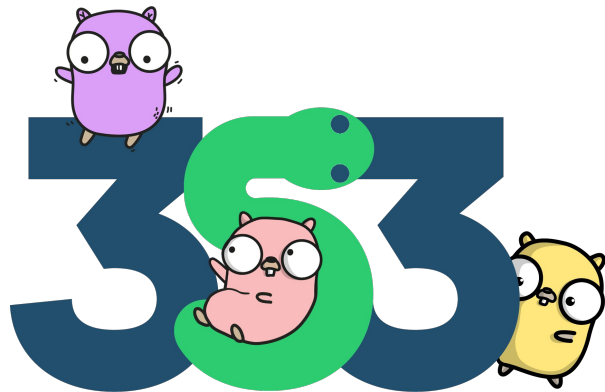


# In the Loop

miki @tebeka

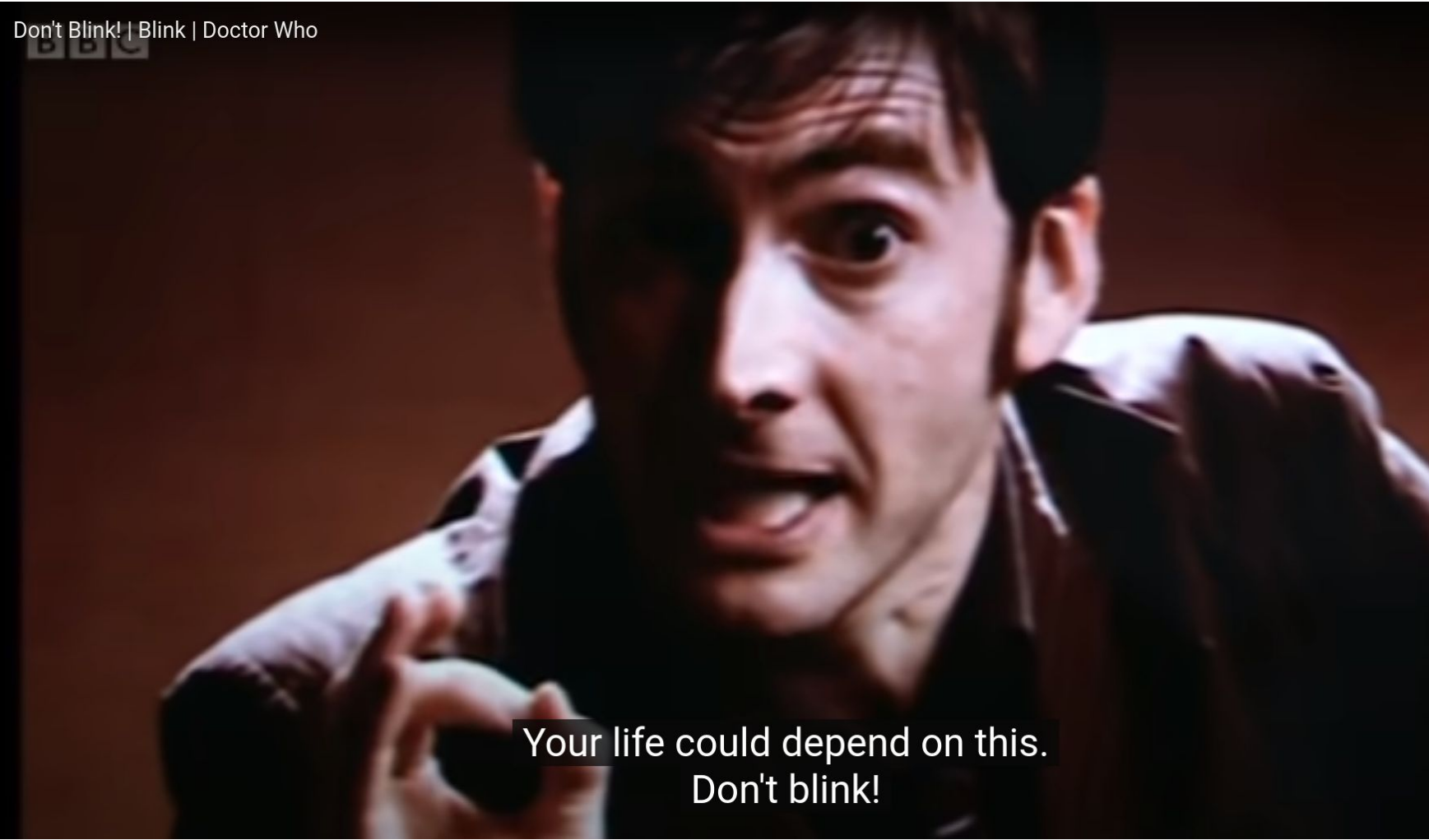
miki@353solutions.com

CEO, CTO, UFO ...



