GO

Programming Language

fhLUG Go Workshop
PAPESH Konstantin
Hagenberg, 2023-04-17

What’s on today's dinner menu?

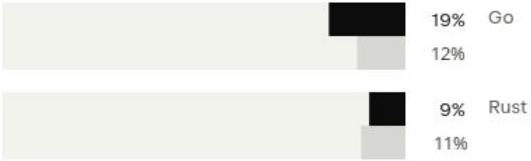
- What is Go?
- Where is it used?
- What makes Go different?
- A code example
- How to get started

What is Go?

- Functional
- Imperative
- Statically typed
- Compiled
- Without VM
- Garbage collected

But where is it used?

- Docker
- Kubernetes
- Traefik
- Caddy
- Gogs
- Hugo
- Matrix-dendrite



© Jetbrains Developer Survey 2022

What makes it different?

No semicolons and no parentheses

```
for i := 0; i < 10; i++ {
    count++
}
```

Type inference and types after names

```go
var a = 1
c := 12
var b float64 = 1.2
```

Pointers!

```
a Thingy := getThingy()
ptrToA *Thingy := &a
ptrToThingy *Thingy := new(Thingy)
```

But:

```
a.foo()
b.foo()
```

Only for loops

```
for i := 0; i < 10; i++ {...}

for !done {...}

for i, value := range values {...}
```

Slices instead of vectors

```
a := []int{1, 2, 3, 4, 5, 6}
a = append(a, 7)
```

Maps

```
a := make(map[int]string)
m[0] = “Null”
m[1] = “Eins”
```

Multiple return values

```
func returnTwoThings() (int, string) {
   return 4711, "Go > Rust"
}
func main() {
a, b := returnTwoThings()
}
```

defer statement

```
func deferDemo() {
    output, err := os.Create(“test.txt”)
    defer output.Close()
}
```

But let's dive deeper

Packages

```
package foo
------------------------------------------------------------------



package bar

import “foo”

func doSomething() {
    foo.doSomethingFromFoo()
    ...
}
```

Generics *(finally)*

```
func Sum[V int | int64 | float64](m []V) V {
   var s V
   for _, v := range m {
       s += v
   }
   return s
}

func main() {
 nums := []float64{1,2.5,3,4,5}
 fmt.Printf("Sum: %v\n", Sum(nums))
}
```

… and interfaces

```
type Shape interface {
   getWidth() int
   getHeight() int
}

type Rectangle struct {...}

func (r *Rectangle) getWidth() int {...}
func (r *Rectangle) getHeight() int {...}

var shape Shape = new(Rectangle)
shape.getHeight()
```

No classes, but structs

```
package thingy

type thingy struct {
    Count int //public (available outside of thingy)
    x int //private
    y int //private
}
func NewThingy(x int, y int) *thingy {
    t := thingy{0, x, y}
    return &thingy
}
```

Methods

```
func (this *Thingy) DoSomething(x int) string {
    this.string = “Hello World”
    return this.string
}

func (m Meters) toFeet() Feet {
    return m * 3,281;
}
```

Embedding structs

```
type Person struct {
   Name string
   Birthdate Date
}

type Student struct {
   Person
   ID int
}

var a Student
a.Name = “Klaus”
a.ID = 12
```

Duck-typing

```
type Shape interface {
   GetWidth() float64
}
type Circle struct {
   radius float64
}
func (c Circle) GetWidth() float64 {
  return c.radius * 2
}
func main() {
 var c Shape = Circle{10.0}
 fmt.Println(c.GetWidth())
}
```

Goroutines

```
doFirstThing()
doSecondThing()
```

Goroutines

```
go doFirstThing()
doSecondThing()
```

Channels

```
channel := make(chan int)
go doSomething() {
   channel <- 5
}

number := <- channel
```

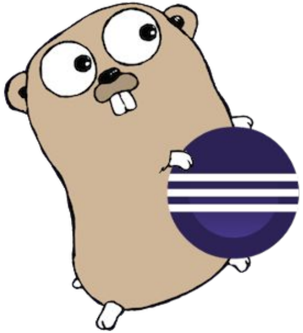
Channels

```
for response := range channel {
    fmt.Println(response)
}
```

Buffered channel

```go
channel := make(chan int, 100)
```

How to get started



References

- The Go Programming Language, Alan A. A. Donovan
  - ISBN 978-0134190440
  - https://www.gopl.io/
- The Go Programming Language Specification
  - https://golang.org/ref/spec
- Effective Go
  - https://golang.org/doc/effective_go.html
- Go by example
  - https://gobyexample.com/
- A C++ developer looks at Go
  - https://www.murrayc.com/permalink/2017/06/26/a-c-developer-looks-at-go-the-programming-language-part-1-simple-features/