To GO or not to GO

Соколов Евгений

Как называть GO программистов?

Как называть GO программистов?

- GOисты

Как называть GO программистов?

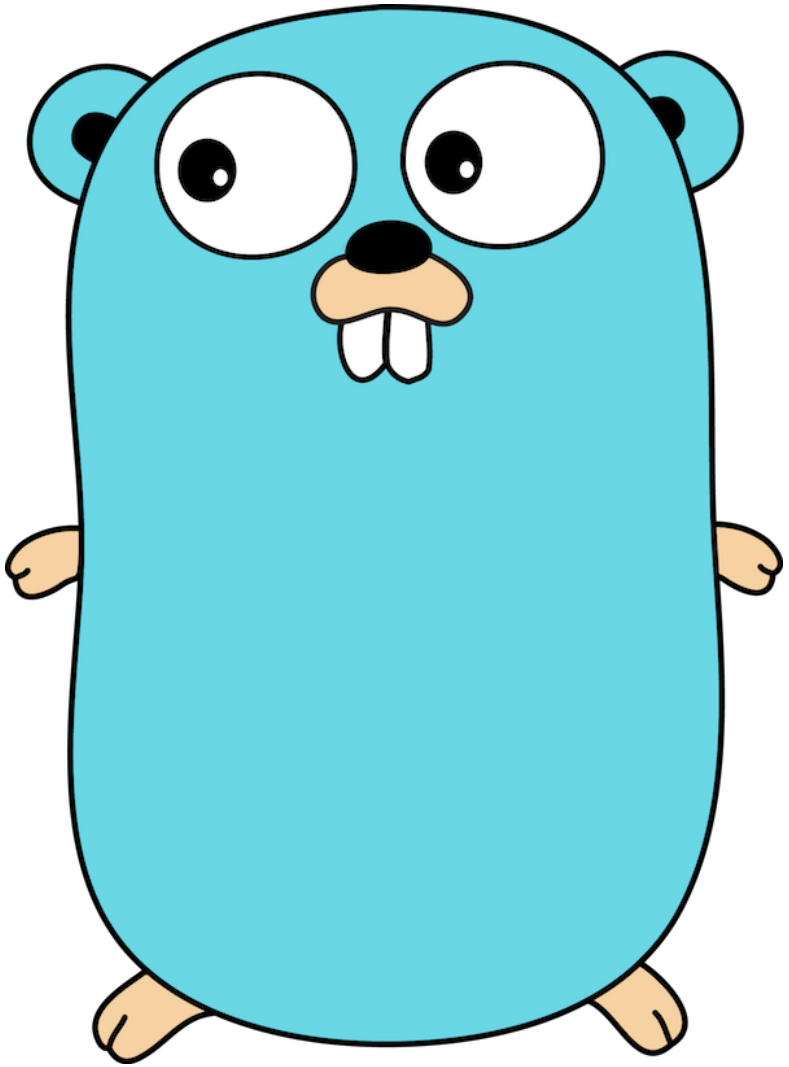
- GOисты
- GOнщики

Как называть GO программистов?

- GOисты
- GOнщики
- GOшники

Как называть GO программистов?

- GOисты
- GOнщики
- GOшники
- GOферы



Gopher(англ.) - суслик

Кратко о Golang

Go (часто также golang) — компилируемый многопоточный язык программирования, разработанный внутри компании Google.

Ключевые моменты:

- Разрабатывался как замена C/C++
- Минимум ключевых слов, легко читаемый код
- Отсутствие привычного ООП
- Сборка мусора
- Встроенные средства распараллеливания, простые и эффективные

Для каких задач подходит Golang

- ✓High-load приложения
- ✓Разработка DevOps и админских инструментов
- ✓Клиент-серверные приложения
- ✓Обработка Big Data
- ✓… и все остальное

Чего не нужно ждать от Go

- Полноценного ООП
- Механизма исключений
- Краткости
- Обилия библиотек на все случаи жизни

Работа с модулями

Модуль —специальным образом описанный пакет, содержащий информацию о своей версии.

