



ТИНЬКОФФ

Введение в GO

Лекция №1

План



1. История языка
2. Базовые типы данных
3. Управление выполнением
4. Исходники



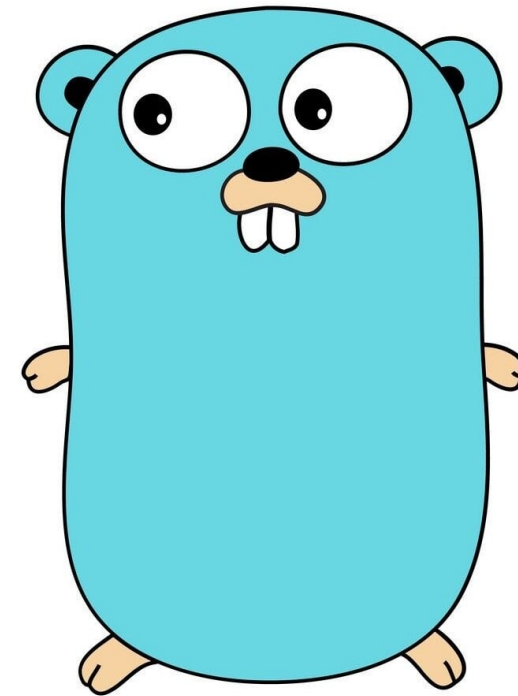
ТИНЬКОФФ

История языка



Google language

Разработка Go началась в сентябре 2007 года, его непосредственным проектированием занимались Роберт Гризмер, Роб Пайк и Кен Томпсон



Роб Пайк



Разработчик операционных систем и языков программирования, работавший с 1980 года в Bell Labs, где в соавторстве с другим программистом написал графический терминал Blit для Unix, и также позднее участвовал в создании операционных систем Plan 9 и Inferno. Один из создателей кодировки UTF-8. Автор текстовых редакторов Sam и Acme.

Кен Томпсон



Пионер компьютерной науки, известен своим вкладом в создание языка программирования С и операционной системы UNIX.

Роберт Гризмер



Швейцарский ученый-компьютерщик. Он наиболее известен своей работой над языком программирования Go. До Go он работал над движком Google V8 JavaScript, языком Sawzall, виртуальной машиной Java HotSpot и системой Strongtalk.

Google language

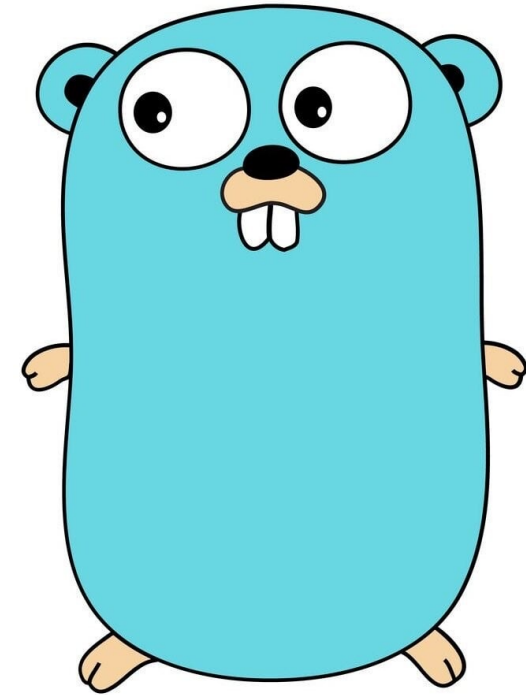
Разработка Go началась в сентябре 2007 года, его непосредственным проектированием занимались Роберт Гризмер, Роб Пайк и Кен Томпсон

Релиз первой версии состоялся 10 ноября 2009 года. Google представили Go — язык программирования, который совместил простоту разработки на Python и скорость C++.

Сейчас на нём уже написано множество сервисов, например, Docker и Kubernetes.

