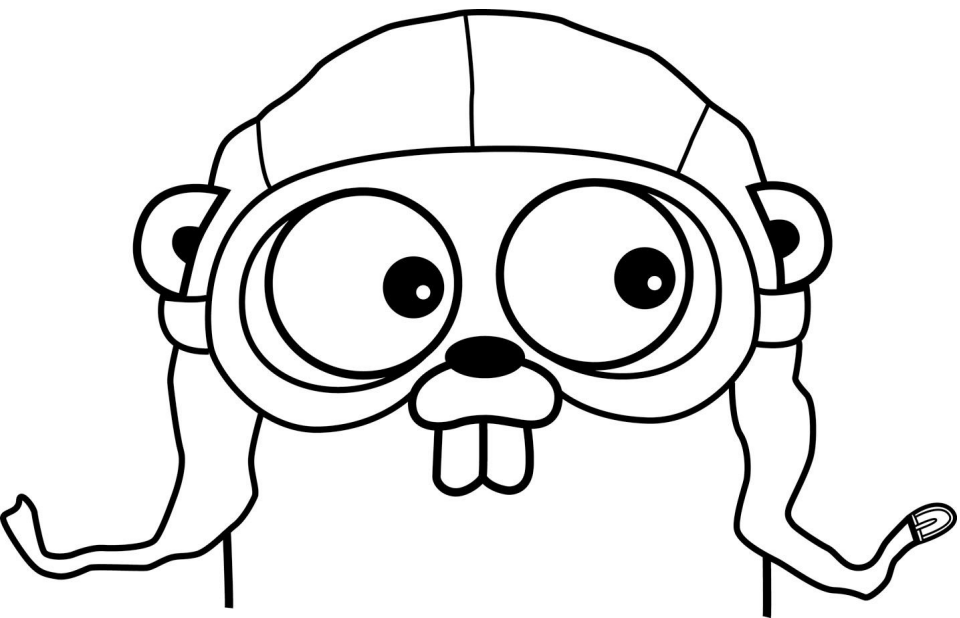


# Learn Go by contributing

Часть #1



<https://github.com/Quasilyte/kfu-go-2018>  
MIT license © 2018

The original Go gopher image copyright  
© 2009 Renee French under

<http://slack.golang-ru.com>

#kfu-go-2018

# Выявление ожиданий от курса

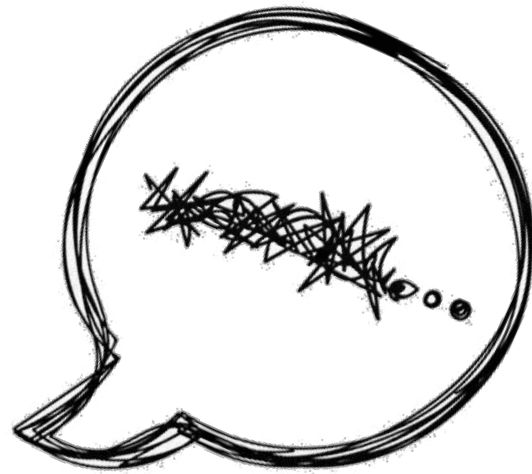
- Основы языка программирования Go
- Семинар по вовлечению в разработку Go
- Что-то другое



Краткое введение в Go

Выбрать задачу

Решить задачу



Патч проходит ревью

Патч попадает в Go

# Области задач

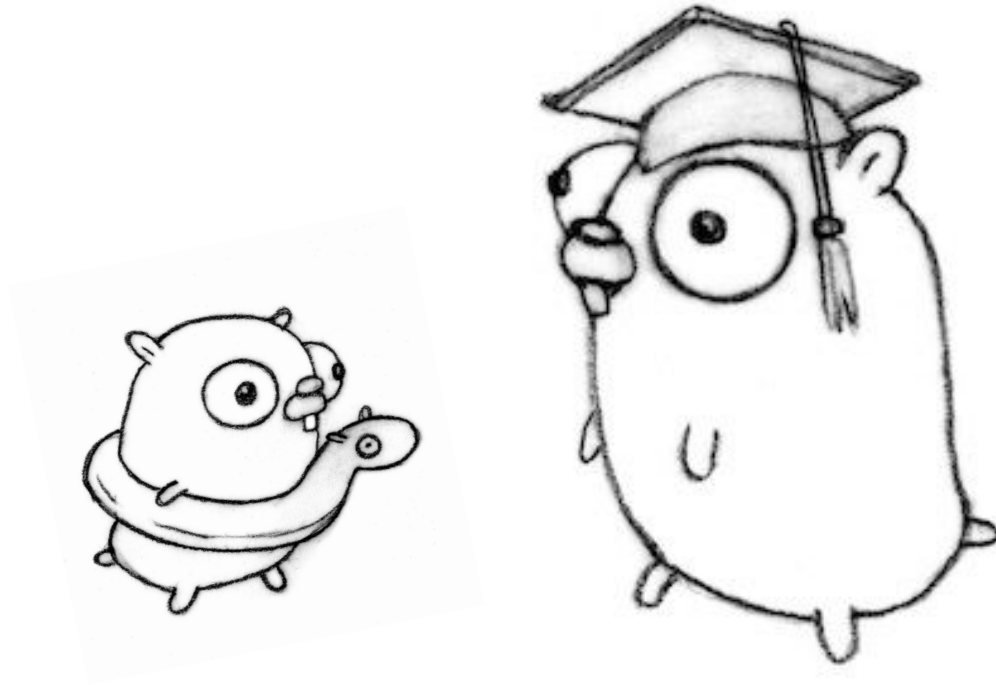
В разработке каких  
частей Go можно  
участвовать?

