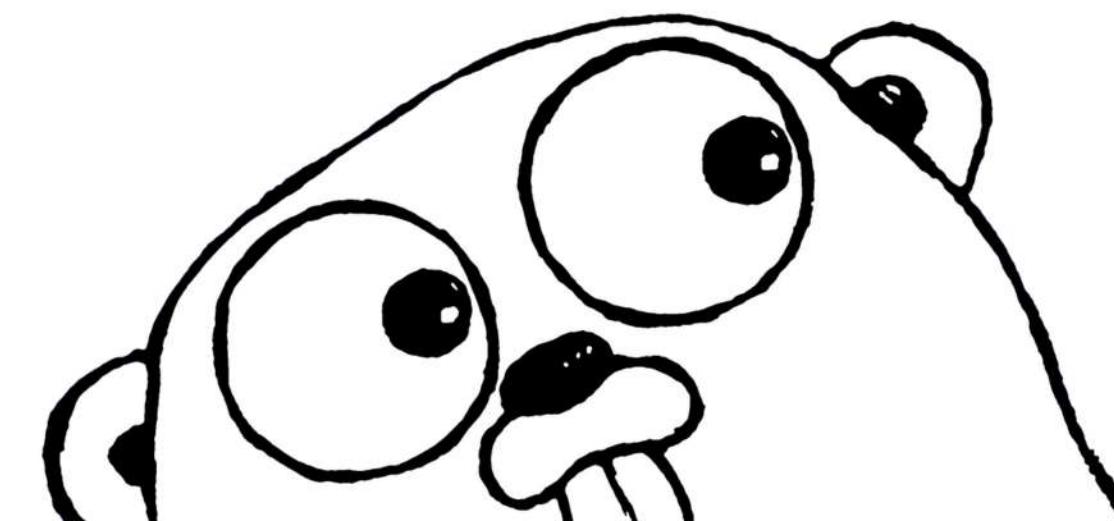


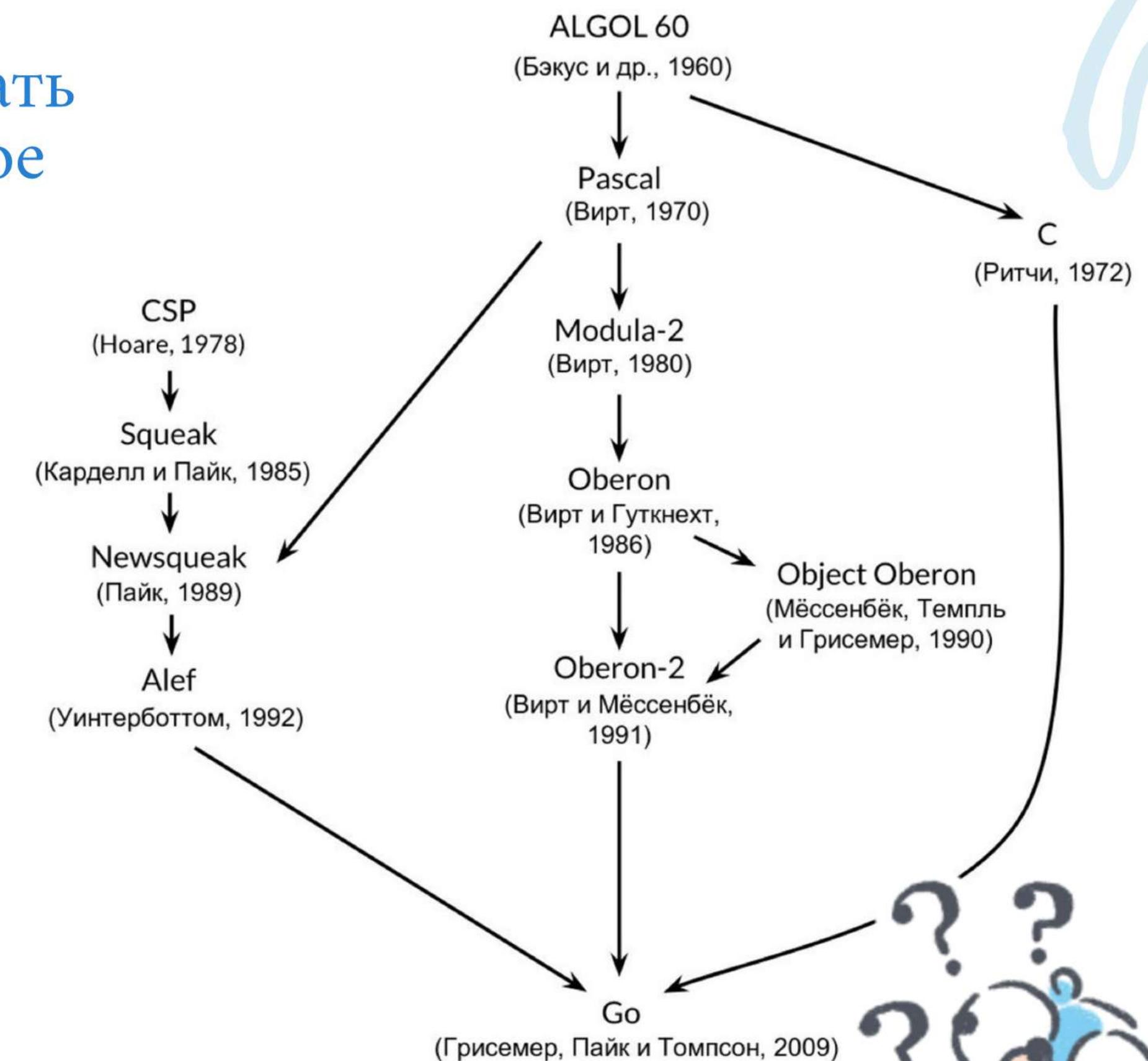
GO Основы языка

Структура программы. Переменные. Типы данных.
Константы. Вывод в консоль. Арифметические операции.
Условные выражения. Присвоение.
Условные конструкции. Циклы.