Go разрабатывался как open-сорс проект и публично вышел в 2012 году

<https://github.com/golang/go>



Динамика и вектор развития

Go Release Team

View 1

View 2

View 3

Filter by keyword or by field

Planned

59

go #35678

x/build: add a darwin/amd64 longest builder

Builders

NeedsInvestigation

new-builder

go #49055

x/build: add longest builder for darwin/arm64

arch-arm64

Builders

NeedsDecision

new-builder

go #53862

x/build: build cross-compiled releases for secondary ports

Builders

NeedsInvestigation

In Progress

9

go #49649

x/build: add longest builders for linux-arm and linux-arm64

Builders

NeedsInvestigation

new-builder

go #37486

cmd/dist: add json output support

NeedsInvestigation

go #54773

x/build: upgrade netbsd-386 and netbsd-amd64 builders to pick up recent libpthread fixes

Builders

NeedsInvestigation

Done

3

go #56907

x/build: shard linux-amd64-longtest-race and windows-amd64-longtest-newcc SlowBots

NeedsFix

go #56854

access: running TryBots

NeedsDecision

go #44862

x/build/cmd/updatedst: verify that target GOROOT's bin/go is in PATH

Builders

NeedsFix

9

Динамика и вектор развития

<https://github.com/golang/proposal>

golang / proposal Public

Watch 342 Fork 402

<> Code Pull requests 5 Actions Projects Security Insights

master 1 branch 0 tags

Go to file Add file <> Code

About

Go Project Design Document

Readme

BSD-3-Clause license

3k stars

342 watching

402 forks

jba design/56345-structured-logging.md: s/Logger/*Logger/ ... 77a08c2 14 days ago 354 commits		
design	design/56345-structured-logging.md: s/Logger/*Logger/	14 days ago
CONTRIBUTING.md	CONTRIBUTING.md: remove note about not accepting Pull Requests	5 years ago
LICENSE	initial commit of empty subrepo	7 years ago
PATENTS	initial commit of empty subrepo	7 years ago
README.md	README: clarify that all API changes are substantial/notable	10 months ago

ОСНОВНЫЕ ВОЗМОЖНОСТИ




- Go — язык со строгой статической типизацией. Доступен автоматический вывод типов, для пользовательских типов — «утиная типизация».
- Полноценная поддержка указателей, но без возможности применять к ним арифметические операции, в отличие от C/C++/D.
- Строковый тип со встроенной поддержкой юникода.
- Использование динамических массивов (срезов), хеш-таблиц (словарей), вариант цикла для обхода коллекции.

ОСНОВНЫЕ ВОЗМОЖНОСТИ

- Средства функционального программирования: неименованные функции, замыкания, передача функций в параметрах и возврат функциональных значений.
- Автоматическое управление памятью со сборщиком мусора.
- Средства объектно-ориентированного программирования ограничиваются интерфейсами. Полиморфное поведение обеспечивается реализацией интерфейсов типами. Реализация наследования отсутствует, но типы-структуры могут включать другие типы-структуры в себя.
- Средства параллельного программирования: встроенные в язык потоки (go routines), взаимодействие потоков через каналы и другие средства организации многопоточных программ.
- Достаточно лаконичный и простой синтаксис, основанный на Си, но существенно доработанный, с большим количеством синтаксического сахара.

Другие языки

ЯП созданы для решение конкретных задач, можно сравнить относительно конкретной цели – что вам важно и чем вы готовы жертвовать?

 Factors	 Rust	 Go
Performance	Rust is better than Go	Go is not faster
Concurrency	Rust lacks concurrency	Build-in Concurrency
Memory Management	The program cannot compile if not memory safe	Go handles memory automatically
Development Speed	Development needs more time	Goland web development is speedy
Compilation Speed	Low	Faster
Libraries	Good Libraries	Good Libraries
Complexity	Rust is complex to understand	Go is a simple language
Functionality	Many features and functionalities	Fewer functionalities

Применение go

Go сравним по производительности с C/C++, но программировать на нём легче. Если в C/ C++ приходится вручную управлять памятью, то компилятор Golang берёт эти заботы на себя.

Golang — это компилируемый многопоточный язык с открытым исходным кодом. В основном его применяют в веб-сервисах и клиент-серверных приложениях.



Применение go

The Google logo, featuring the word "Google" in its characteristic multi-colored font.The Meta logo, featuring a blue infinity symbol and the word "Meta" in black.The Netflix logo, featuring the word "NETFLIX" in red capital letters.The Twitch logo, featuring the word "Twitch" in a stylized purple font.