Основные идеи:

✓ При импорте фиксируется используемая версия модуля
✓ При сборке импортированные модули автоматически обновляются
✓ Система сборки контролирует все зависимости на совместимость изменений

Работа с модулями. Практика

```
 1  package main
 2
 3  import (
 4      "github.com/stretchr/testify/assert"
 5      "testing"
 6  )
 7
 8  func TestNewModule(t *testing.T) {
 9      assert.Equal(t, 5, 4)
10  }
11
```

Работа с модулями. Практика

```
 1  package main
 2
 3  import (
 4      "github.com/stretchr/testify/assert"
 5      "testing"
 6  )
 7
 8  func TestNewModule(t *testing.T) {
 9      assert.Equal(t, 5, 4)
10  }
11
```

```
$ go test

main.go:3:8: no required module provides package
github.com/stretchr/testify/assert; to add it:

go get github.com/stretchr/testify/assert
```

Работа с модулями. Практика

```
 1  package main
 2
 3  import (
 4      "github.com/stretchr/testify/assert"
 5      "testing"
 6  )
 7
 8  func TestNewModule(t *testing.T) {
 9      assert.Equal(t, 5, 4)
10  }
11
```

```
$ go test
main.go:3:8: no required module provides package
github.com/stretchr/testify/assert; to add it:
go get github.com/stretchr/testify/assert
$ go get github.com/stretchr/testify/assert
go: downloading github.com/stretchr/testify v1.8.1
go: downloading github.com/davecgh/go-spew v1.1.1
go: downloading github.com/pmezard/go-difflib v1.0.0
go: downloading gopkg.in/yaml.v3 v3.0.1
go: added github.com/davecgh/go-spew v1.1.1
go: added github.com/pmezard/go-difflib v1.0.0
go: added github.com/stretchr/testify v1.8.1
go: added gopkg.in/yaml.v3 v3.0.1
```

Работа с модулями. Практика

```
 1  package main
 2
 3  import (
 4      "github.com/stretchr/testify/assert"
 5      "testing"
 6  )
 7
 8  func TestNewModule(t *testing.T) {
 9      assert.Equal(t, 5, 4)
10  }
11
```

```
$ go test
--- FAIL: TestNewModule (0.00s)
    maintest.go:7:
            Error Trace:
/home/esokolov/test1/maintest.go:7
            Error:      Not equal:
                        expected: 5
                        actual  : 4
            Test:       TestNewModule
FAIL
exit status 1
FAIL        testmodule  0.012s
```

```
$ go test
main.go:3:8: no required module provides package
github.com/stretchr/testify/assert; to add it:
go get github.com/stretchr/testify/assert
$ go get github.com/stretchr/testify/assert
go: downloading github.com/stretchr/testify v1.8.1
go: downloading github.com/davecgh/go-spew v1.1.1
go: downloading github.com/pmezard/go-difflib v1.0.0
go: downloading gopkg.in/yaml.v3 v3.0.1
go: added github.com/davecgh/go-spew v1.1.1
go: added github.com/pmezard/go-difflib v1.0.0
go: added github.com/stretchr/testify v1.8.1
go: added gopkg.in/yaml.v3 v3.0.1
```

Чистый код «из коробки»

gofmt -w yourcode.go

go fmt path/to/your/package

```
1  package main
2
3  import "fmt"
4
5  func AwesomeFunc()error{for i:=0;i<10;i++{fmt.Println("Awesome print")};return nil}
```

Чистый код «из коробки»

gofmt -w yourcode.go

go fmt path/to/your/package

```
1  package main
2
3  import "fmt"
4
5  func  AwesomeFunc( )  error  {
6  for i := 0 ; i < 10 ; i++    {
7  fmt.Println("Awesome print") }
8  return           nil         }
```

Чистый код «из коробки»