«Go – это язык программирования с открытым исходным кодом, который позволяет легко создавать простое, надежное и эффективное программное обеспечение»

– go.dev



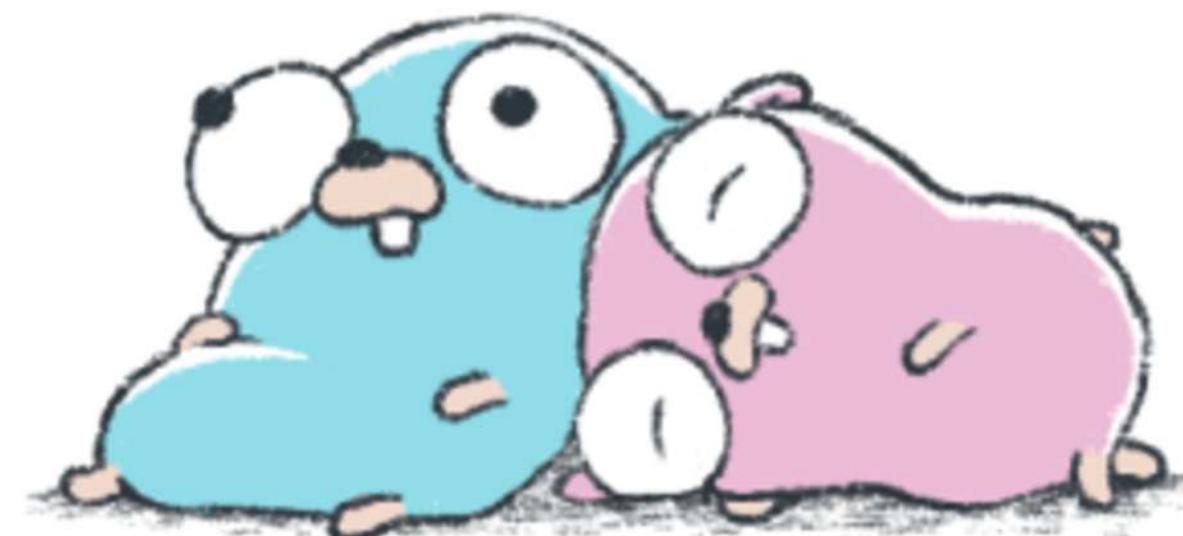
Цели

Go был создан для решения реальных проблем, возникающих при разработке программного обеспечения в Google.

- Эффективность статической типизации;
- Удобство динамической типизации;
- Безопасность;
- Хорошая поддержка многопоточности;
- Эффективный сборщик мусора;
- Быстрая компиляция;
- Эффективная работа с большими объемами кода.

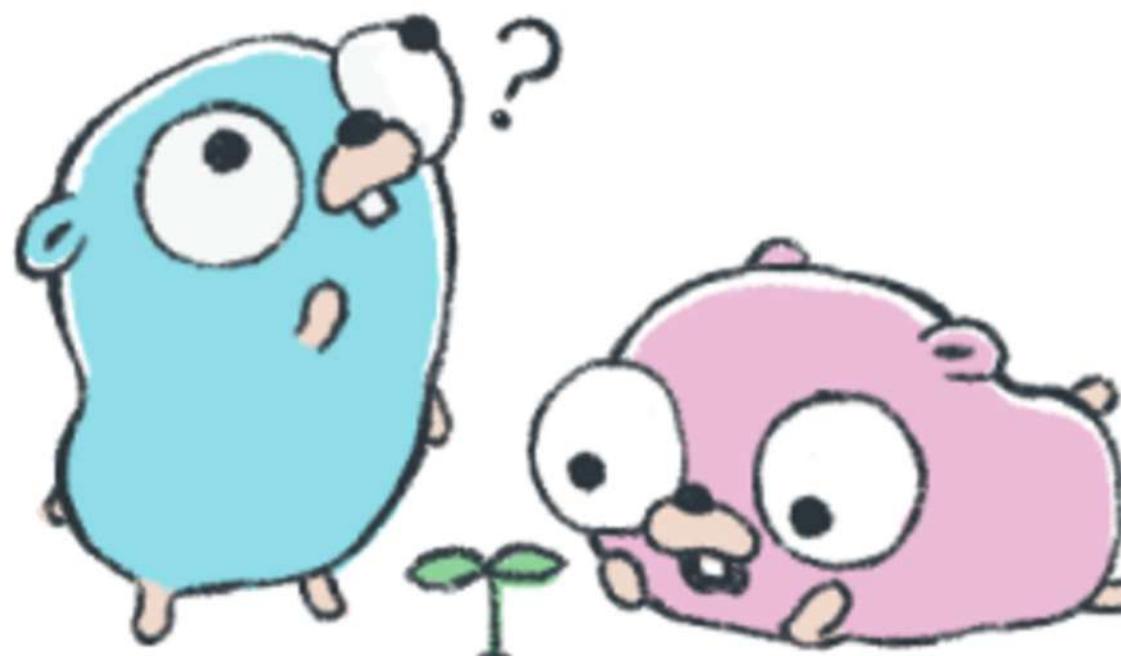
```
# 1 пакет за 200 миллисекунд
time go build -a -v math

# 84 пакета за 7 секунд
time go build -a -v github.com/golang/protobuf/<<.
```