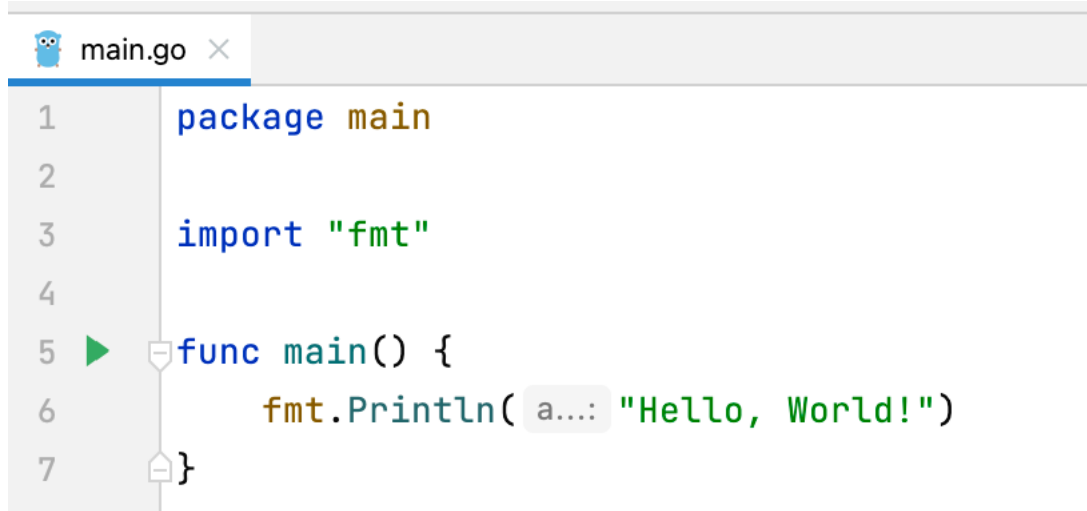
A 3D rendered white gear with ten teeth, casting a soft shadow on the surface below. The word 'Вопросы' is centered within the gear's hole in a bold, black, sans-serif font.

Вопросы

Инструкция по установке

<https://go.dev/doc/install>

Hello World - IDE



```
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Println("Hello, World!")
7 }
```

The screenshot shows a code editor window titled 'main.go'. The code is written in Go and consists of seven lines. Line 1: 'package main'. Line 2: an empty line. Line 3: 'import "fmt"'. Line 4: an empty line. Line 5: 'func main() {' with a green play button icon to its left. Line 6: ' fmt.Println("Hello, World!")' with a light blue selection highlight on the string. Line 7: '}' with a small icon to its left. The code is color-coded: 'package' is blue, 'main' is orange, 'import' is blue, 'func' is blue, 'main' is blue, 'fmt' is orange, 'Println' is blue, and strings are green.

go run main.go

Пакет `fmt` реализует форматированный ввод-вывод с функциями, аналогичными `printf` и `scanf` языка C

Hello World - ONLINE

<https://go.dev/play/>

The Go Playground

```
1 // You can edit this code!  
2 // Click here and start typing.  
3 package main  
4  
5 import "fmt"  
6  
7 func main() {  
8     fmt.Println("Hello, 世界")  
9 }  
10
```

Hello, 世界

Program exited.



ТИНЬКОФФ

Базовые типы данных



Базовые типы данных

Название	Тип
Логический	bool
Целочисленный	int
Беззнаковый целочисленный	uint
Вещественный	float
Комплексные числа	complex
Строки	string

<https://github.com/golang/go/blob/master/src/go/types/basic.go#L50>

Объявление переменных

```
package main
```

```
func main() {  
    var p int = 1000
```

```
    var a int  
    a = 5
```

```
    var (  
        k int  
        f int  
    )
```

```
    b := 50  
    c, _, e := 12, 123, 900  
}
```

Логические

```
package main
```

```
func main() {  
    var flag bool
```

```
    flag = true
```

```
    otherFlag := flag  
}
```

Логические

```
package main
```

```
func main() {  
    var flag bool
```

```
    flag = true
```

```
    otherFlag := flag // Unused variable 'otherFlag'  
}
```


Логические

```
package main
```

```
import "fmt"
```

```
func main() {  
    var flag bool
```

```
    flag = true
```

```
    otherFlag := flag // ok
```

```
    fmt.Println(flag, otherFlag)  
}
```

Логические

```
package main
```

```
import "fmt"
```

```
func main() {  
    var flag bool
```

```
    flag = true
```

```
    otherFlag := flag + false // Invalid operation: flag + false (the operator + is not defined on bool)
```

```
    fmt.Println(flag, otherFlag)  
}
```

Целые числа

DATA TYPE	DESCRIPTION
int8	8-bit signed integer
int16	16-bit signed integer
int32	32-bit signed integer
int64	64-bit signed integer
uint8	8-bit unsigned integer
uint16	16-bit unsigned integer
uint32	32-bit unsigned integer
uint64	64-bit unsigned integer
int	Both in and uint contain same size, either 32 or 64 bit.
uint	Both in and uint contain same size, either 32 or 64 bit.
rune	It is a synonym of int32 and also represent Unicode code points.
byte	It is a synonym of int8 .
uintptr	It is an unsigned integer type. Its width is not defined, but its can hold all the bits of a pointer value.

