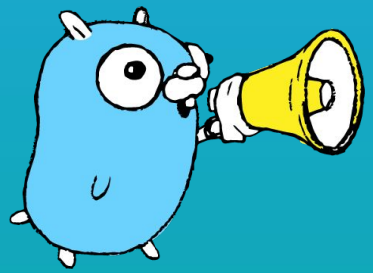
Devfest Cardiff 2019

Zero to Gopher

J Gregory

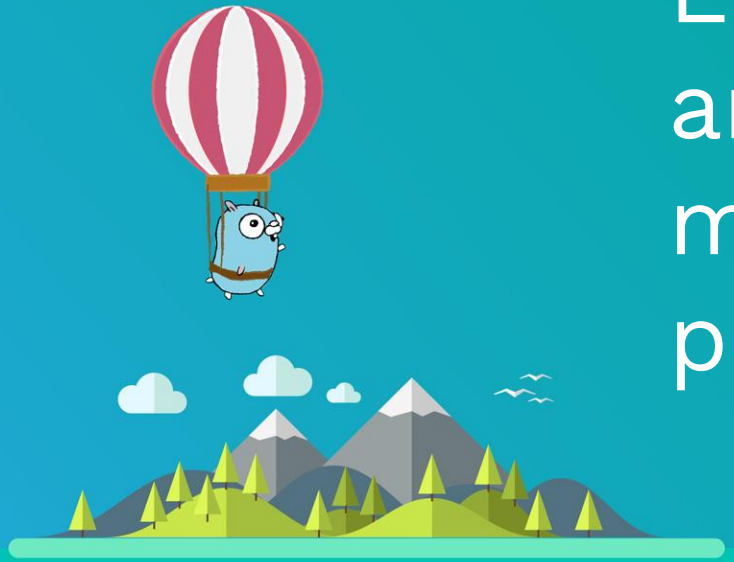
Cardiff Go

@n3crophonic



Let’s Go!

History and overview



“Eliminate the slowness and clumsiness … to make the process more productive and scalable”

Rob Pike - SPLASH 2012

Zero to Gopher

A lightning language tour

“

25 keywords

const func import package type var

chan interface map struct

select case default goto switch break fallthrough if else for range continue return

defer go

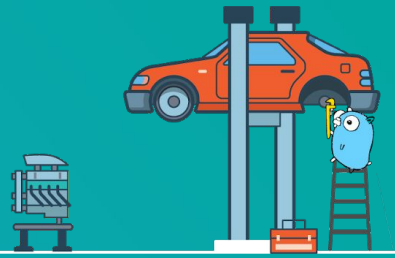
“

19 basic types

```
bool string
byte // alias for uint8
int  int8  int16  int32  int64
uint uint8 uint16 uint32 uint64 uintptr
rune // represents a Unicode code point
float32 float64 complex64 complex128
```

“

6 composite types + 3 containers



pointers, structs

functions, channels, interfaces

containers
array, map, slice

Cool stuff

No semicolons

Well mostly - except for, well *for*

```
package main

import “fmt”

func main() {

  var answer int = 42
  fmt.Println(“The answer is”,answer)

  for i:=0 ; i<10 ; i++ {
    fmt.Println(“Number!: %d”, i)
  }

  // … other code
}
```

Say goodbye to “missing semicolon” errors!

Privacy by convention

Access restriction by letter casing

```
package awesome

func LoudElephant() {
  fmt.Println(“LOUD TRUMPET!”)
}

func silentNinja() {
  fmt.Println(“only you see me”)
}
```

```
// … somewhere in a different package

awesome.LoudElephant() // Trumpet!
awesome.silentNinja()  // Compile error!
```

Now You See It

now you don’t

Cross compilation

All platforms, one compiler

```
// Set the OS and architecture to
// build for as environment vars
// passed to go build
//
// Some examples:

$> GOOS=linux GOARCH=amd64 go build

$> GOOS=plan9 GOARCH=386 go build

$> GOOS=darwin GOARCH=amd64 go build

$> GOOS=windows GOARCH=amd64 go build
```

Build
anywhere,
for anywhere!