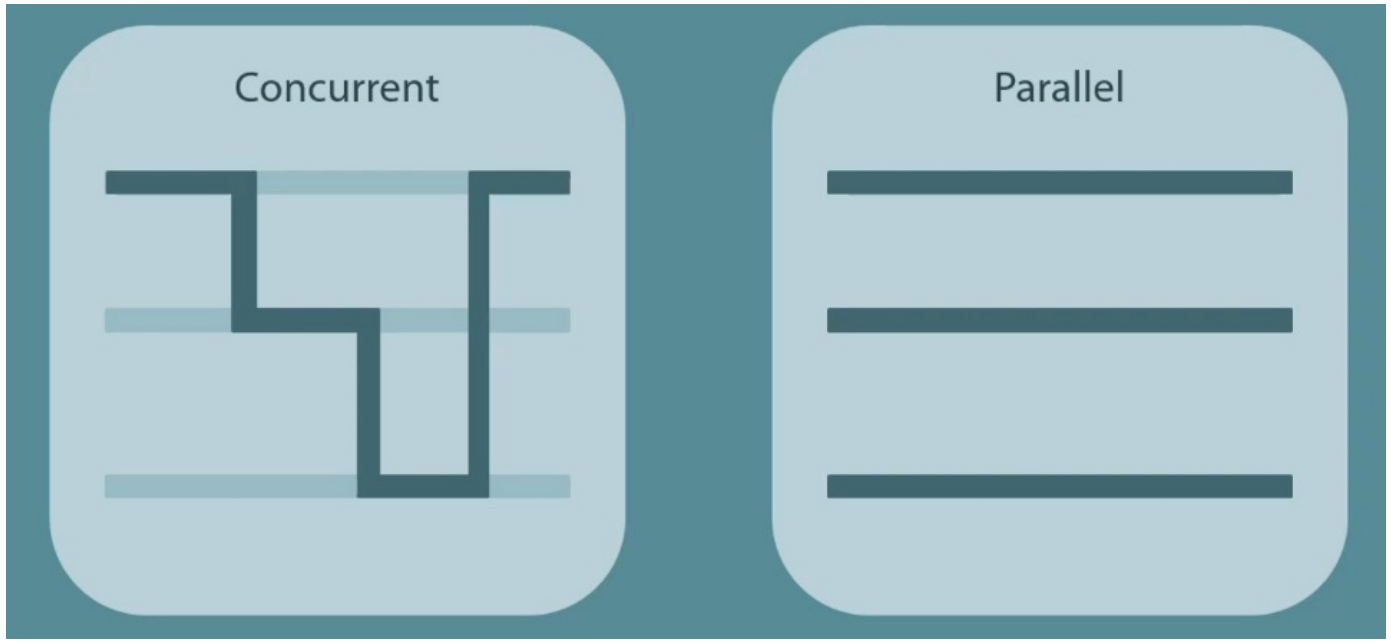
gofmt -w yourcode.go

go fmt path/to/your/package

```
 1  package main
 2
 3  import "fmt"
 4
 5  func AwesomeFunc() error {
 6      for i := 0; i < 10; i++ {
 7          fmt.Println("Awesome print")
 8      }
 9      return nil
10  }
```