Целые числа

Type		Length	Range
Signed	int8	1 Byte	[-128, 127]
	int16	2 Byte	[-32768, 32767]
	int32	4 Byte	[-2147483648, 2147483647]
	int64	8 Byte	[-9223372036854775808, 9223372036854775807]
Unsigned	uint8	1 Byte	[0, 255]
	uint16	2 Byte	[0, 65535]
	uint32	4 Byte	[0, 4294967295]
	uint64	8 Byte	[0, 18446744073709551615]

Целые числа

```
package main
```

```
import "fmt"
```

```
func main() {
```

```
    var (  
        a int32  
        b int64  
    )
```

```
    a = 90
```

```
    fmt.Println(a + b)  
}
```

Целые числа

```
package main
```

```
import "fmt"
```

```
func main() {
```

```
    var (  
        a int32  
        b int64  
    )
```

```
    a = 90
```

```
    fmt.Println(a + b) // Invalid operation: a + b (mismatched types int32 and int64)  
}
```

Целые числа

```
package main
```

```
import "fmt"
```

```
func main() {
```

```
    var (  
        a int32  
        b int64  
    )
```

```
    a = 90
```

```
    fmt.Println(int64(a) + b) // ok  
}
```

Целые числа

```
package main
```

```
import "fmt"
```

```
func main() {
```

```
    var a int8
```

```
    a = 127           // int8 { -128; 127 }
```

```
    fmt.Println(a + 1) // - 128
```

```
    fmt.Println(a + 2) // - 127
```

```
}
```

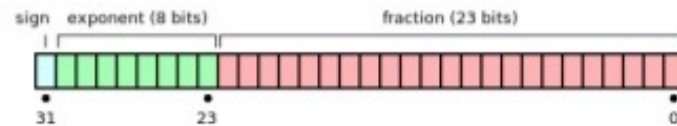

Вещественные числа

Floating Point

- **Single-precision**

- 32 bits

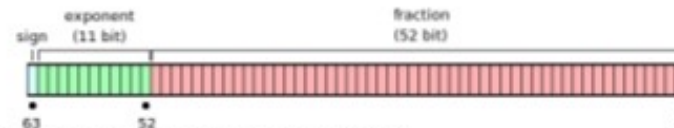
- 1 bit - sign
 - 8 bits - exponent
 - 23 bits - fraction



- **Double-precision**

- 64 bits

- 1 bit - sign
 - 11 bits - exponent
 - 52 bits - fraction



- Note how the size of the fraction increases more than the exponent between single-precision and double-precision
 - Accuracy (precision) is more important than range

Вещественные числа

```
package main
```

```
import (  
    "fmt"  
    "math"  
)
```

```
func main() {  
    var f float32 = 139.8125
```

```
    bits := math.Float32bits(f)
```

```
    fmt.Printf("%b \n", bits) // 1000011000010111101000000000000  
}
```

Вещественные числа

```
package main

import (
    "fmt"
)

func main() {
    fmt.Println(1 / 30)      // 0
    fmt.Println(1.0 / 30.0) // 0.033333333333333333
}
```

А если делим на 0

```
package main
```

```
import "fmt"
```

```
func main() {
```

```
    a := 0
```

```
    fmt.Println(100 / a) // panic: runtime error: integer divide by zero  
}
```

Константы

```
const Pi float64 = 3.14159265358979323846
```

```
const zero = 0.0 // untyped floating-point constant
```

```
const (
```

```
    size int64 = 1024
```

```
    eof      = -1 // untyped integer constant
```

```
)
```

```
const a, b, c = 3, 4, "foo" // a = 3, b = 4, c = "foo", untyped integer and string constants
```

```
const u, v float32 = 0, 3 // u = 0.0, v = 3.0
```