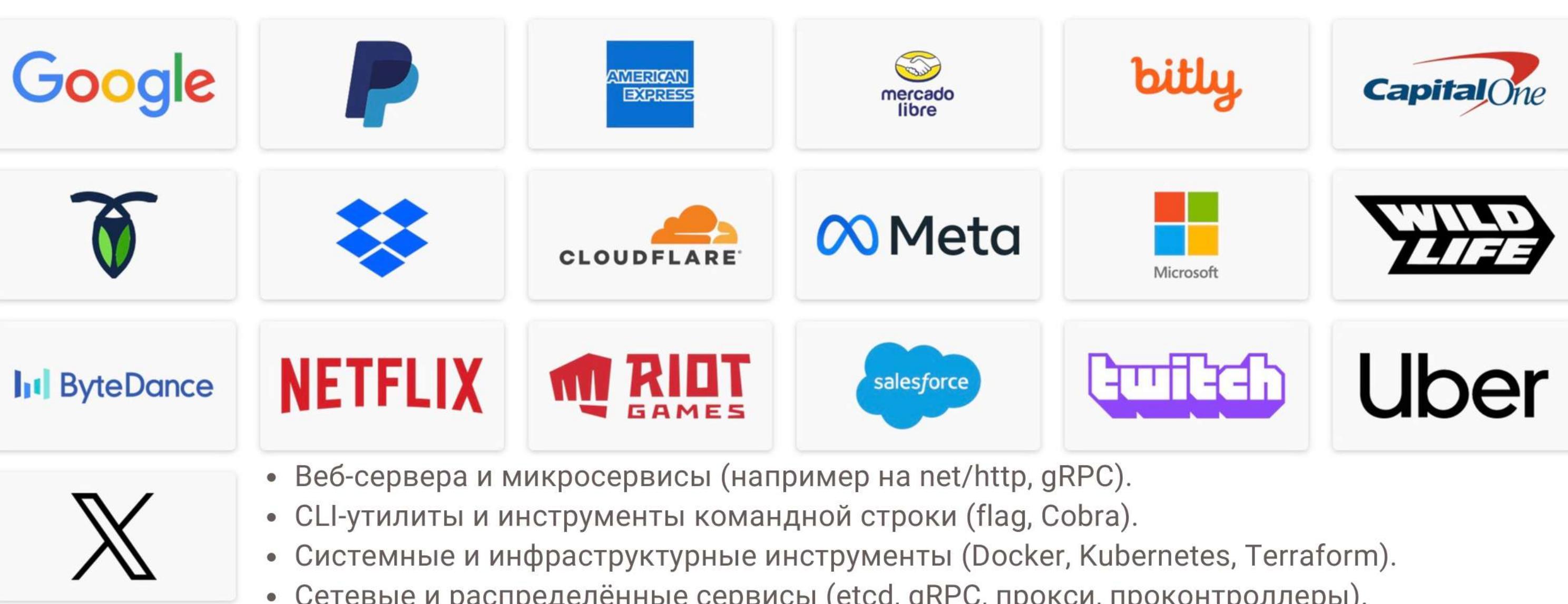
Характеристики

Что стоит помнить?



Свойство	Описание
Компилируемый	Go компилируется в нативный бинарник — выполняется ОС без рантайма, быстрее интерпретируемых языков.
Статическая типизация	Переменные имеют фиксированный тип — ошибки типов ловятся на этапе компиляции.
Автоматическая сборка мусора	Встроенный GC (mark-and-sweep) управляет памятью параллельно с выполнением программы.
Параллелизм	Горутины + каналы для лёгкой конкурентной работы и коммуникации между задачами.
Поддержка многоядерности	GOMAXPROCS задаёт число процессоров для выполнения горутин (по умолчанию — число ядер).
Generics (обобщения)	С версии 1.18 — параметризованные типы для повторного использования кода с разными типами.
Простота тестирования	Встроенный пакет <code>testing</code> и команда <code>go test</code> для юнит-тестов и бенчмарков.
Стандартная библиотека	Широкий набор пакетов для сетей, I/O, сжатия, криптографии и т.д.
Кроссплатформенность	Сборки для Linux, macOS, Windows, FreeBSD, iOS, Android и др.

Что написано на Go ?



- Веб-сервера и микросервисы (например на net/http, gRPC).
- CLI-утилиты и инструменты командной строки (flag, Cobra).
- Системные и инфраструктурные инструменты (Docker, Kubernetes, Terraform).
- Сетевые и распределённые сервисы (etcd, gRPC, прокси, проконтроллеры).
- Системы мониторинга и логирования (Prometheus, экспортёры).
- Распределённые базы и хранилища (CockroachDB, хранилища метаданных).
- Статические генераторы сайтов и утилиты сборки (Hugo).
- Высоконагруженные и параллельные задачи – горутины и каналы для конкурентности.

Структура программы

```
package main

import "fmt"

func main() {
    fmt.Println("Hello, World!")
}
```

```
package main

import "fmt"

func main() {

    fmt.Println("Hello!"); fmt.Println("Hello");
    fmt.Println("Chao!")
}
```

Каждый файл принадлежит пакету: package <name> – в исполняемой программе главный пакет должен быть main.

- Пакеты для использования подключаются после объявления пакета:
import "fmt".
- После импортов идут объявления типов, переменных, констант и функций.
- Входная точка программы – функция func main(); должна быть определена для создания исполняемого файла.
- Базовый элемент – инструкция (обычно одна на строку).
- Несколько инструкций в одной строке можно разделять ;, но это снижает читаемость.

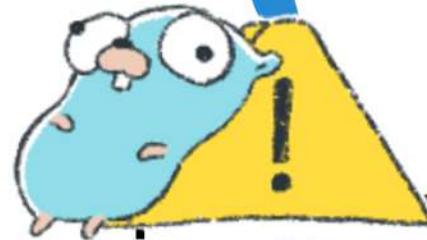
```
go version # узнать версию компилятора
go run <имя файла>.go # компиляция и запуск
go build <имя файла>.go # компиляция
```

Структура программы

Комментарии

- Описание кода, не влияет на компиляцию.
- Однострочный: // – всё после // в строке игнорируется.
- Многострочный: /* ... */ – может занимать несколько строк.

```
/*
Первая программа
на языке Go
*/
package main // определение пакета для текущего файла
import "fmt" // подключение пакета fmt
// определение функции main
func main() {
    fmt.Println("Hello World!") // вывод строки на консоль
}
```



- Имя: буквы, цифры, _
- Первый символ: буква или _
- Регистр важен: `hello` ≠ `Hello`.
- Не использовать ключевые слова:
break, case, chan, const, continue, default, defer, else, fallthrough, for, func, go, goto, if, import, interface, map, package, range, return, select, struct, switch, type, var



Переменные

Переменная – именованный участок памяти для значения определённого типа.

- Объявление: `var` имя тип или кратко
имя := значение (только внутри функции).
- Значение можно менять в любое время (переменные изменяемы).

