

# Let's Go!

Mansi Sharma

January 20, 2021



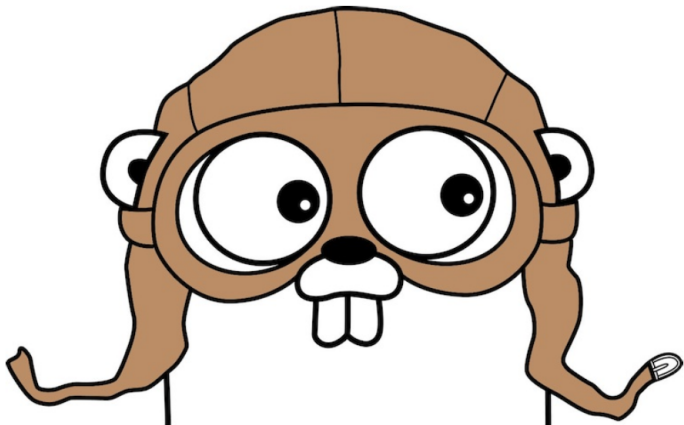
# What is Go?

Go is a **compiled**, **concurrent**, **garbage-collected**,  
**statically typed** language developed at



*GO*

see a bit of code!



# Hello, World!

- Programs start running in package main.
- It is good style to use the factored import statement.
- A name is exported if it begins with a capital letter.

```
1 package main
2
3 import (
4     "fmt"
5     "math"
6 )
7
8 // import "fmt"
9 // import "math"
10
11 func main() {
12     fmt.Printf("Now you have %v problems.\n", math.Sqrt(7))
13 }
14
```

# Variables

- The var instruction declares a list of variables
- The type is informed at the end
- The var instruction could includes initializers, 1 per variable. In this case, the type could be ommited

```
$ go run variables.go  
0 false false false
```

variables.go

```
1 package main  
2  
3 import "fmt"  
4  
5 var c, python, java bool  
6  
7 func main() {  
8     var i int  
9     fmt.Println(i, c, python, java)  
10 }
```

```
$ go run variables-with-initiali  
1 2 true false no!
```

variables-with-initializers.go

```
1 package main  
2  
3 import "fmt"  
4  
5 var i, j int = 1, 2  
6  
7 func main() {  
8     var c, python, java = true, false, "no!"  
9     fmt.Println(i, j, c, python, java)  
10 }
```

# Short Variable Declarations

- Inside a function, the short attribution instruction `:=` can be used instead of a `var` declaration

```
$ go run short-variable-declarations.go  
1 2 3 true false no!
```

short-variable-declarations.go

```
1 package main  
2  
3 import "fmt"  
4  
5 i, j := 1, 2  
6 func main() {  
7     k := 3  
8     c, python, java := true, false, "no!"  
9  
10    fmt.Println(i, j, k, c, python, java)  
11 }  
12
```

# Constants

- Constants are declared like variables but with keyword `const`
- Can not use the syntax `:=`

```
$ go run constants.go  
Hello world! Happy 3.14 Day! Go rules?
```

```
constants.go  
1 package main  
2  
3 import "fmt"  
4 const Pi = 3.14  
5  
6 func main() {  
7     const World = "world! "  
8     fmt.Print("Hello ", World)  
9     fmt.Print("Happy ", Pi, " Day! ")  
10  
11     const Truth = true  
12     fmt.Print("Go rules? ", Truth)  
13 }
```

# Functions

- Type comes after the parameter name, like variables
- Shorten (x int, y int) to (x, y int)

`func Hello(name string) string`

Diagram illustrating the components of a Go function signature:

- `func`: Function keyword
- `Hello`: Function name
- `(name string)`: Parameter list (parameter name followed by parameter type)
- `string`: Return type

```
$ go run functions.go
55
```

functions.go

```
1 package main
2
3 import "fmt"
4
5 func add(x int, y int) int {
6     return x + y
7 }
8
9 func main() {
10     fmt.Println(add(42, 13))
11 }
12
```



