

Goroutines



01

About

02

Example

03

Closures

Goroutines

- | Goroutines allow functions to run concurrently
 - | Can also run function literals / closures
- | Go will automatically select parallel or asynchronous execution
- | New goroutines can be created with the `go` keyword

Example - Basic

```
func count(amount int) {  
    for i := 1; i <= amount; i++ {  
        time.Sleep(100 * time.Millisecond)  
        fmt.Println(i)  
    }  
}  
  
func main() {  
    go count(5)  
    fmt.Println("wait for goroutine")  
    time.Sleep(1000 * time.Millisecond)  
    fmt.Println("end program")  
}
```

```
> go run ./lecture.go  
wait for goroutine  
1  
2  
3  
4  
5  
end program
```

Example - Closures

```
counter := 0

wait := func(ms time.Duration) {
    time.Sleep(ms * time.Millisecond)
    counter += 1
}

fmt.Println("Launching goroutines")
go wait(100)
go wait(900)
go wait(1000)

fmt.Println("Launched.      Counter =", counter)
time.Sleep(1100 * time.Millisecond)
fmt.Println("Waited 1100ms. Counter =", counter)
```

```
> go run ./lecture.go
Launching goroutines
Launched.      Counter = 0
Waited 1100ms. Counter = 3
```

Recap

- | Goroutines allow functions & closures to run concurrently
- | Use the `go` keyword to create a new **goroutine**
- | The function that starts a goroutine will **not** wait for it to finish
 - | Both the calling function and goroutine will run to completion
- | Closure captures are shared among all goroutines
 - | Easy to parallelize code