Переменные

Формы объявления

- Явное с типом: var x int
- С инициализацией: var x int = 5
- Вывод типа: var x = 5
- Краткое (инференция, только в функциях): x := 5
- Несколько через запятую: var a, b, c string
- Группировка:

```
var (  
    name string = "Tom"  
    age  int    = 27  
)
```

Многократное изменение

```
var hello string = "Hello world"  
fmt.Println(hello)  
hello = "Hello Go"  
fmt.Println(hello)
```



Переменные

```
var имя_переменной тип_переменной  
  
var имя_переменной тип_переменной = значение_переменной  
  
var имя_переменной = значение_переменной  
  
имя_переменной := значение_переменной // тип переменной выводится исходя из ее значения  
  
имя_переменной_1, имя_переменной_2 := значение_переменной_1, значение_переменной_2  
  
var имя_переменной_1, имя_переменной_2 тип_переменных // все переменные имеют один тип  
  
var (  
    имя_переменной_1 тип_переменной_1  
    имя_переменной_2 тип_переменной_2  
)  
  
var (  
    имя_переменной_1 = значение_переменной_1  
    имя_переменной_2 = значение_переменной_2  
)
```



Типы данных

Go имеет ряд встроенных типов данных, а также позволяет определять свои типы.

Логический тип или тип `bool` может иметь одно из двух значений: `true` (истина) или `false` (ложь).

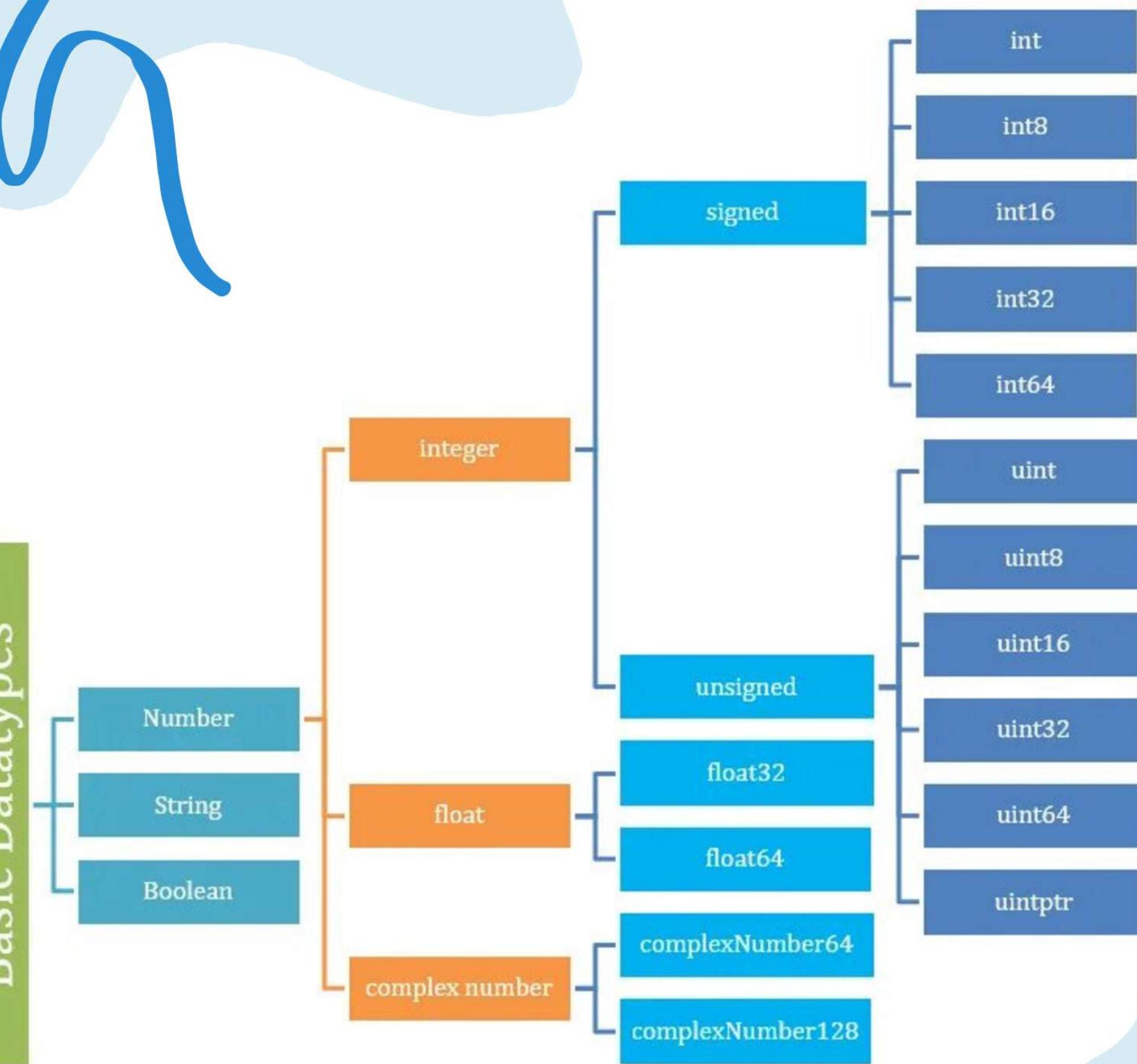
```
var isTrue bool = false
isStudying := true
```

```
package main

import (
    "fmt"
)

func main() {
    var isSunday bool = true
    if isSunday {
        fmt.Println("Today is Sunday!")
    }
}
```

Basic Datatypes



Типы данных

Числовой тип данных

Можем подразделить на целочисленный, с плавающей запятой, символьный.

```
var a int8 = -1
var b uint8 = 2
var c byte = 3
var d rune = -7
var f uint32 = 8
var g uint64 = 10
var h int = 102
var j uint = 105

fmt.Println("a: ", a)
fmt.Println("b: ", b)
fmt.Println("c: ", c)
fmt.Println("d: ", d)
fmt.Println("f: ", f)
fmt.Println("g: ", g)
fmt.Println("h: ", h)
fmt.Println("j: ", j)
```

Тип	Диапазон (включительно)	Размер (байт)	Примечание
int8	-128 ... 127	1	
int16	-32 768 ... 32 767	2	
int32	-2 147 483 648 ... 2 147 483 647	4	
	-9 223 372 036 854 775 808 ... 9 223 372 036 854 775		
int64	807	8	
uint8	0 ... 255	1	
uint16	0 ... 65 535	2	
uint32	0 ... 4 294 967 295	4	
uint64	0 ... 18 446 744 073 709 551 615	8	
byte	0 ... 255		1 синоним uint8
rune	-2 147 483 648 ... 2 147 483 647	4	синоним int32 (кодовая точка Unicode)
int	зависит от платформы: соответствует либо int32, либо int64	4 или 8	размер платформозависимый
uint	зависит от платформы: соответствует либо uint32, либо uint64	4 или 8	беззнаковый, платформозависимый

Типы данных

Числовой тип данных

Числа с плавающей точкой

Для представления дробных чисел есть два типа:

- float32: представляет число с плавающей точкой от 1.4×10^{-45} до 3.4×10^{38} (для положительных). Занимает в памяти 4 байта (32 бита)
- float64: представляет число с плавающей точкой от 4.9×10^{-324} до 1.8×10^{308} (для положительных) и занимает 8 байт.