IOTA

```
const (  
  c0 = iota // c0 == 0  
  c1 = iota // c1 == 1  
  c2 = iota // c2 == 2  
)
```

```
const (  
  a = 1 << iota // a == 1 (iota == 0)  
  b = 1 << iota // b == 2 (iota == 1)  
  c = 3          // c == 3 (iota == 2, unused)  
  d = 1 << iota // d == 8 (iota == 3)  
)
```

```
const (  
  u    = iota * 42    // u == 0   (untyped integer constant)  
  v float64 = iota * 42 // v == 42.0 (float64 constant)  
  w    = iota * 42.    // w == 84  (untyped integer constant)  
)
```

```
const x = iota // x == 0  
const y = iota // y == 0
```

IOTA - ENUM

```
package main
```

```
import "fmt"
```

```
type Direction int
```

```
const (  
    North Direction = iota  
    East  
    South  
    West  
)
```

```
func (d Direction) String() string {  
    return [...]string{"North", "East", "South", "West"}[d]  
}
```

```
func main() {  
    fmt.Println(North) // North  
}
```

Значения по умолчанию

```
package main
```

```
import "fmt"
```

```
func main() {
```

```
    var (  
        f float64  
        i int  
        s string  
        b bool  
    )
```

```
    fmt.Println(f, i, s, b) // 0 0 false  
}
```


Размерность типов

```
package main
```

```
import (  
    "fmt"  
    "unsafe"  
)
```

```
func main() {  
    var (  
        f float64  
        i int  
        s string  
        b bool  
)
```

```
    fmt.Println(unsafe.Sizeof(f), unsafe.Sizeof(i), unsafe.Sizeof(s), unsafe.Sizeof(b))  
}
```

Размерность типов

```
package main
```

```
import (  
    "fmt"  
    "unsafe"  
)
```

```
func main() {  
    var (  
        f float64  
        i int  
        s string  
        b bool  
    )
```

```
    fmt.Println(unsafe.Sizeof(f), unsafe.Sizeof(i), unsafe.Sizeof(s), unsafe.Sizeof(b)) // 8 8 16 1  
}
```

Комплексные числа

```
package main
```

```
import (  
    "fmt"  
)
```

```
func main() {
```

```
    var a = complex(2, -2)
```

```
    fmt.Printf("Реальная часть: %f \n", real(a))
```

```
    fmt.Printf("Комплексная часть: %f \n", imag(a))
```

```
}
```

Какой тип?

```
package main

import (
    "fmt"
)

func main() {
    var p int = 1000

    fmt.Printf("%T", p)
}
```