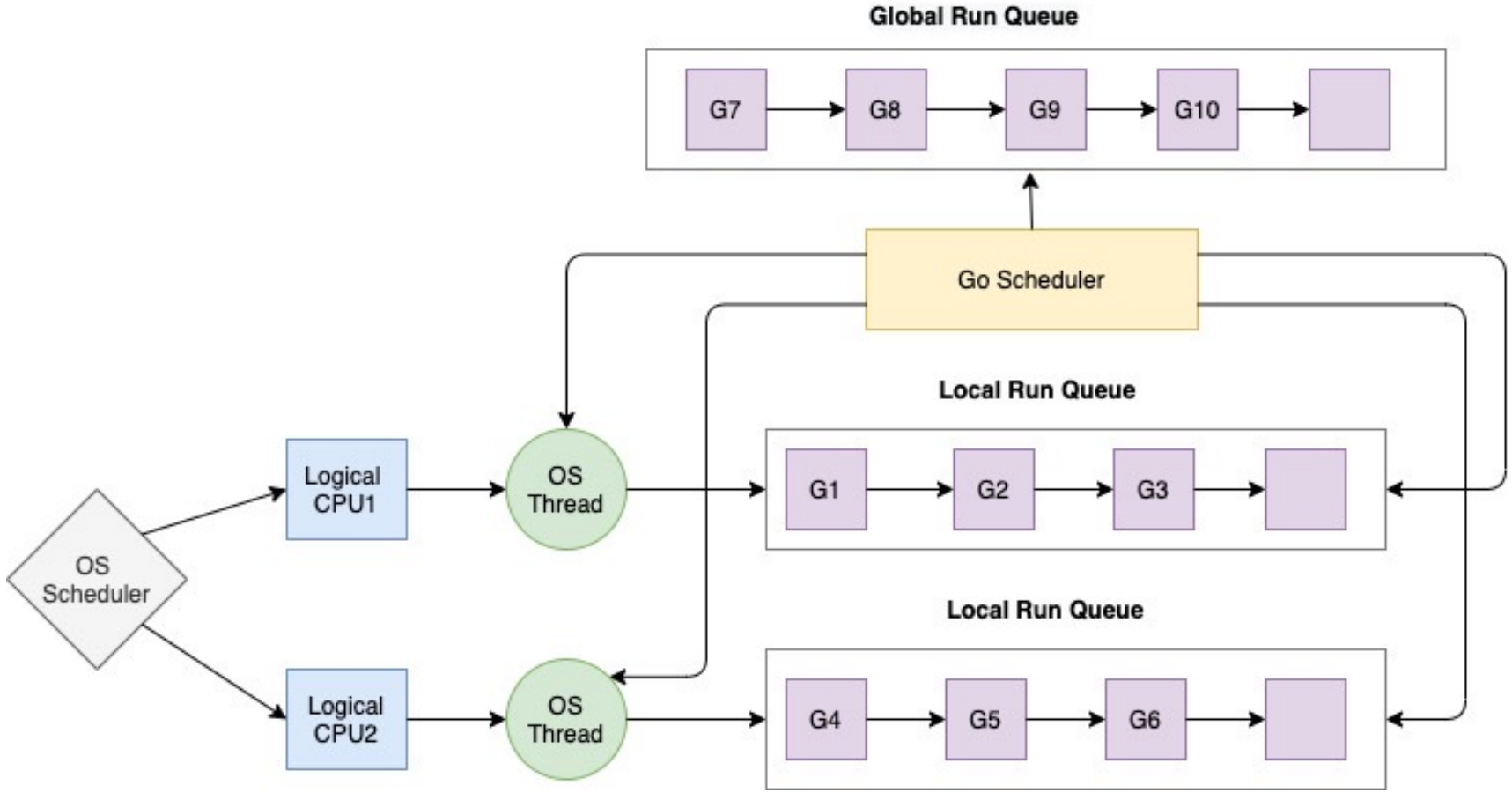
«Многопоточность» в Golang

Конкурентность vs Параллелизм



В Golang «многопоточность» подразумевает и конкурентность, и параллелизм одновременно.

«Многопоточность» в Golang. Планировщик



«Многопоточность» в Golang. Горутины

Горутины (goroutines) - обёрточный функционал потоков, которыми управляет рантайм Go.

Оссобенности:

- ✓ легковесны
- ✓ масштабируемы
- ✓ практически «потоки»
- ✓ требуют меньше памяти (2KB)
- ✓ дополнительная память предоставляется горутинам во время выполнения

«Многопоточность» в Golang. Практика

```
 1  package main
 2
 3  import (
 4      "fmt"
 5      "time"
 6  )
 7
 8  func MyFunc(i int) error {
 9      <-time.After(time.Millisecond * 100)
10      fmt.Println("Hello from" , i)
11      return nil
12  }
13
14  func main() {
15      for i := 0; i < 10; i++ {
16          MyFunc(i)
17      }
18      return
19  }
20
```

«Многопоточность» в Golang. Практика

```go
package main

import (
    "fmt"
    "time"
)

func MyFunc(i int) error {
    <-time.After(time.Millisecond * 100)
    fmt.Println("Hello from" , i)
    return nil
}

func main() {
    for i := 0; i < 10; i++ {
        MyFunc(i)
    }
    return
}

