakan function *NumGoroutine* dari package *runtime* untuk mengetahui jumlah *Goroutine* yang sedang berjalan.

Jika kita jalankan pada terminal kita, maka hasilnya akan seperti pada gambar kedua.

```
20  func main() {
21    fmt.Println("main execution started")
22
23    go firstProcess(8)
24
25    secondProcess(8)
26
27    fmt.Println("No. of Goroutines:", runtime.NumGoroutine())
28
29    fmt.Println("main execution ended")
30
31  }
32
33  func firstProcess(index int) {
34    fmt.Println("First process func started")
35    for i := 1; i <= index; i++ {
36      fmt.Println("i=", i)
37    }
38    fmt.Println("First process func ended")
39  }
40
41  func secondProcess(index int) {
42    fmt.Println("Second process func started")
43    for j := 1; j <= index; j++ {
44      fmt.Println("j=", j)
45    }
46    fmt.Println("Second process func ended")
47  }
48
```

```
main execution started
Second process func started
j= 1
j= 2
j= 3
j= 4
j= 5
j= 6
j= 7
j= 8
Second process func ended
No. of Goroutines: 2
main execution ended
```

Goroutines (Asynchronous process #2)

Jika kita perhatikan hasilnya pada gambar kedua, function *firstProcess* tidak menampilkan hasilnya. Ini terjadi karena setiap *Goroutine* bekerja secara asynchronous dan satu *Goroutine* tidak akan saling tunggu menunggu dengan *Goroutine* lainnya.

Kemudian jika kita perhatikan kembali pada hasilnya, terdapat 2 jumlah *Goroutine* yang sedang berjalan padahal kita hanya menjalankan satu *function* yang dijadikan sebagai sebuah *Goroutine.* Ini terjadi karena faktanya, function *main* juga merupakan sebuah *Goroutine* sehingga function *main* tidak akan menunggu *Goroutine* lainnya selesai tereksekusi. Inilah yang menjadi penyebab function *firstProcess* tidak menampilkan hasilnya walaupun sebetulnya *function* tersebut telah tereksekusi.

```go
func main() {
  fmt.Println("main execution started")

  go firstProcess(8)

  secondProcess(8)

  fmt.Println("No. of Goroutines:", runtime.NumGoroutine())

  fmt.Println("main execution ended")

}

func firstProcess(index int) {
  fmt.Println("First process func started")
  for i := 1; i <= index; i++ {
    fmt.Println("i=", i)
  }
  fmt.Println("First process func ended")
}

func secondProcess(index int) {
  fmt.Println("Second process func started")
  for j := 1; j <= index; j++ {
    fmt.Println("j=", j)
  }
  fmt.Println("Second process func ended")
}
```

```
main execution started
Second process func started
j= 1
j= 2
j= 3
j= 4
j= 5
j= 6
j= 7
j= 8
Second process func ended
No. of Goroutines: 2
main execution ended
```

Goroutines (Asynchronous process #3)

Perlu diingat disini bahwa ketika kita menjalankan sebuah *Goroutine,* maka *Goroutine* tersebut akan membutuhkan waktu yang sedikit lebih lama untuk memulai dibandingkan dengan *function* biasa. Maka dari itu untuk sekarang, kita akan membutuhkan suatu *function* yang akan menahan function *main* untuk langsung menyelesaikan eksekusinya.

Perhatikan pada line 70, kita menggunakan function *Sleep* yang berasal dari package *time.* Lalu kita konstanta bernama *Second* dari package *time* yang merepresentasikan bilangan detik. Kemudian kita kalikan dengan angka 2 agar func *Sleep* mampu menahan function *main* selama 2 detik sebelum function *main* menyelesaikan eksekusinya.

```
53  package main
54
55  import (
56    "fmt"
57    "runtime"
58    "time"
59  )
60
61  func main() {
62    fmt.Println("main execution started")
63
64    go firstProcess(8)
65
66    secondProcess(8)
67
68    fmt.Println("No. of Goroutines:", runtime.NumGoroutine())
69
70    time.Sleep(time.Second * 2)
71
72    fmt.Println("main execution ended")
73
74  }
75
76  func firstProcess(index int) {
77    fmt.Println("First process func started")
78    for i := 1; i <= index; i++ {
79      fmt.Println("i=", i)
80    }
81    fmt.Println("First process func ended")
82  }
83
84  func secondProcess(index int) {
85    fmt.Println("Second process func started")
86    for j := 1; j <= index; j++ {
87      fmt.Println("j=", j)
88    }
89    fmt.Println("Second process func ended")
90  }
91
```

Goroutines (Asynchronous process #4)

Sekarang jika kita jalankan, maka hasilnya akan seperti pada gambar disebelah kanan. Bisa kita lihat bahwa sekarang function *firstProcess* telah menampilkan hasilnya.

```
main execution started
Second process func started
j= 1
j= 2
j= 3
j= 4
j= 5
j= 6
j= 7
j= 8
Second process func ended
No. of Goroutines: 2
First process func started
i= 1
i= 2
i= 3
i= 4
i= 5
i= 6
i= 7
i= 8
First process func ended
main execution ended
```