Errors, not exceptions

You don't need to be exceptional

```
func doThings() error {

  f, err := os.Create("afile")

  if err != nil {
    log.Println("Error!:",err)
    return err
  }

  // … more awesome code
  //

  return nil // no error!
}
```

Errors are always handled, not thrown

Statically, not noisily, typed

Go can infer common types for you

```
package main

func main() {

  // Explicitly declare and assign
  var name string = “Jo”

  // Let Go do the heavy lifting!
  pet := “capybara”       // string
  age := 25               // int
  score := 32.4           // float64
  science := 0.67 + 0.5i  // complex128

  // … other code
}
```

Go is statically typed, but you don’t always have to be explicit!

The compiler is on your side

If you do something that looks wrong, the code will not compile

```
package main

func main() {

    x := 42

}
```

```
$> go run main.go

./prog.go:5:2: x declared and not used
```

Go will try hard to not let you do silly things!

Only one loop - for

Simple looping with a single construct

```
for i := 0 ; i < 10 ; i++ {
  // do stuff …
}
```

classic for loop

```
for i < 10 {
  // do stuff …
}
```

while loop

```
for {
  // do stuff …
}
```

infinite loop

```
for i, v := range sliceOrMap {
  // do stuff …
}
```

range loop

One loop - all loops!

Ensure stuff happens with defer()

Let Go remember for you

```
func readFile(name string) {

  f, err := os.Create(name)
  if err != nil {
    panic(err)
  }
  defer f.Close()

  // .. do cool stuff with the file

  // .. don't need to remember to close file
  //    defer will take care of it as it
  //    drops out of scope!
  return
}
```

Never forget to clean up!

One formatter, one style

go fmt takes the arguments away

```
$> go fmt ./...
```

```
package   main

func main()  {

    name := “Jo”
  x,y   := 1, 2

  // other code …

  }
```



```
package main

func main() {

  name := “Jo”
  x, y := 1, 2

  // other code …

}
```

No more arguments about curly brackets!

Concurrency with Goroutines

Go is concurrent by design

```
func main() {
  worker()
  worker()

  // … more awesomeness
}
```

```
func main() {
  go worker()
  go worker()

  // … more awesomeness
}
```

Easy and safe scalability

Channels

Safe concurrent data with channels

```
func main() {
  c := make(chan int, 1)
  go addOne(41, c)
  result := <- c
  fmt.Println("Result was:", result)
}

func addOne(i int, c chan int) {
  c <- i + 1
}
```

```
$> go run main.go

Result was 42
```

No more mutexes!

Slices

Windows to the soul of your arrays

```
func sliceIt() {
  s := make([]byte, 2)

  s[0] = ‘c’
  s[1] = ‘a’
  fmt.Println(s)

  s = append(s, ‘t’)
  fmt.Println(s)
}
```

```
$> go run main.go
ca
cat
```

Slice it any way you like

Quick FAQ

Short answer is Yes and No

- Structs can have attributes and behaviour

  However

- There is no type hierarchy

Why doesn't Go have generics?

To reduce complexity

- Generics come at cost of complexity in the type system and runtime
- Go is about maximising familiarity and ease of use

Should I use Go for everything?

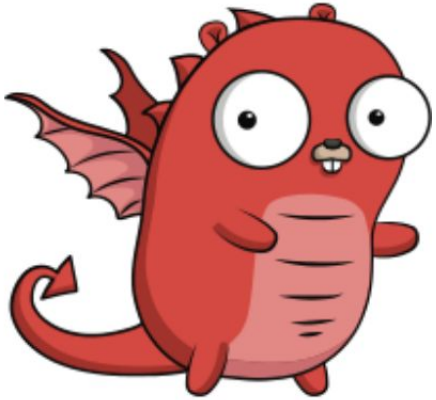
You can... but...

- All languages have strengths and weaknesses
- Go excels at microservices and web servers and tools

However

- Other languages are stronger in specific areas

A quick word about Cardiff Go!



Cardiff Go

Twitter: @cdfgolang

Github: github.com/golang-cymru

Web: https://golang.cymru

Meetup: /Cardiff-Go-Meetup

Thanks for listening!

J Gregory

Twitter: @n3crophonic

Github: github.com/necrophonic

Instagram: cafpanda

Any Questions?