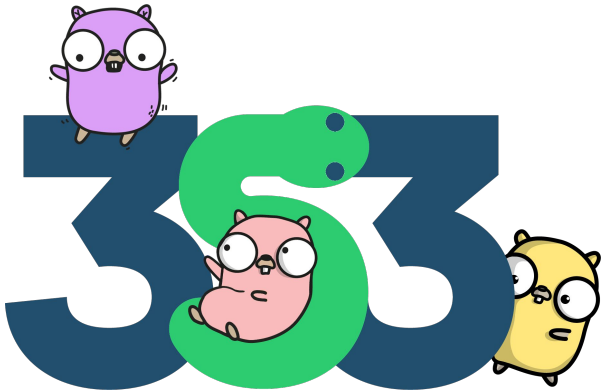
In the Loop

miki @tebeka
miki@353solutions.com

CEO, CTO, UFO ...



the boring one

```
total := 0
for i := 0; i < 1000; i++ {
    if i%3 == 0 || i%5 == 0 {
        total += i
    }
}
fmt.Println(total)
```

euler_1.go

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
}
```

pali.go

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)
```

euler_2.go

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)
    }
```

slice.go

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 ☺️

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

```
    // 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)
```

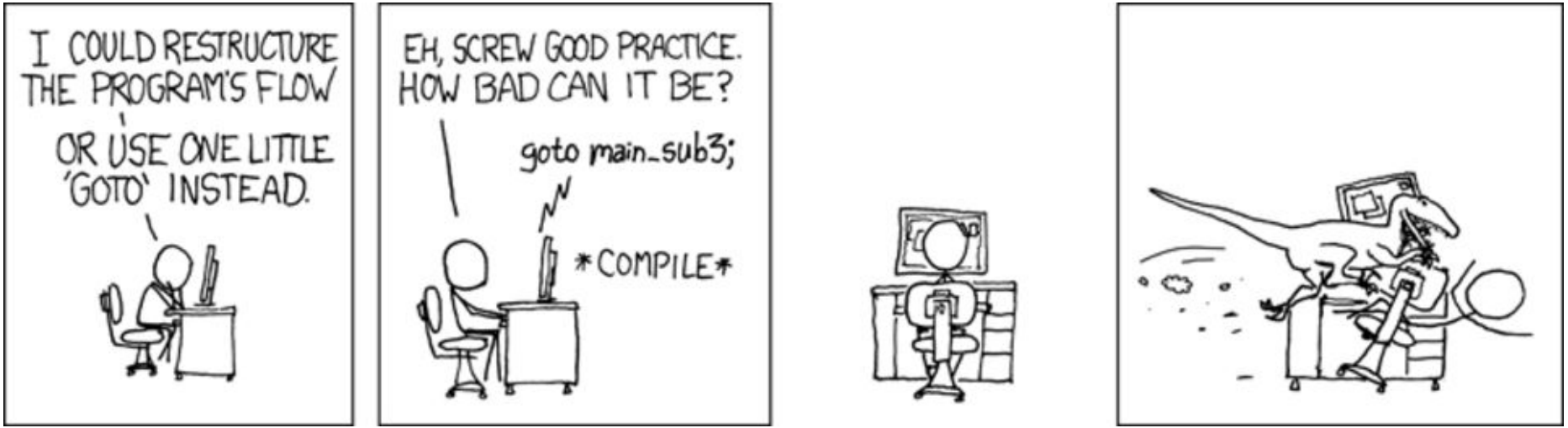
nested.go

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

Thank You

miki @tebeka
miki@353solutions.com

In the Loop

miki @tebeka
miki@353solutions.com

CEO, CTO, UFO ...