Тип float32 обеспечивает шесть десятичных цифр точности, в то время как точность, обеспечиваемая типом float64, составляет около 15 цифр

```
package main

import "fmt"

func main() {

    var f float32 = 18
    var g float32 = 4.5
    var d float64 = 0.23
    var pi float64 = 3.14
    var e float64 = 2.7

    fmt.Println("f: ", f)
    fmt.Println("g: ", g)
    fmt.Println("d: ", d)
    fmt.Println("pi: ", pi)
    fmt.Println("e: ", e)
}
```

Типы данных

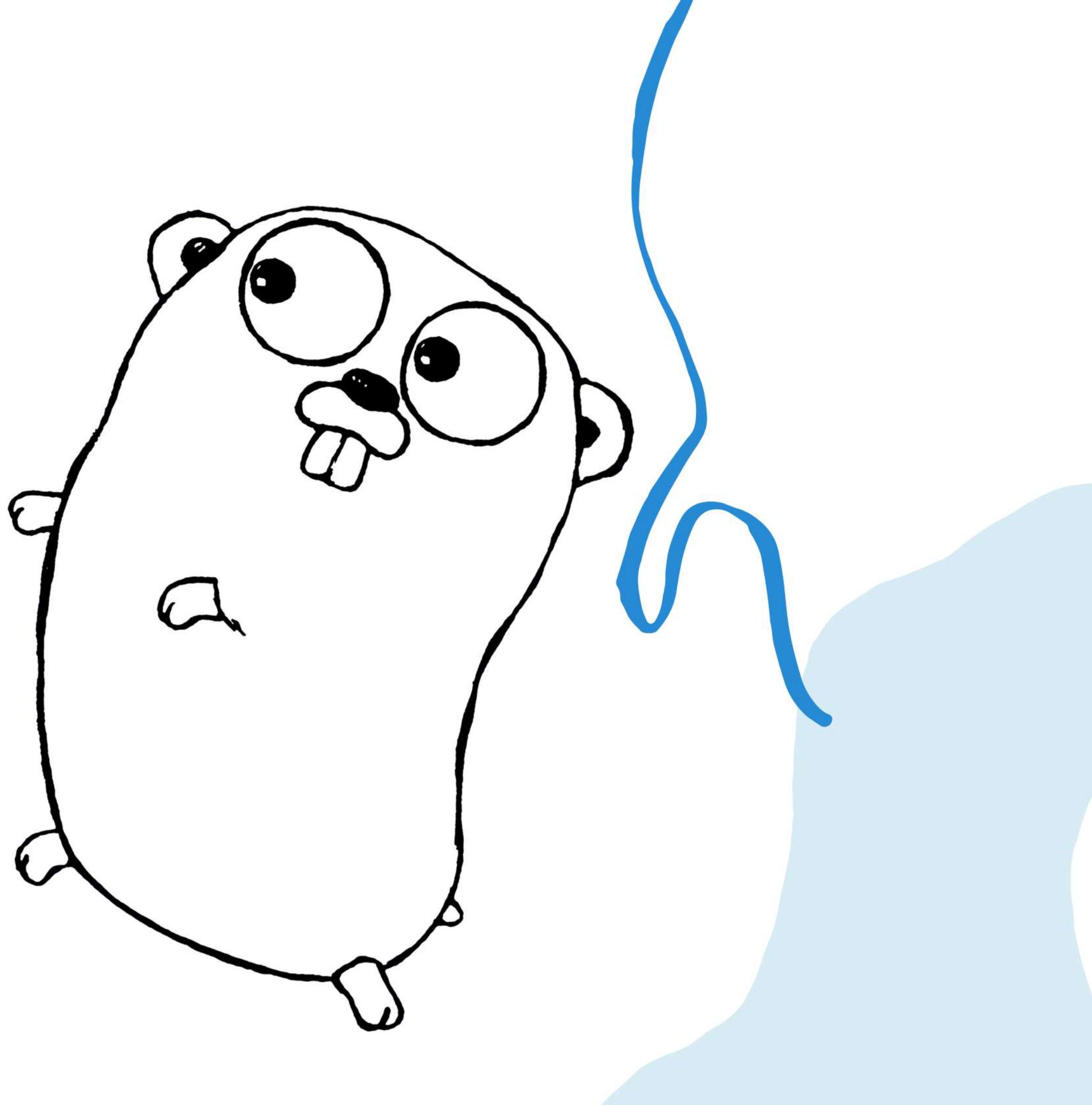
Числовой тип данных

Комплексные числа

Существуют отдельные типы для представления комплексных чисел:

- **complex64**: комплексное число, где вещественная и мнимая части представляют числа float32
- **complex128**: комплексное число, где вещественная и мнимая части представляют числа float64

```
var f complex64 = 1+2i  
var g complex128 = 4+3i
```



Строки

```
package main
import "fmt"

func main() {
    // Объявление и вывод обычной строки
    var name string = "Hello Go"
    fmt.Println(name)

    // Использование управляемых
    // последовательностей
    fmt.Println("Hello\nGo!")
    // \n – переход на новую строку
    fmt.Println("Hello\tGo!")
    // \t – табуляция
    fmt.Println("Path: C:\\Go")
    // \\ – вывод обратного слеша
    fmt.Println("She said: \"Hi\"")
    // \" – кавычки внутри строки
}
```

Строки представлены типом `string`.

В Go строке соответствует строковый литерал - последовательность символов, заключенная в двойные кавычки.