# Multiple Return Values

- A function can have multiple return values

```
$ go run multiple-results.go  
world hello
```

multiple-results.go

```
1 package main  
2  
3 import "fmt"  
4  
5 func swap(x, y string) (string, string) {  
6     return y, x  
7 }  
8  
9 func main() {  
10     a, b := swap("hello", "world")  
11     fmt.Println(a, b)  
12 }  
13
```

# Looping For

- Go has only one looping construct, the for loop
- No parentheses required, braces are always required
- The init and post statements are optional

```
$ go run for.go
45
```

for.go

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     sum := 0
7     for i := 0; i < 10; i++ {
8         sum += i
9     }
10    fmt.Println(sum)
11 }
```

```
$ go run for-continuu
1024
```

for-continued.go

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     sum := 1
7     for ; sum < 1000; {
8         sum += sum
9     }
10    fmt.Println(sum)
11 }
```

# For is Go's "while" and forever

- Semicolon can be removed and you will have while
- for can run forever

```
$ go run for-is-go-while.go  
1024
```

for-is-gos-while.go

```
1 package main  
2  
3 import "fmt"  
4  
5 func main() {  
6     sum := 1  
7     for sum < 1000 {  
8         sum += sum  
9     }  
10    fmt.Println(sum)  
11 }
```

```
$ go run forever.go  
process took too long
```

forever.go

```
1 package main  
2  
3 func main() {  
4     for {  
5     }  
6 }
```

# if Condition

- No parentheses required, braces are always required

if.go

```
1 package main
2
3 import (
4     "fmt"
5     "math"
6 )
7
8 func sqrt(x float64) string {
9     if x < 0 {
10         return sqrt(-x) + "i"
11     }
12     return fmt.Sprintf(math.Sqrt(x))
13 }
14
15 func main() {
16     fmt.Println(sqrt(2), sqrt(-4))
17 }
18
```

if-with-a-short-statement.go

```
1 package main
2
3 import (
4     "fmt"
5     "math"
6 )
7
8 func pow(x, n, lim float64) float64 {
9     if v := math.Pow(x, n); v < lim {
10         return v
11     }
12     return lim
13 }
14
15 func main() {
16     fmt.Println(
17         pow(3, 2, 10),
18         pow(3, 3, 20),
19     )
20 }
21
```

# Switch

- only runs the selected case, not all the cases that follow
- break statement is not required

switch-evaluation-order.go

```
1 package main
2
3 import (
4     "fmt"
5     "time"
6 )
7
8 func main() {
9     fmt.Println("When's Saturday?")
10    today := time.Now().Weekday()
11    switch time.Saturday {
12    case today + 0:
13        fmt.Println("Today.")
14    case today + 1:
15        fmt.Println("Tomorrow.")
16    case today + 2:
17        fmt.Println("In two days.")
18    default:
19        fmt.Println("Too far away.")
20    }
21 }
22
```

switch-with-no-condition.go

```
1 package main
2
3 import (
4     "fmt"
5     "time"
6 )
7
8 func main() {
9     t := time.Now()
10    switch {
11    case t.Hour() < 12:
12        fmt.Println("Good morning!")
13    case t.Hour() < 17:
14        fmt.Println("Good afternoon.")
15    default:
16        fmt.Println("Good evening.")
17    }
18 }
19
```

# Defer

- Semicolon can be removed and you will have while
- for can run forever

```
$ go run for-is-go-while.go  
1024
```

for-is-gos-while.go

```
1 package main  
2  
3 import "fmt"  
4  
5 func main() {  
6     sum := 1  
7     for sum < 1000 {  
8         sum += sum  
9     }  
10    fmt.Println(sum)  
11 }
```

```
$ go run forever.go  
process took too long
```

forever.go

```
1 package main  
2  
3 func main() {  
4     for {  
5     }  
6 }
```

# Stacking Defer

- Deferred function calls are pushed onto a stack

defer-multi.go

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Println("counting")
7
8     for i := 0; i < 10; i++ {
9         defer fmt.Println(i)
10    }
11
12    fmt.Println("done")
13 }
14
```

```
counting
done
9
8
7
6
5
4
3
2
1
0
```

# Questions?

