

Go Channel



Communicating sequential processes(CSP)



golang : Channel

create main.go in folder chapter13-9 :

```
func main() {  
    array := []int{7, 2, 8, -9, 4, 0}  
  
    ch := make(chan int)  
  
    go sum(array[:len(array)/2], ch)  
    go sum(array[len(array)/2:], ch)  
    x := <-ch  
    y := <-ch  
    fmt.Println(x, y, x+y)  
}  
  
func sum(array []int, ch chan int) {  
    sum := 0  
    for _, value := range array {  
        sum += value  
    }  
    ch <- sum  
}
```

run -> no error -> push to your git repository



golang : Unbuffered Channel

create main.go in folder chapter13-10 :

```
func main() {  
    c := make(chan int)  
    //c <- 1  
    //c <- 2  
    go func() { c <- 1 }()  
    //go func() { c <- 2 }()  
    fmt.Println(<-c)  
    //fmt.Println(<-c)  
}
```

A send operation on an unbuffered channel blocks the sending goroutine until another goroutine executes a corresponding receive on the same channel, at which point the value is transmitted and both goroutines may continue.

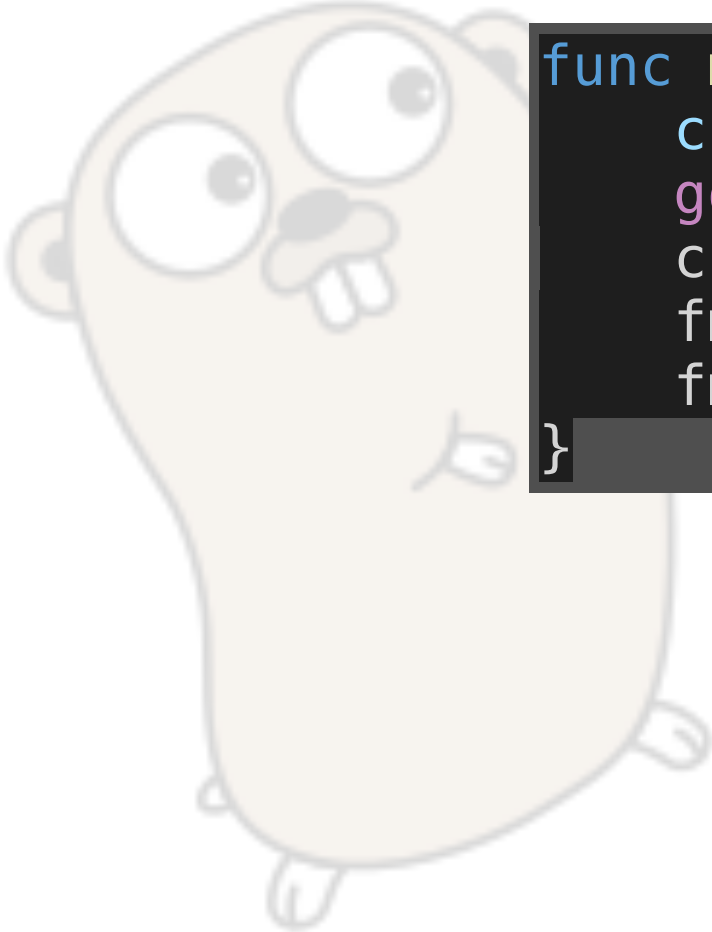
Brian W. Kernighan. "The Go Programming Language (Addison-Wesley Professional Computing Series)

run -> no error -> push to your git repository



golang : Buffered Channel

create main.go in folder chapter13-11 :