Строка в Go – неизменяемая структура данных, представляющая байтовый срез (UTF-8). Каждый байт хранит часть символа Unicode (кодировка переменной длины).

Строки неизменяемы – после создания изменить содержимое нельзя.

Специальные управляемые последовательности:

\n – переход на новую строку

\r – возврат каретки

\t – табуляция

\" – двойная кавычка внутри строки

\\ – обратный слеш

Значение по умолчанию

```
package main
import "fmt"

func main() {
    // Переменные без инициализации
    var age int          // числовой тип → значение по умолчанию 0
    var isEnabled bool   // логический тип → значение по умолчанию false
    var message string   // строка → значение по умолчанию ""

    // Вывод значений переменных
    fmt.Println("age:", age)           // age: 0
    fmt.Println("isEnabled:", isEnabled) // isEnabled: false
    fmt.Println("message:", message)    // message: (пустая строка)
}
```



Неявная типизация

При определении переменной мы можем опускать тип в том случае, если мы явно инициализируем переменную каким-нибудь значением

В этом случае компилятор на основании значения неявно выводит тип переменной:

- Если присваивается **строка**, то то соответственно переменная будет представлять тип **string**
- Если присваивается **целое число**, то переменная представляет тип **int**
- Если присваивается **true** или **false**, то переменная представляет тип **bool**
- Если присваивается число с плавающей точкой, то переменная представляет тип **float64**

```
package main

import "fmt"

func main() {

    var age = 41
    var isEnabled = false
    var message = "Hello"
    var koef = 1.2

    fmt.Printf("age type: %T\n", age)
    fmt.Printf("isEnabled type: %T\n",
isEnabled)
    fmt.Printf("message type: %T\n",
message)
    fmt.Printf("koef type: %T\n", koef)
}
```

```
package main
import "fmt"

func main() {
    // Определение одиночной константы
    const pi float64 = 3.1415
    fmt.Println("pi =", pi)
    // Попытка изменить значение вызовет ошибку:
    // pi = 2.7182

    // Константы можно инициализировать только
    // константными значениями
    var m int = 7
    // const k = m

    const s = 5
    const n = s
    fmt.Println("s =", s, "n =", n)

    // Множественное определение констант
    const (
        e = 2.7182
        g = 9.81
    )
    fmt.Println("e =", e, "g =", g)
```

Константы

Константы, как и переменные, хранят некоторые данные, но в отличие от переменных значения констант нельзя изменить, они устанавливаются один раз.

```
// Автоматическая инициализация (повторяет
// предыдущее значение)
const (
    a = 1
    b
    c
    d = 3
    f
)
fmt.Println("a, b, c, d, f =", a, b, c, d, f)
```

```

package main
import "fmt"

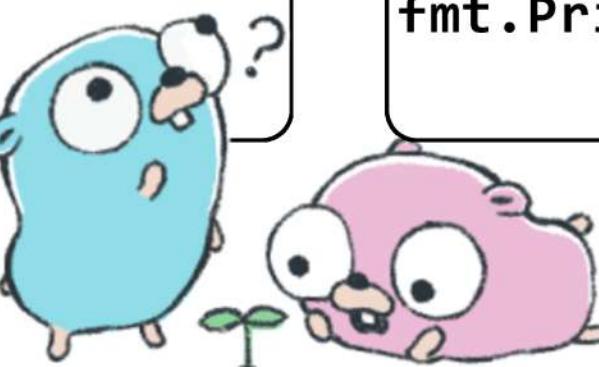
func main() {
    const // iota сбрасывается в 0
    (
        C0 = iota // здесь iota равно 0,
        // увеличивается с каждой строкой
        C1 // увеличение на 1, iota равна 1
        C2 = iota // iota равна 2
    )

    fmt.Println("C0:", C0) // C0: 0
    fmt.Println("C1:", C1) // C1: 1
    fmt.Println("C2:", C2) // C2: 2

    const // iota сбрасывается в 0
    (
        C3 = iota // C3 = 0
    )

    fmt.Println("C3:", C3) // C3: 0
}

```



Константы

Идентификатор iota – это удобный инструмент для генерации последовательностей чисел и автоматических значений констант.

- iota начинается с 0 внутри каждого блока const.
- Для каждой следующей константы в этом блоке iota увеличивается на 1.
- Используется для создания последовательных чисел, битовых флагов, перечислений (enum) и т.д.

```

const (
    KB = 1 << (10 * iota) // 1 << (10*0) = 1
    MB                      // 1 << (10*1) = 1024
    GB                      // 1 << (10*2) = 1048576
)
fmt.Println(KB, MB, GB)

```

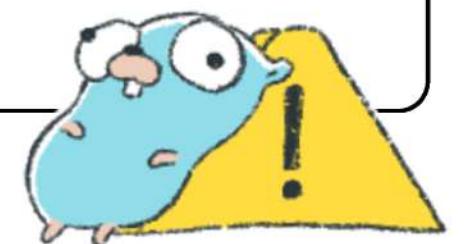
Вывод в консоль

Для вывода информации на консоль используется пакет fmt

- `fmt.Print()` – вывод без переноса строки
- `fmt.Println()` – вывод с переносом строки
- `fmt.Sprintf()` – форматированный вывод (по шаблону)

```
package main
import "fmt"

func main() {
    fmt.Print("Hello ")
    fmt.Println("Go!")
    fmt.Printf("Version: %d\n", 1)
}
```



Вывод в консоль

Println()

- При выводе в конец добавляется перевод строки
- При выводе нескольких значений они разделяются пробелом

```
// Пример 1: вывод одного значения
```

```
var n = 6  
fmt.Println(n) // 6
```

```
// Пример 2: вывод нескольких значений
```

```
var m = 5  
var p = 7  
fmt.Println(m, n, p) // 5 6 7
```

```
// Пример 3: вывод значений разных типов
```

```
var name = "Go"  
var version = 1.22  
var isFast = true  
fmt.Println("Language:", name, "Version:", version, "Fast:", isFast)  
// Language: Go Version: 1.22 Fast: true
```

Вывод в консоль

- Выводит данные без автоматического перевода строки (в отличие от `Println`).
- Не добавляет пробелы между аргументами.
- Можно использовать "`\n`" для перехода на новую строку вручную.
- Удобна, когда нужно точно контролировать формат вывода.