- Stdlib
- Toolchain
  - Компилятор
  - Ассемблеры
  - ...
- Утилиты
  - Линтеры
  - Работа с кодом
  - ...

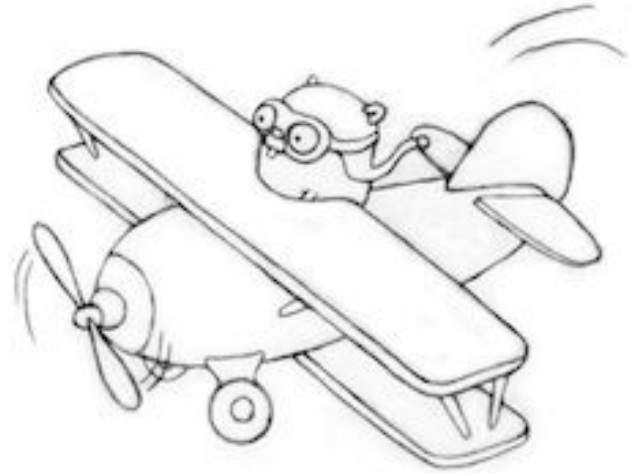
# Виды задач

На чём можно  
сконцентрироваться в  
рамках одной задачи?

- Оптимизация
- Рефакторинг
- Тестирование
- Документирование
- Новые фичи
- Исправление ошибок



Вы сами выбираете уровень сложности



Введение в *Go*



# Learn X in Y minutes

Для самых нетерпеливых

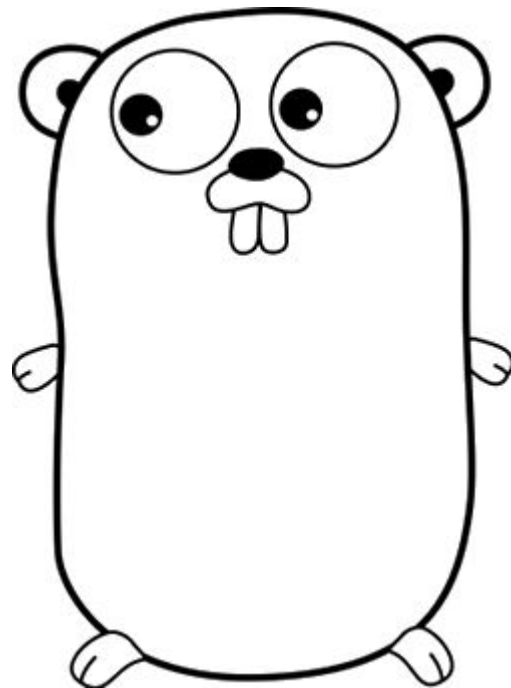
(<https://learnxinyminutes.com>)

## Свойства *Go*

- Сильная (строгая) статическая типизация
- Сборка мусора
- Компилируемый. Быстро компилируемый
- Very opinionated
- Минимализм
- Встроенные средства параллелизма

# Основные концепции

- Программы
- Пакеты
  - Типы
  - Функции и методы
  - Переменные и константы

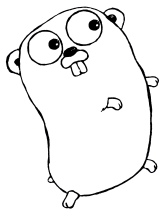


С++ намного “мощнее” и  
больше

*Go* следует принципу “Less is more”

# Сравнение с C++ - возврат локального адреса

```
func f(x int) *int {      int* f(int x) {  
    return &x;             return &x;  
}
```



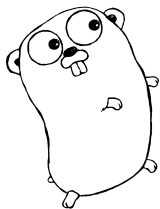
# Сравнение с C++ - подход к контейнерам

```
var xs []int
xs = append(xs, 9)
x := xs[0]
```

// Доступ к элементам  
// проверяемый.

```
std::vector<int> xs;
xs.push_back(9);
int x = xs.at(0);
```

// Оператор [] не  
// проверяет выхода за  
// границы вектора.