```

```
$ go run | ts '%.Ss'
06.845330s Hello from  0
06.945677s Hello from  1
07.045978s Hello from  2
07.146555s Hello from  3
07.246888s Hello from  4
07.347327s Hello from  5
07.447700s Hello from  6
07.547948s Hello from  7
07.648438s Hello from  8
07.748820s Hello from  9
```

«Многопоточность» в Golang. Практика

```
 1  package main
 2
 3  import (
 4      "fmt"
 5      "time"
 6  )
 7
 8  func MyFunc(i int) error {
 9      <-time.After(time.Millisecond * 100)
10      fmt.Println("Hello from" , i)
11      return nil
12  }
13
14  func main() {
15      for i := 0; i < 10; i++ {
16          go MyFunc(i)
17      }
18      <-time.After(time.Millisecond * 1100)
19      return
20  }
21
```

«Многопоточность» в Golang. Практика

```
 1  package main
 2
 3  import (
 4      "fmt"
 5      "time"
 6  )
 7
 8  func MyFunc(i int) error {
 9      <-time.After(time.Millisecond * 100)
10      fmt.Println("Hello from" , i)
11      return nil
12  }
13
14  func main() {
15      for i := 0; i < 10; i++ {
16          go MyFunc(i)
17      }
18      <-time.After(time.Millisecond * 1100)
19      return
20  }
21
```

```
$ go run | ts '%.Ss'
43.417916s Hello from  4
43.418109s Hello from  9
43.418171s Hello from  7
43.418196s Hello from  6
43.418235s Hello from  8
43.418292s Hello from  5
43.418349s Hello from  1
43.418404s Hello from  2
43.418489s Hello from  0
43.418549s Hello from  3
```

«Многопоточность» в Golang. Синхронизация

Канал - объект связи, с помощью которого горутины обмениваются данными

Особенности:

- ✓ можно отправить в качестве параметров в различные горутины
- ✓ работают и как публикатор, и как подписчик
- ✓ бывают буферизованные и НЕбуферизованные

«Многопоточность» в Golang. Практика. НЕбуферизованные каналы

```
 8  func MyFunc(data chan string) {
 9      fmt.Println("Waiting for data...")
10      text := <-data
11      fmt.Println(text)
12  }
13
14  func main() {
15      dataChan := make(chan string)
16
17      fmt.Println("Run MyFunc gorutine")
18      go MyFunc(dataChan)
19
20      dataChan <- "hehey"
21      <-time.After(time.Second)
22  }
```

```
$ go run | ts '%.Ss'
56.622596s Run MyFunc gorutine
56.622697s Waiting for data...
56.622747s hehey
```

«Многопоточность» в Golang. Практика. НЕбуферизованные каналы

```
 8  func MyFunc(data chan string) {
 9      fmt.Println("Waiting for data...")
10      text := <-data
11      fmt.Println(text)
12  }
13
14  func main() {
15      dataChan := make(chan string)
16
17      dataChan <- "hehey"
18
19      fmt.Println("Run MyFunc gorutine")
20      go MyFunc(dataChan)
21
22      <-time.After(time.Second)
23  }
```

«Многопоточность» в Golang. Практика. НЕбуферизованные каналы

```
 8  func MyFunc(data chan string) {
 9      fmt.Println("Waiting for data...")
10      text := <-data
11      fmt.Println(text)
12  }
13
14  func main() {
15      dataChan := make(chan string)
16
17      dataChan <- "hehey"
18
19      fmt.Println("Run MyFunc gorutine")
20      go MyFunc(dataChan)
21
22      <-time.After(time.Second)
23  }
```

```
$ go run | ts '%.Ss'
fatal error: all goroutines are asleep - deadlock!


goroutine 1 [chan send]:
main.main()
   main.go:17 +0x59
exit status 2
```