Print()

```
package main
import "fmt"

func main() {
    m := 5
    n := 6

    fmt.Print("m=", m, "; n=", n) // m=5; n=6
    fmt.Print("\n")              // ручной переход на новую строку
    fmt.Print("Sum=", m+n)      // Sum=11

    // Сравнение с Println:
    fmt.Println()                // новая строка
    fmt.Println("m=", m, "; n=", n) // m= 5 ; n= 6 (с пробелами и переносом)
}
```

Вывод в консоль

```
package main

import "fmt"

func main() {
    name := "Go"
    version := 1.22
    year := 2025
    isFast := true

    // Форматированный вывод
    fmt.Printf("Language: %s | Version: %.2f | Year: %d | Fast: %t\n",
               name, version, year, isFast)

    // Выравнивание и ширина вывода
    fmt.Printf("Right align: |%10s|\n", name)
    fmt.Printf("Left align: |%-10s|\n", name)
    fmt.Printf("Float width: |%6.2f|\n",
               version)
}
```

Printf()

- Позволяет форматировать вывод по шаблону с помощью спецификаторов.
- Не добавляет перевода строки автоматически – если нужно, добавляйте \n.
- Удобна для точного и читаемого вывода данных разных типов.
- Поддерживает ширину, точность и выравнивание при формировании.

%d	Десятичное целое
%x, %o, %b	Целое в шестнадцатеричном, восьмеричном и двоичном представлениях
%f, %g, %e	Числа с плавающей точкой
%t	Логическое значение: true или false
%c	Руна (символ Unicode)
%s	Строка
%q	Выводит в кавычках строку типа "abc" или символ 'a'
%v	Любое значение в естественном формате
%T	Тип любого значения
%%	Символ процента

```
package main

import "fmt"

func main() {
    var floatval = 3456.8789

    fmt.Printf("%f: %f\n", floatval)
// точность по умолчанию
    fmt.Printf("%9f: %9f\n", floatval)
// ширина 9, точность по умолчанию
    fmt.Printf("%.2f: %.2f\n", floatval)
// 2 знака после запятой
    fmt.Printf("%9.2f: %9.2f\n", floatval)
// ширина 9, точность 2
    fmt.Printf("%-9.2f: %-9.2f!\n", floatval)
// ширина 9, точность 2, выравнивание влево
}
```

```
%f: 3456.878900
%9f: 3456.878900
%.2f: 3456.88
%9.2f: 3456.88
%-9.2f: 3456.88 !
```

Вывод в консоль

Printf()

К спецификаторам можно добавлять различные флаги, которые влияют на форматирование значений.

Ширина: число перед спецификатором – минимальная длина вывода.

- %9f – число займет минимум 9 позиций, при необходимости дополняется пробелами.

Точность: после точки указывается количество знаков после запятой.

- %.2f – 2 знака в дробной части.

Комбинация ширины и точности:

- %9.2f – ширина 9, точность 2
- %9.f – ширина 9, точность 0

Флаг - – выравнивание влево (значение дополняется пробелами справа).

```
package main

import "fmt"

func main() {
    a := 4
    b := 6

    // Сложение
    c := a + b
    fmt.Println("a + b =", c) // 10

    // Вычитание
    d := a - b
    fmt.Println("a - b =", d) // -2
}
```

Арифметические операции

В Go поддерживаются основные арифметические операции: +, -, *, /, %, ++, --.

- **Операнды** – значения, над которыми производится операция.
- **Результат** – число того же типа (если оба операнда одного типа).



Арифметические операции

```
package main

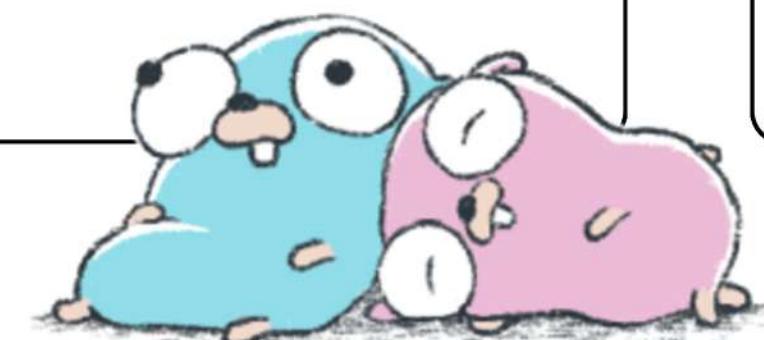
import "fmt"

func main() {
    a := 4
    b := 6

    // Умножение
    c := a * b
    fmt.Println("a * b =", c) // 24

    // Деление целых чисел
    var x int = 10
    var y int = 4
    fmt.Println("x / y =", x / y) // 2

    // Деление вещественных чисел
    var k float32 = 10
    var l float32 = 4
    fmt.Println("k / l =", k / l) // 2.5
}
```



В Go поддерживаются основные арифметические операции: +, -, *, /, %, ++, --.

- **Операнды** – значения, над которыми производится операция.
- **Результат** – число того же типа (если оба операнда одного типа).

```
package main

import "fmt"

func main() {
    var c int = 35 % 3
    fmt.Println("35 % 3 =", c) // 2
}
```

Арифметические операции

```
package main
import "fmt"

func main() {
    a := 8

    // Постфиксный инкремент
    a++
    fmt.Println("a++ =", a) // 9

    // Постфиксный декремент
    a--
    fmt.Println("a-- =", a) // 8
    // Работают только как самостоятельные
    выражения!
}
```

В Go поддерживаются основные арифметические операции: +, -, *, /, %, ++, --.

- **Операнды** – значения, над которыми производится операция.
- **Результат** – число того же типа (если оба операнда одного типа).



Возвращают bool (true или false).

Используются для проверки условий (if, for, и т.д.).

Основные операции:

- == — равно
- != — не равно
- > — больше
- < — меньше
- >= — больше или равно
- <= — меньше или равно

Условные выражения

Операции сравнения

```
package main

import "fmt"

func main() {
    a := 8
    b := 3

    fmt.Println("a == b:", a == b) // false
    fmt.Println("a != b:", a != b) // true
    fmt.Println("a > b:", a > b) // true
    fmt.Println("a < b:", a < b) // false
    fmt.Println("a >= b:", a >= b) // true
    fmt.Println("a <= b:", a <= b) // false
}
```

Условные выражения

Работают с логическими значениями (bool).