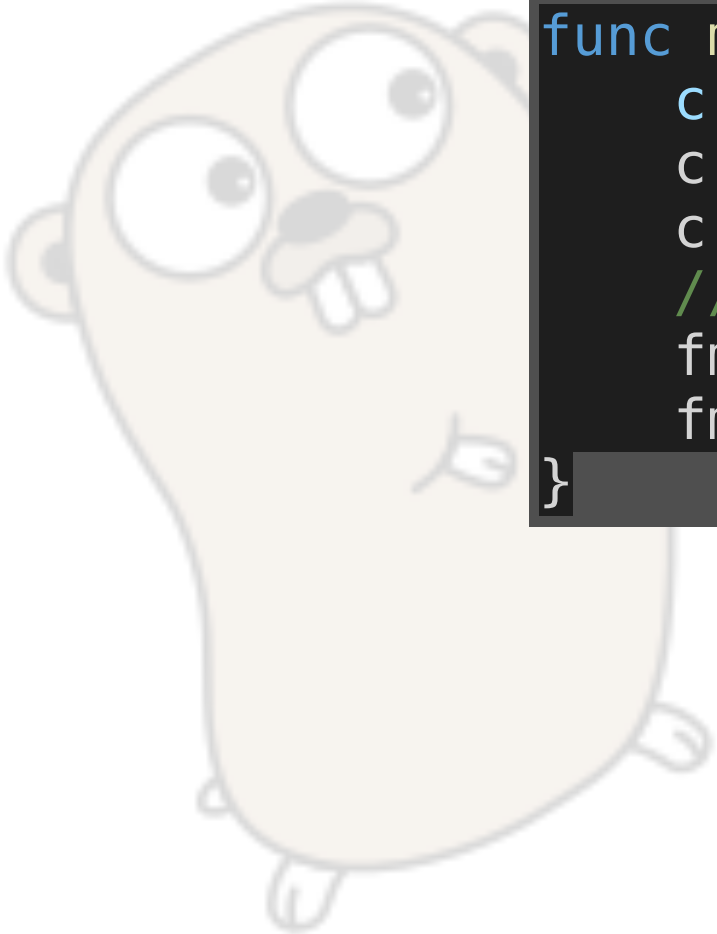
```
func main() {  
    ch := make(chan int, 3)  
    go func() { ch <- 1 }()  
    ch <- 2  
    fmt.Println("cap:", cap(ch))  
    fmt.Println("len:", len(ch))  
}
```

run -> no error -> push to your git repository



golang : Buffered Overfilled

create main.go in folder chapter13-12 :



```
func main() {  
    c := make(chan int, 2)  
    c <- 1  
    c <- 2  
    // c <- 3 // overfilled the buffer  
    fmt.Println(<-c)  
    fmt.Println(<-c)  
}
```

run -> no error -> push to your git repository



golang : Goroutine Leak

create main.go in folder chapter13-13 :

```
func main() {  
    c := make(chan int)  
    go func() { c <- 1 }()  
    go func() { c <- 2 }()  
    go func() { c <- 3 }()  
    fmt.Println(<-c)  
    fmt.Println(<-c)  
}
```

Fix by use buffered channel

```
func main() {  
    c := make(chan int, 2)  
    go func() { c <- 1 }()  
    go func() { c <- 2 }()  
    go func() { c <- 3 }()  
    fmt.Println(<-c)  
    fmt.Println(<-c)  
}
```

run -> no error -> push to your git repository



golang : Channel

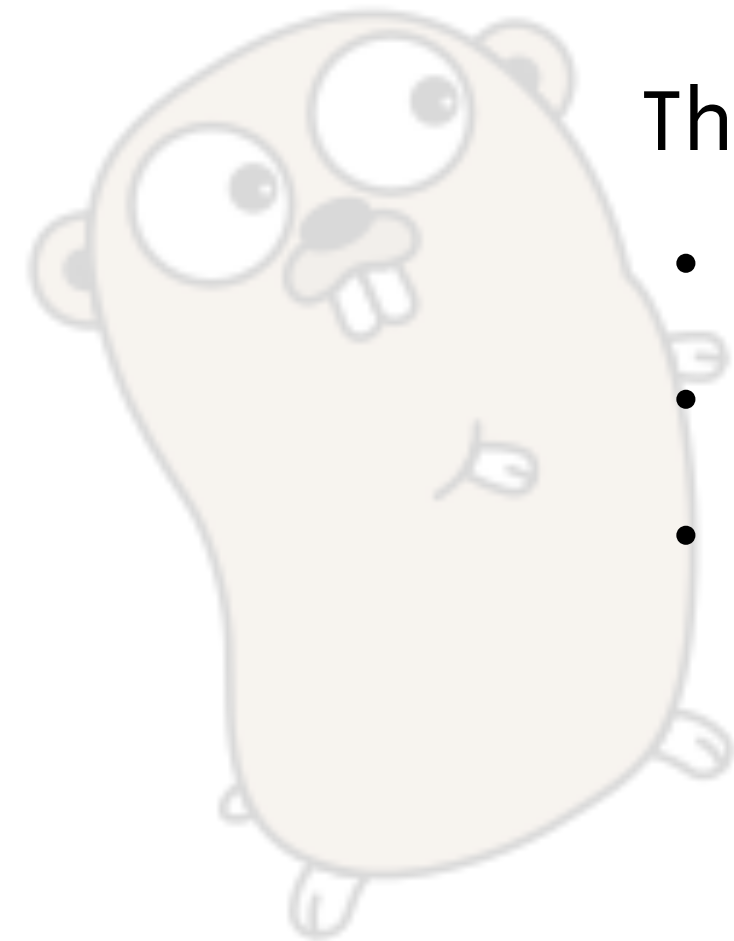
Three channel principal operations

- send
- receive
- close

```
channel <- value
```

```
<-channel
```

```
close(channel)
```