А еще можно создать свой

```
package main
```

```
import (  
    "fmt"  
)
```

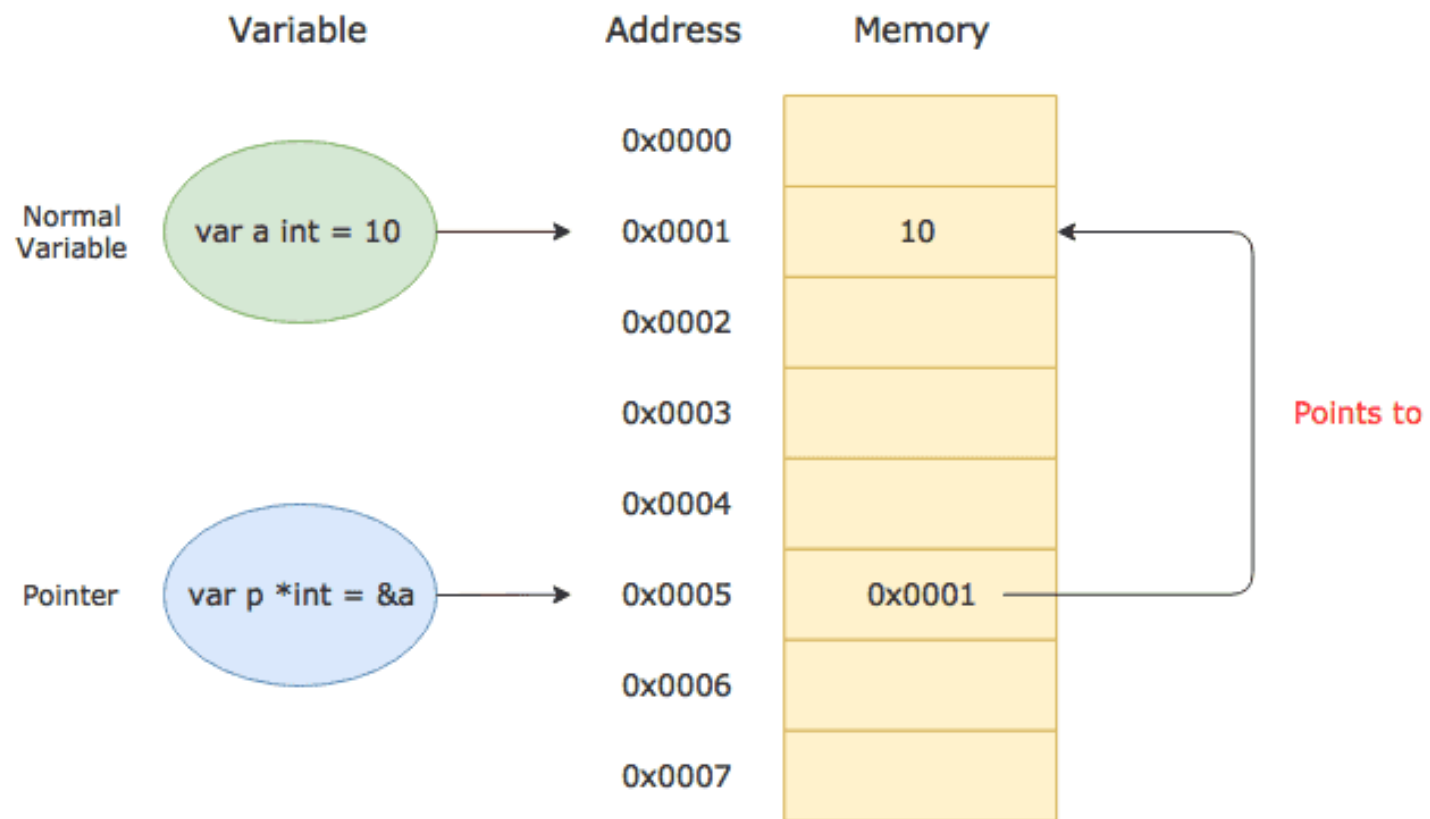
```
type MyType string
```

```
func main() {  
    var p MyType
```

```
    p = "hello"
```

```
    fmt.Printf("%T", p) // main.MyType  
}
```

Указатели



Указатели

```
package main
```

```
import "fmt"
```

```
func main() {
```

```
    var ptr *int
```

```
    a := 1230
```

```
    fmt.Println(&a)    // 0xc00001c098
```

```
    ptr = &a
```

```
    fmt.Println(ptr)   // 0xc00001c098
```

```
    fmt.Println(*ptr)  // 1230
```

```
    fmt.Println(&ptr)  // 0xc000012028
```

```
    *ptr = 90
```

```
    fmt.Println(a)    // 1230 или 90?
```

```
}
```

Указатели

How Pointers works in Go



A 3D rendered white gear with ten teeth, casting a soft shadow on the surface below. The word 'Вопросы' is centered within the gear's hole in a bold, black, sans-serif font.

Вопросы



ТИНЬКОФФ

Управление выполнением



Ветвление

```
package main
```

```
import "fmt"
```

```
func main() {  
    flag := 1 == 23  
    counter := 0
```

```
    if flag {  
        counter += 1  
    } else {  
        counter -= 1  
    }
```

```
    fmt.Println(flag, counter)  
}
```

в go нет тернарного оператора!

Ветвление

```
package main
```

```
import "fmt"
```

```
func main() {  
    if a := 0; a > 10 {  
        fmt.Println("Success")  
    }  
}
```

Ветвление

```
package main
```

```
import "fmt"
```

```
func main() {  
    flag := "a"
```

```
    switch flag {
```

```
    case "s":
```

```
        fmt.Println("S")
```

```
    case "a":
```

```
        fmt.Println("A")
```

```
    default:
```

```
        fmt.Println("NULL")
```

```
    }
```

```
}
```

Ветвление. Fallthrough

```
package main
```

```
import "fmt"
```

```
func test(value int) {
```

```
    switch value {
```

```
    case 1:
```

```
        // Для 1, обрабатываем как 1 и проваливаемся (fallthrough) к 0.
```

```
        fmt.Println("1")
```

```
        fallthrough
```

```
    case 0:
```

```
        fmt.Println("0")
```

```
    case 2:
```

```
        fmt.Println("2")
```

```
    }
```

```
}
```

```
func main() {
```

```
    test(0) // 0
```

```
    fmt.Println("====")
```

```
    test(1) // 1 0
```

```
}
```