Основные операции:

- `!` – отрицание (инвертирует значение)
- `&&` – конъюнкция, `true`, если оба операнда `true`
- `||` – дизъюнкция, `true`, если хотя бы один операнд `true`

Логические операции

```
package main
import "fmt"

func main() {
    a := true
    b := !a           // false
    c := !b           // true

    fmt.Println("a:", a)
    fmt.Println("b = !a:", b)
    fmt.Println("c = !b:", c)

    fmt.Println("4 > 5 && 6 > 8:", 4 > 5 && 6 > 8)    // false
    fmt.Println("3 <= 5 && 10 > 8:", 3 <= 5 && 10 > 8) // true
    fmt.Println("4 > 5 || 6 > 8:", 4 > 5 || 6 > 8)      // false
    fmt.Println("3 == 5 || 10 > 8:", 3 == 5 || 10 > 8) // true
}
```

Операции присвоения

Операторы

```
package main
import "fmt"

func main() {
    var x int = 12 // простое присвоение
    y := 4

    x += y
    fmt.Println("x += y:", x)
// x += y: 16

    x -= y
    fmt.Println("x -= y:", x)
// x -= y: 12

    x *= 3
    fmt.Println("x *= 3:", x)
// x *= 3: 36

    x /= y
    fmt.Println("x /= y:", x)
// x /= y: 9

    x %= 7
    fmt.Println("x %= 7:", x)
// x %= 7: 2
}
```

Операторы присваивания устанавливают значение переменной.

Их можно использовать для присвоения значения новой переменной или изменения значения переменной путем вычисления любого определенного выражения.

Они сначала вычисляют выражение, заданное справа от оператора присваивания, а затем присваивают конечный результат переменной, записанной слева от оператора присваивания.

Условные конструкции

```
package main
import "fmt"

func main() {
    age := 20

    if age < 18 {
        fmt.Println("You are a minor")
    } else if age < 65 {
        fmt.Println("You are an adult")
    } else {
        fmt.Println("You are a senior")
    }
}
```

Используются для ветвления логики программы в зависимости от условий.

- Условие должно возвращать значение типа bool.
- Основные формы:
 1. if – выполняется блок, если условие истинно.
 2. if ... else – выполняется один из двух блоков.
 3. if ... else if ... else – несколько проверок с альтернативами.

Условные конструкции

```
package main
import "fmt"

func main() {
    day := 3

    switch day {
    case 1:
        fmt.Println("Monday")
    case 2:
        fmt.Println("Tuesday")
    case 3:
        fmt.Println("Wednesday")
    default:
        fmt.Println("Another day")
    }
}
```

Используется для проверки значения переменной против нескольких вариантов (case).

- Упрощает длинные цепочки if ... else if
- В Go не нужен break, переход к следующему кейсу по умолчанию не происходит.
- Для принудительного перехода к следующему кейсу используется ключевое слово fallthrough.

Условные конструкции

fallthrough позволяет перейти к выполнению следующего case, даже если условие не совпадает.

```
package main
import "fmt"

func main() {
    num := 2

    switch num {
    case 1:
        fmt.Println("One")
    case 2:
        fmt.Println("Two")
        fallthrough
    case 3:
        fmt.Println("Three")
    default:
        fmt.Println("Other")
    }
}
```

Циклы

```
package main
import "fmt"

func main() {
    for i := 1; i < 10; i++{
        fmt.Println(i * i)
    }
}
```

```
package main
import "fmt"

func main() {
    var i = 1
    for ; i < 10; i++{
        fmt.Println(i * i)
    }
}
```

Циклы позволяют в зависимости от определенного условия выполнять некоторые действия множество раз.

```
for [инициализация счетчика]; [условие]; [изменение счетчика]{
    // действия
}
```

```
package main
import "fmt"

func main() {
    var i = 1
    for ; i < 10;{
        fmt.Println(i * i)
        i++
    }
}
```

```
package main
import "fmt"

func main() {
    var i = 1
    for i < 10{
        fmt.Println(i * i)
        i++
    }
}
```

Циклы

Ряд типов данных представляют набор данных. Например, строка может быть представлена как набор символов. Кроме того, в Go есть комплексные типы, которые представляют коллекции данных, в частности, массивы, срезы, словари.

```
for индекс, значение := range набор_данных{
    // действия
}
```

```
package main
import "fmt"

func main() {
    str := "Hello"
    for index, value := range str {
        fmt.Println("Index:", index, " Value:", value)
    }
}
```

Циклы

Ряд типов данных представляют набор данных. Например, строка может быть представлена как набор символов. Кроме того, в Go есть комплексные типы, которые представляют коллекции данных, в частности, массивы, срезы, словари.

```
for индекс, значение := range набор_данных{
    // действия
}
```

```
package main
import "fmt"

func main() {
    str := "Hello"
    for index, value := range str {
        fmt.Printf("Index: %d, Value: %c\n", index, value)
    }
}
```

Циклы

Ряд типов данных представляют набор данных. Например, строка может быть представлена как набор символов. Кроме того, в Go есть комплексные типы, которые представляют коллекции данных, в частности, массивы, срезы, словари.

```
for индекс, значение := range набор_данных{
    // действия
}
```

```
package main
import "fmt"

func main() {
    str := "Hello"
    for _, value := range str {
        fmt.Printf("%c ", value)
    }
    fmt.Printf("%c ", 10)
}
```

Циклы

- *break* – полностью завершает выполнение текущего цикла.
- *continue* – пропускает оставшуюся часть тела цикла и переходит к следующей итерации.
- Помогают управлять потоком выполнения и избегать лишних действий.

```
package main
import "fmt"

func main() {
    for i := 1; i <= 5; i++ {
        if i == 3 {
            continue // пропускаем 3
        }
        if i == 5 {
            break // останавливаем цикл на 5
        }
        fmt.Println(i)
    }
}
```

Неформальный чат

По материалам, вопросам и предложениям