«Многопоточность» в Golang. Практика. Буферизованные каналы

```
 8  func MyFunc(data chan string) {
 9      fmt.Println("Waiting for data...")
10      text := <-data
11      fmt.Println(text)
12  }
13
14  func main() {
15      dataChan := make(chan string,1)
16
17      dataChan <- "hehey"
18
19      fmt.Println("Run MyFunc gorutine")
20      go MyFunc(dataChan)
21
22      <-time.After(time.Second)
23  }
```

```
$ go run | ts '%.Ss'
49.791121s Run MyFunc gorutine
49.791244s Waiting for data...
49.791295s hehey
```

«Утка должна крякать» или пару слов о типизации

```
 1  package main
 2
 3  import "fmt"
 4
 5  type (
 6      Quacker interface {
 7          Quack()
 8      }
 9      Duck     struct{}
10      Drake    struct{}
11      Duckling struct{}
12  )
13
14  func (Duck) Quack() {
15      fmt.Println("Duck: quack")
16  }
17
18  func (Drake) Quack() {
19      fmt.Println("Drake: ...")
20  }
21
22  func (Duckling) Quack() {
23      fmt.Println("Duckling: quack")
24  }
```

```
27  func main() {
28      quackers := []Quacker{Duck{}, Duckling{}, Drake{}}
29      for  , q := range quackers {
30          q.Quack()
31      }
32  }
```

```
$ go run
Duck: quack
Duckling: quack
Drake: ...
```

«Утка должна крякать» или пару слов о типизации

```
 1  package main
 2
 3  import "fmt"
 4
 5  type (
 6      Quacker interface {
 7          Quack()
 8      }
 9      Duck     struct{}
10      Drake    struct{}
11      Duckling struct{}
12  )
13
14  func (Duck) Quack() {
15      fmt.Println("Duck: quack")
16  }
17
18  func (Drake) Quack() {
19      fmt.Println("Drake: ...")
20  }
21
22  func (Duckling) Quack() {
23      fmt.Println("Duckling: quack")
24  }
```

```
27  func main() {
28      quackers := []Quacker{Duck{}, Duckling{}, Drake{}}
29      for  , q := range quackers {
30          q.Quack()
31      }
32  }
```

```
$ go run
Duck: quack
Duckling: quack
Drake: ...
```

«Утка должна крякать» или пару слов о типизации

```
 1  package main
 2
 3  import "fmt"
 4
 5  type (
 6      Quacker interface {
 7          Quack()
 8      }
 9      Duck     struct{}
10      Drake    struct{}
11      Duckling struct{}
12  )
13
14  func (Duck) Quack() {
15      fmt.Println("Duck: quack")
16  }
17
18  func (Drake) Quack() {
19      fmt.Println("Drake: ...")
20  }
21
22  func (Duckling) Quack() {
23      fmt.Println("Duckling: quack")
24  }
```

```
27  func main() {
28      quackers := []Quacker{Duck{}, Duckling{}, Drake{}}
29      for  , q := range quackers {
30          q.Quack()
31      }
32  }
```

```
$ go run
Duck: quack
Duckling: quack
Drake: ...
```

«Утка должна крякать» или пару слов о типизации

```
 1  package main
 2
 3  import "fmt"
 4
 5  type (
 6      Quacker interface {
 7          Quack()
 8      }
 9      Duck     struct{}
10      Drake    struct{}
11      Duckling struct{}
12  )
13
14  func (Duck) Quack() {
15      fmt.Println("Duck: quack")
16  }
17
18  func (Drake) Quack() {
19      fmt.Println("Drake: ...")
20  }
21
22  func (Duckling) Quack() {
23      fmt.Println("Duckling: quack")
24  }
```

```
27  func main() {
28      quackers := []Quacker{Duck{}, Duckling{}, Drake{}}
29      for  , q := range quackers {
30          q.Quack()
31      }
32  }
```

```
$ go run
Duck: quack
Duckling: quack
Drake: ...
```