Don't Blink! | Blink | Doctor Who

BBC



Your life could depend on this.  
Don't blink!

simple

```
fmt.Println(1 << 1)
```

euler\_1.go

## the boring one

```
total := 0
➤ for i := 0; i < 1000; i++ {
    if i%3 == 0 || i%5 == 0 {
        total += i
    }
}

fmt.Println(total)
```

## two variables

```
func isPalindrome(s string) bool {  
    chars := []rune(s)  
    ➤    for i, j := 0, len(chars)-1; i < j; i, j = i+1, j-1 {  
        if chars[i] != chars[j] {  
            return false  
        }  
    }  
  
    return true  
}
```

# while

```
total, a, b := 0, 1, 1
➤  for a <= 4_000_000 {
    if a%2 == 0 {
        total += a
    }
    a, b = b, a+b
}

fmt.Println(total)
```

# forever

```
➤ func handler(p Provider) {  
    for {  
        msg := p.Next()  
        if msg == nil {  
            break  
        }  
  
        fmt.Printf("%+v\n", msg)  
    }  
}
```

handler.go

## range ∴ slice

```
    cart := []string{"bread", "butter", "beer"}  
    // indices  
➤    for i := range cart {  
        fmt.Println(i)  
    }  
    // index + value  
➤    for i, v := range cart {  
        fmt.Println(i, v)  
    }  
    // values  
➤    for _, v := range cart {  
        fmt.Println(v)  
    }
```



## range ∴ value semantics

```
players := []Player{  
    {"Rick", 1_000_000},  
    {"Morty", 13},  
}
```

```
➤ for _, player := range players {  
    player.points += 353  
}  
fmt.Printf("%v\n", players)  
// [{Rick 1000000} {Morty 13}]
```

## range ∴ pointer(ish) semantics

```
players := []Player{
    {"Rick", 1_000_000},
    {"Morty", 13},
}
```

```
➤ for i := range players {
    players[i].Points += 353
}
fmt.Printf("%v\n", players)
// [{Rick 1000353} {Morty 366}]
```

scores\_ref.go

`range ∴ map`

Same as slices 😊

`s/index/key/`

## range ∴ channel

```
ch := make(chan int)
go func() {
    for i := 0; i < 3; i++ {
        ch <- i
    }
    close(ch)
}()
```

```
➤ for v := range ch {
    fmt.Println(v)
}
```

## range ∴ nothing

```
// fan out
ch := make(chan Result)
for _, url := range urls {
    url := url
    go func() {
        ch <- Result{url, checkURL(url)}
    }()
}
// collect
➤ for range urls {
    r := <-ch
    fmt.Printf("%s: %v\n", r.URL, r.Err)
}
```

urls.go

closure ∴ loop variable url captured by func literal

```
// fan out
ch := make(chan Result)
for _, url := range urls {
    url := url
    go func() {
        ch <- Result{url, checkURL(url)}
    }()
}
```

See [redefining for loop variable semantics](#) by Russ Cox.

urls.go

# nested

```
found := false
for r := range mat {
    for c := range mat[0] {
        if v := mat[r][c]; v < 0 {
            found = true
            fmt.Println("found", v)
            break
        }
    }
}
fmt.Println("negatives:", found)
```



## nested ∴ fix

```
found := false
```

**loop:**

```
for r := range mat {
```

```
    for c := range mat[0] {
```

```
        if v := mat[r][c]; v < 0 {
```

```
            found = true
```

```
            fmt.Println("found", v)
```

```
            break loop
```

```
        }
```

```
    }
```

```
}
```

```
fmt.Println("negatives:", found)
```

nested\_label.go



# goto



<https://xkcd.com/292/>

# standard library

```
$ ag --vimgrep -s --go 'goto\s+' ~/sdk/go1.19.5/src | \
  grep -v testdata | \
  grep -v _test.go | \
  wc -l
```

**610**

## nested ∴ goto

```
found := false
for r := range mat {
    for c := range mat[0] {
        if v := mat[r][c]; v < 0 {
            found = true
            fmt.Println("found", v)
            goto end
        }
    }
}
```

**end:**

```
fmt.Println("has even:", found)
```

nested\_goto.go

## iteration ∴ scanner

```
lnum := 0
s := bufio.NewScanner(r)
➤ for s.Scan() {
    lnum++
    ➤ if strings.Contains(s.Text(), term) {
        fmt.Printf("%d: %s\n", lnum, s.Text())
    }
}
➤ if err := s.Err(); err != nil {
    log.Fatalf("error: %s", err)
}
```

See [discussion: standard iterator interface](#) by Ian Lance Taylor  
fgrep.go

# Thank You

miki @tebeka

miki@353solutions.com



# In the Loop

miki @tebeka

miki@353solutions.com

CEO, CTO, UFO ...



```
runtime: goroutine stack exceeds 1000000000-byte limit
runtime: sp=0xc020160398 stack=[0xc020160000, 0xc040160000]
fatal error: stack overflow
```

```
runtime stack:
runtime.throw({0x46581a?, 0x4bc880?})
    /.../panic.go:1047 +0x5d fp=0x7ffef80d78d0 sp=0x7ffef80d78a0 pc=0x42d91d
runtime.newstack()
    /.../stack.go:1103 +0x5cc fp=0x7ffef80d7a88 sp=0x7ffef80d78d0 pc=0x444bcc
runtime.morestack()
    /.../asm_amd64.s:570 +0x8b fp=0x7ffef80d7a90 sp=0x7ffef80d7a88 pc=0x45496b
```

```
goroutine 1 [running]:
main.loop(0x1555528?)
    /.../loop.go:3 +0x30 fp=0xc0201603a8 sp=0xc0201603a0 pc=0x457c70
main.loop(0x0?)
    /.../loop.go:4 +0x1c fp=0xc0201603c0 sp=0xc0201603a8 pc=0x457c5c
main.loop(0x0?)
    /.../loop.go:4 +0x1c fp=0xc0201603d8 sp=0xc0201603c0 pc=0x457c5c
main.loop(0x0?)
    /.../loop.go:4 +0x1c fp=0xc0201603f0 sp=0xc0201603d8 pc=0x457c5c
main.loop(0x0?)
    /.../loop.go:4 +0x1c fp=0xc020160408 sp=0xc0201603f0 pc=0x457c5c
main.loop(0x0?)
    /.../loop.go:4 +0x1c fp=0xc020160420 sp=0xc020160408 pc=0x457c5c
```