golang : Channel-Direction

create main.go in folder chapter13-14 :


```
func main() {  
    pings := make(chan string, 1)  
    pongs := make(chan string, 1)  
    ping(pings, "passed message")  
    pong(pings, pongs)  
    fmt.Println(<-pongs)  
}  
  
func ping(pings chan<- string, msg string) {  
    pings <- msg  
}  
  
func pong(pings <-chan string, pongs chan<- string) {  
    msg := <-pings // receive  
    pongs <- msg // send  
}
```

run -> no error -> push to your git repository



golang : Channel-Direction

create main.go in folder chapter13-15 :



```
func main() {  
    ch := make(chan int, 10)  
    go fibonacci(cap(ch), ch)  
  
    for i := range ch {  
        fmt.Println(i)  
    }  
}  
  
func fibonacci(n int, ch chan int) {  
    x, y := 0, 1  
    for i := 0; i < n; i++ {  
        ch <- x  
        x, y = y, x+y  
    }  
    close(ch)  
}
```

run -> no error -> push to your git repository



golang : Channel-Select

create main.go in folder chapter13-16 :

```
func main() {  
    ch := make(chan int)  
    quit := make(chan int)  
    go func() {  
        for i := 0; i < 10; i++ {  
            fmt.Println(<-ch)  
        }  
        quit <- 0  
    }()  
    fibonacci(ch, quit)  
}
```


```
func fibonacci(ch, quit chan int) {  
    x, y := 0, 1  
    for {  
        select {  
        case ch <- x:  
            x, y = y, x+y  
        case <-quit:  
            fmt.Println("quit")  
            return  
        }  
    }  
}
```

run -> no error -> push to your git repository



golang : Select with Timeout

create main.go in folder chapter13-20 :



```
func main() {  
    c1 := make(chan string, 1)  
    go func() {  
        time.Sleep(2 * time.Second)  
        c1 <- "result 1"  
    }()  
  
    select {  
    case res := <-c1:  
        fmt.Println(res)  
    case <-time.After(1 * time.Second):  
        fmt.Println("timeout 1")  
    }  
}
```

run -> no error -> push to your git repository



golang : Pipeline #1

create main.go in folder chapter13-17 :

```
func main() {  
    naturals := make(chan int)  
    squares := make(chan int)  
  
    go func() {  
        for x := 0; ; x++ {  
            naturals <- x  
        }  
    }()  
  
    go func() {  
        for {  
            x := <-naturals  
            squares <- x * x  
        }  
    }()  
  
    for {  
        fmt.Println(<-squares)  
    }  
}
```

run -> no error -> push to your git repository



golang : Pipeline #2

create main.go in folder chapter13-18 :

```
func main() {  
    naturals := make(chan int)  
    squares := make(chan int)  
  
    go func() {  
        for x := 0; x < 100; x++ {  
            naturals <- x  
        }  
        close(naturals)  
    }()  
    go func() {  
        for x := range naturals {  
            squares <- x * x  
        }  
        close(squares)  
    }()  
  
    for x := range squares {  
        fmt.Println(x)  
    }  
}
```

run -> no error -> push to your git repository



golang : Pipeline #3

create main.go in folder chapter13-19 :

```
func main() {  
    naturals := make(chan int)  
    squares := make(chan int)  
  
    go counter(naturals)  
    go squarer(naturals, squares)  
    print(squares)  
}
```

```
func counter(out chan<- int) {  
    for x := 0; x < 100; x++ {  
        out <- x  
    }  
    close(out)  
}  
  
func squarer(in <-chan int, out chan<- int) {  
    for x := range in {  
        out <- x * x  
    }  
    close(out)  
}  
  
func print(in <-chan int) {  
    for x := range in {  
        fmt.Println(x)  
    }  
}
```

run -> no error -> push to your git repository