«Утка должна крякать» или пару слов о типизации

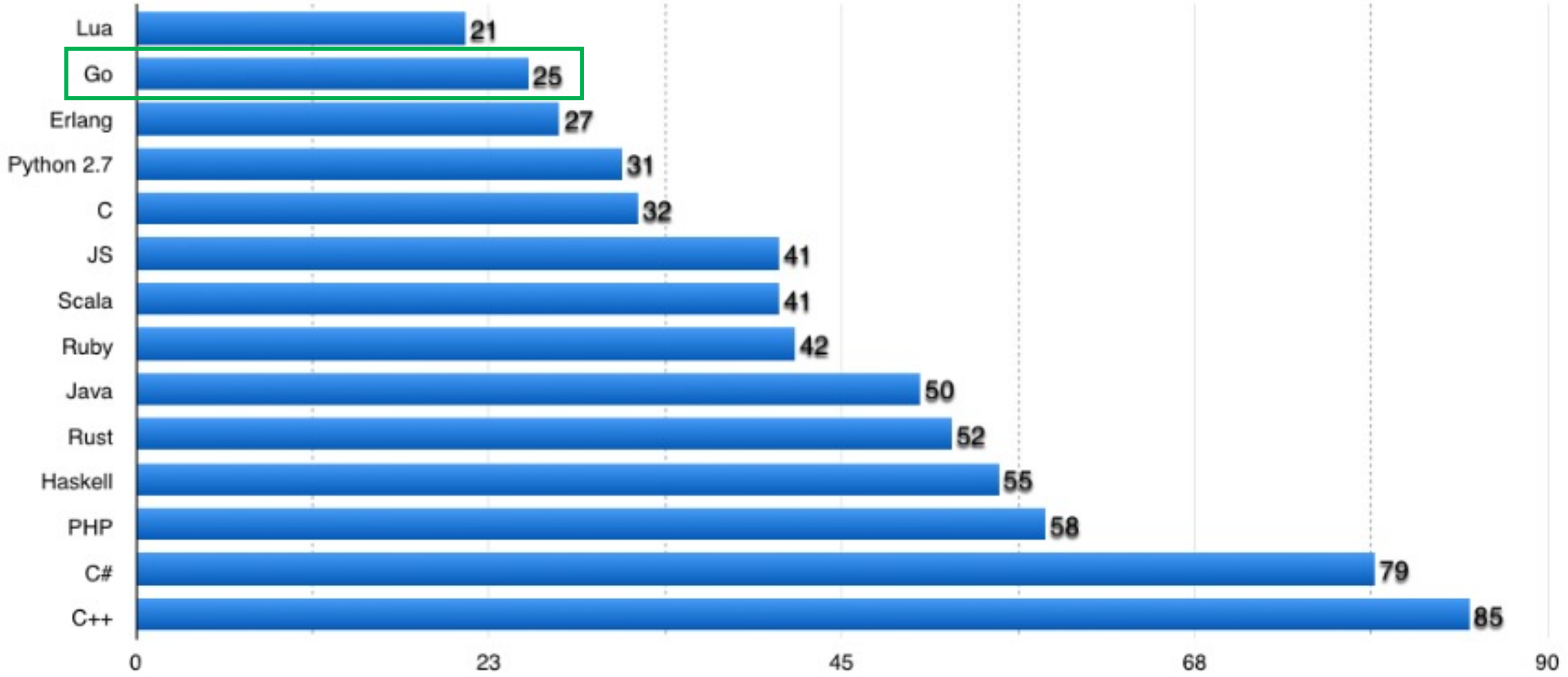
```
 1  package main
 2
 3  import "fmt"
 4
 5  type (
 6      Quacker interface {
 7          Quack()
 8      }
 9      Duck     struct{}
10      Drake    struct{}
11      Duckling struct{}
12  )
13
14  func (Duck) Quack() {
15      fmt.Println("Duck: quack")
16  }
17
18  func (Drake) Quack() {
19      fmt.Println("Drake: ...")
20  }
21
22  func (Duckling) Quack() {
23      fmt.Println("Duckling: quack")
24  }
```

```
27  func main() {
28      quackers := []Quacker{Duck{}, Duckling{}, Drake{}}
29      for  , q := range quackers {
30          q.Quack()
31      }
32  }
```