# Сравнение с C++ - неявное приведение типов

```
var x float32 = 1.5
```

```
var y float64 = 2.5
```

```
a := x + float32(y)
```

```
b := float64(x) + y
```

```
c := x + y // Error!
```

```
d := add32(y) // Error!
```

```
float x = 1.5;
```

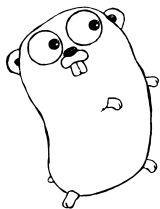
```
double y = 2.5;
```

```
auto a = x + (double)y;
```

```
auto b = (float)x + y;
```

```
auto c = x + y; // OK
```

```
auto d = add32(y); // OK
```



# Сравнение с C++ - спецификация типа

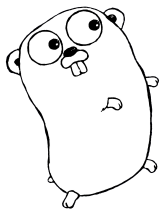
\*Type

Type\*

// Нельзя выразить  
// дополнительную  
// информацию через  
// тип.

const Type\*  
Type const\*  
const Type const\*

volatile Type\*

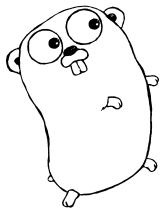




# Сравнение с C++ - функции

- Только обычные функции, как в C
- Методы - это функции, у которых первым аргументом идёт receiver (this)

Функции могут быть перегруженными, иметь параметры по умолчанию, быть шаблонными, вычисляемыми на этапе компиляции, а ещё вам желательно знать об особенностях линковки.



## Сравнение с C++ - обработка ошибок

```
if x < 0 {  
    return 0, errors.New("negative arg")  
}
```

// В Go нет эффективного механизма исключений.  
// Ошибки возвращаются как часть результата.

```
if (x < 0) {  
    throw NegativeArgExn(x);  
}
```

// В C++ можно реагировать на ошибку выбросом  
// исключения.

## Идиома “comma, ok”

```
func lookup(d Dict, key string) (int, bool)
```

```
var d Dict
```

```
d.Insert("a-key", -1)
```

```
a, ok := lookup(d, "a-key")
```

```
println(a, ok) // -1, true
```

```
b, ok := lookup(d, "b-key")
```

```
Println(b, ok) // 0, false
```

# Встроенные контейнеры

|                      |                                |   |
|----------------------|--------------------------------|---|
| <code>[]T</code>     | <code>[]int</code>             | Динамический массив с элементами типа T. В Go его называют “слайс” (slice). |
| <code>[N]T</code>    | <code>[16]byte</code>          | Массив фиксированной длины из N элементов типа T.                           |
| <code>map[K]V</code> | <code>map[string]uint64</code> | Неупорядоченный словарь, который отображает ключи типа K в значения типа V. |

```
xs := make([],int, 64)  
var xs = make([],int, 64)  
var xs []int = make([],int, 64)
```

```
var m = make(map[int][]int)
```

Встроенная функция make.

```
xs := []rune{'a', 'b', 'c'}
```

```
len(xs) // => 3
```

```
xs[0] // => 'a'
```

```
xs[1] // => 'b'
```

```
xs[10] // => panic
```

Slice операции #1

```
xs := []rune{'a', 'b', 'c'}
```

```
xs[0:3] // => {'a', 'b', 'c'}
```

```
xs[1:] // => {'b', 'c'}
```

```
xs[:len(xs)-1] // => {'a', 'b'}
```

Slice операции #2

```
xs := map[string]int{"x":10, "y":20}
```

```
len(xs) // => 2
```

```
xs["x"] // => 10
```

```
xs["y"] // => 20
```

```
v, ok := xs["z"] // => 0, false
```

```
v := xs["z"] // => 0
```

Операции со словарём



|                |                |                                     |
|----------------|----------------|-------------------------------------|
| Go1.4          | Неявный GOROOT | Bootstrap                           |
| Go1.10-scratch | GOROOT         | Прототипирование изменений в Go     |
| Go1.10-root    | GOROOT         | Стабильный GOROOT (для Go1.10-user) |
| Go-user        | GOPATH         | Код, не связанный дистрибутивом Go  |

Рабочее окружение

# GORATH и GOROOT

GORATH - путь в файловой системе, по которому будет производиться поиск импортируемых пакетов.

GOROOT - как GORATH, только для стандартных пакетов Go. Его приоритет всегда выше.

```
$ go get github.com/Quasilyte/kfu-go-2018/...
```

```
src/github.com/Quasilyte/kfu-go-2018/kfugo/turtle
```

github.com/Quasilyte/kfu-go-2018 - путь репозитория

kfugo - название проекта

turtle - название пакета-библиотеки

```
import "github.com/Quasilyte/kfu-go-2018/kfugo/turtle"
```

## Анатомия import пути #1

```
$ go get github.com/Quasilyte/kfu-go-2018/...
```

```
src/github.com/Quasilyte/kfu-go-2018/kfugo/cmd/hello_world/
```

github.com/Quasilyte/kfu-go-2018 - путь репозитория

kfugo - название проекта

cmd - конвенция для main пакетов

hello\_world - название пакета с исполняемой программой

## Анатомия import пути #2

```
package main

import (
    "encoding/json"
    "fmt"
    "log"
)

// Point описывает точку в двумерном пространстве.
type Point struct {
    X float64
    Y float64
}

func main() {
    a := Point{X: 1, Y: 2}

    data, err := json.Marshal(a)
    if err != nil {
        log.Fatal(err)
    }

    fmt.Printf("%v => %s\n", a, data)
    // Печатает:
    // {1 2} => {"X":1,"Y":2}
}
```