ЦИКЛЫ

```
package main
```

```
import "fmt"
```

```
func main() {
```

```
    sum := 0
```

```
    for i := 1; i < 5; i++ {
```

```
        sum += i
```

```
    }
```

```
    fmt.Println(sum) // 10
```

```
}
```

Циклы. While

```
package main
```

```
import "fmt"
```

```
func main() {
```

```
    n := 1
```

```
    for n < 5 {
```

```
        n *= 2
```

```
    }
```

```
    fmt.Println(n) // 8
```

```
}
```


Циклы. Inf

```
package main
```

```
import "fmt"
```

```
func main() {  
    sum := 0
```

```
    for {  
        sum ++  
    }
```

```
    fmt.Println(sum)  
}
```

Циклы. Inf

```
package main
```

```
import "fmt"
```

```
func main() {  
    sum := 0
```

```
    for {  
        sum ++  
        if sum == 10 {  
            break  
        }  
    }
```

```
    fmt.Println(sum) // 10  
}
```

Break – для прерывание цикла

Continue – для перехода к следующей итерации

Циклы. Метка

```
package main
```

```
import "fmt"
```

```
func main() {
```

```
    n, m := 10, 5
```

```
    sum := 0
```

```
    POINT:
```

```
    for i := 0; i < n; i++ {
```

```
        for j := 0; j < m; j++ {
```

```
            sum += i * j
```

```
            if sum > 1000 {
```

```
                break POINT
```

```
            }
```

```
        }
```

```
    }
```

```
    fmt.Println(sum) // 450
```

```
}
```

Циклы. Видимость

```
package main
```

```
import "fmt"
```

```
func main() {
```

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

```
        fmt.Println(&i, i) // будет ли изменяться адрес?
```

```
    }
```

```
    fmt.Println(i) // undefined: i
```

```
}
```

ФУНКЦИИ

```
package main
```

```
import "fmt"
```

```
func add(x int, y int) int {  
    return x + y  
}
```

```
func main() {  
    fmt.Println(add(42, 13))  
}
```

ФУНКЦИИ

```
package main
```

```
import "fmt"
```

```
func add(x, y int) int {  
    return x + y  
}
```

```
func main() {  
    fmt.Println(add(42, 13))  
}
```

ФУНКЦИИ

```
package main
```

```
import "fmt"
```

```
func swap(x, y string) (string, string) {  
    return y, x  
}
```

```
func main() {  
    a, b := swap("hello", "world")  
    fmt.Println(a, b)  
}
```

ФУНКЦИИ

```
package main
```

```
import "fmt"
```

```
func split(sum int) (x, y int) {  
    x = sum * 4 / 9  
    y = sum - x  
    return  
}
```

```
func main() {  
    fmt.Println(split(17))  
}
```


Функции. Много аргументов

```
package main
```

```
import "fmt"
```

```
func add(args ...int) int {
```

```
    total := 0
```

```
    for _, v := range args {
```

```
        total += v
```

```
    }
```

```
    return total
```

```
}
```

```
func main() {
```

```
    fmt.Println(add(1,2,3)) // 6
```

```
}
```

Функции. Анонимные

```
package main
```

```
import "fmt"
```

```
func main() {
```

```
    func (name string) {
```

```
        fmt.Println("Welcome", name)
```

```
    }("Anton")
```

```
}
```

```
// Welcome Anton
```

Функции. Объекты первого рода

```
package main
```

```
import "fmt"
```

```
func split(sum int) (x, y int) {
```

```
    x = sum * 4 / 9
```

```
    y = sum - x
```

```
    return
```

```
}
```

```
func main() {
```

```
    a := split
```

```
    fmt.Printf("%T", a) // func(int) (int, int)
```

```
}
```



Вопросы



ТИНЬКОФФ

Исходники





ДЗ

Материалы

1. <https://habr.com/ru/company/vk/blog/314804> - Ловушки в Go
2. <https://go.dev/ref/spec> - Спецификация
3. <https://go.dev/tour> - Тьюториал

Следующая лекция

Лектор: Михаил Чебаков

Структуры данных и работа с памятью