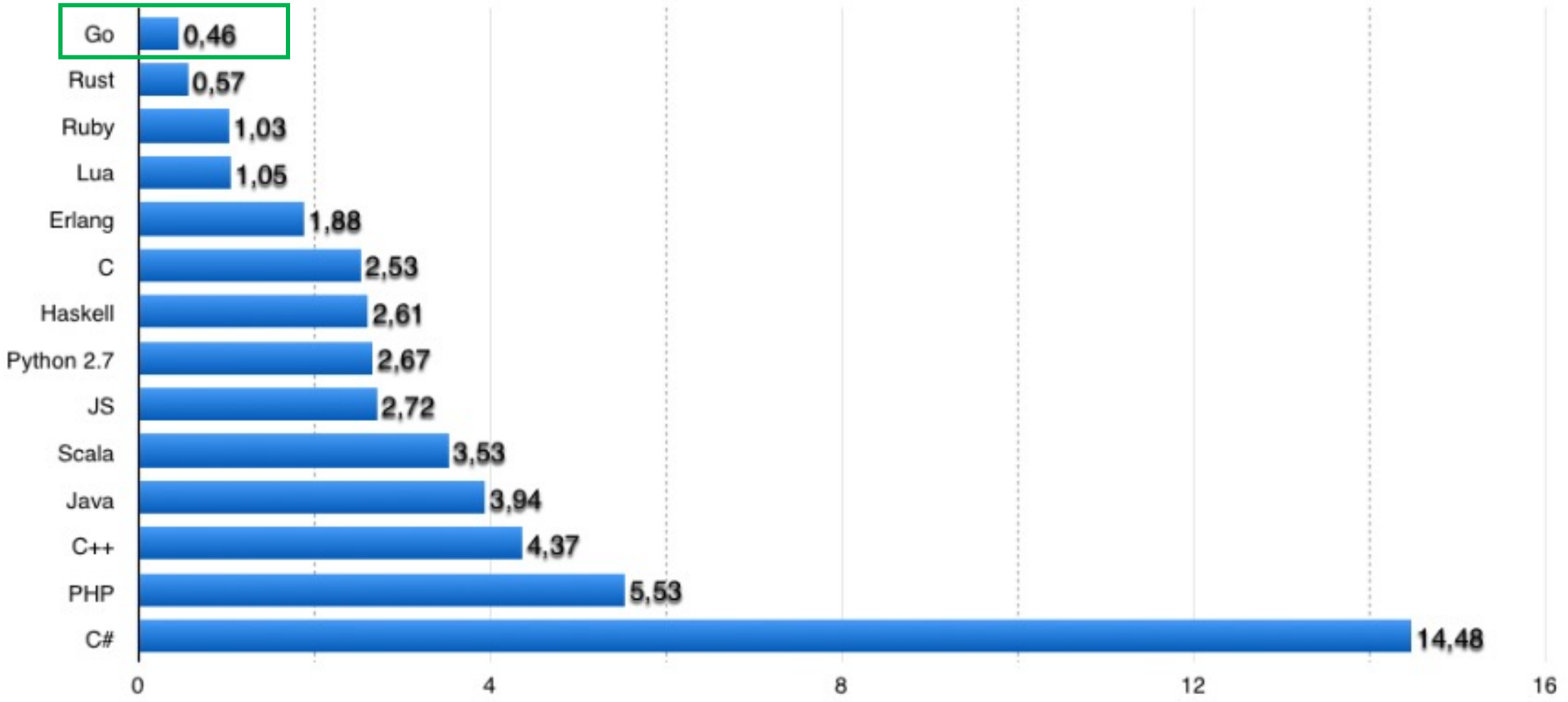
```
$ go run
Duck: quack
Duckling: quack
Drake: ...
```

Почему язык GO простой?

Количество ключевых слов



Количество вопросов на SO / Количество репозиториев на github



Что нужно, чтобы начать?

Official site

- https://go.dev/
- https://go.dev/doc/tutorial/getting-started

Курсы

- https://go.dev/tour/welcome/1
- https://www.codecademy.com/learn/learn-go

Книги

The Go Programming Language | Alan A. A. Donovan · Brian W. Kernighan

Язык программирования Go | Керниган Брайан У., Донован Алан А. А.

IDE

- VS Code
- Goland (Jetbrains)

Q&A