Язык программирования

Обзор

расскажет Михаил В. Шохирев

Клуб программистов
Шадринск
2024-2025

О чём поGOворим

История: кем, когда, где создавался язык.

Цели: зачем создавался язык.

Особенности: чем Go отличается от других языков.

Компиляция (для разных платформ) и выполнение.

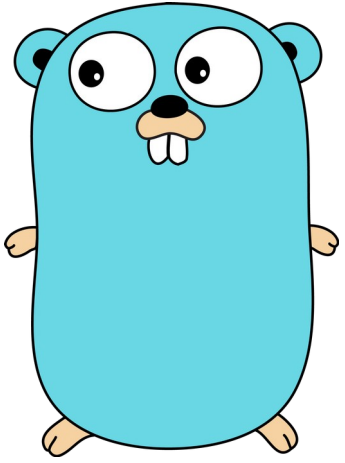
Синтаксис: правописание и стиль. Идиомы. Управляющие конструкции.

Модульность: функции, методы. Пакеты.

Мно*Go*задачность: goroutines, channels.

Инструменты: Go command. Go tools. IDE.

Применение: где, как и почему лучше использовать Go.



Шустрый суслик
Gopher

(автор: Renée French)

Go = "C for the 21st century"

Go — простой быстро компилируемый многопоточный язык программирования со статической типизацией, ориентированный на высокопроизводительную работу в сети и эффективное многопточное выполнение (в виде “родных” исполняемых файлов), легко осваиваемый, с многочисленными надёжными *стандартными* библиотеками.

Проектировщики:

Robert Griesemer *(разработчик V8 JS engine, Java HotSpot VM, Sawzall; его учитель – Niklaus Wirth)*
Rob Pike *(разработчик window system for Unix; Plan 9, Inferno, UTF-8, языка Sawzall)*
Ken Thompson *(разработчик Unix, C, grep, ed, Plan 9, Inferno, UTF-8)*

Разработчики:

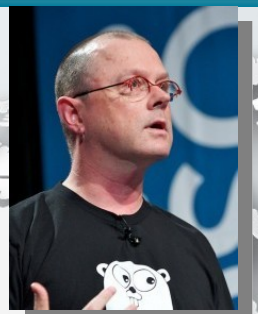
Команда в Google + community.

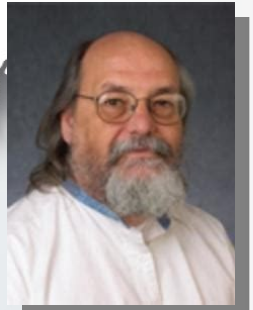
Цель: система программирования для разработки больших надёжных высоконагруженных быстро работающих серверных программных комплексов с распараллеливанием выполнения.

На проектирование Go **повлияли языки** *C, Oberon-2, Active Oberon, Oberon, Modula-2, Modula, Pascal, Alef, Newsqueak, Squeak, CSP, Smalltalk, Limbo, APL, BCPL, occam.*

Разработчиков первоначально объединило их общее недовольство языком C++. Кроме того, они хотели сделать язык с простым синтаксисом, но по современным требованиям.

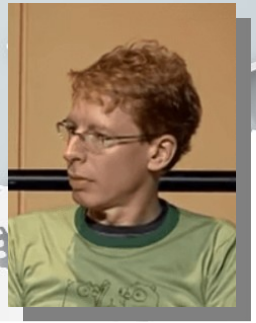
Go: creators





Ken Thompson, *США* («старая инженерная школа»): *разработчик языка C, ОС Unix, Plan 9, Inferno, grep, ed, QED, UTF-8.*

Rob Pike, *Канада* («следующее поколение», специалист по concurrency): *разработчик window system for Unix; ОС Plan 9, Inferno, UTF-8, sam, acme, языков Sawzall, Limbo, Newsqueak.*



Robert Griesemer, *Швейцария* («ученик Никласа Вирта, европейская школа»): *разработчик V8 JS engine, Java HotSpot VM, языка Sawzall, a programming language for vector computers, системы Strongtalk.*

Go: цели создания языка

Разработчики хотели не просто создать новый язык, но спроектировать технологически систему программирования :

* язык, который заставляет программистов делать меньше ошибок;
* удобная система подключения внешних пакетов
*

Хронология и совместимость

2007.09	началась **разработка** Go в компании Google; проектированием занимались: Robert Griesemer, Rob Pike и Ken Thompson.
2009.11.10	был официально **представлен** язык Go.
2011.03.16	go r56: based on release weekly.2011-03-07.1
2012.03.28	**go1.0**: language & a set of core libraries.
2013.05.13	go1.1
2015.08.19	go1.5: compiler & runtime written entirely in Go (with a little assembler).
2017.08.24	go1.9: type aliases.
2020.02.25	go1.14: Go modules.
2022.03.15	go1.18: generics.
2023.08.08	go1.21: min, max, clear built-in functions.
2024.10.01	go1.**23**.2: range over function types; *iter, unique* packages.
2025.02.11	go1.24.0: generic type aliases; runtime have decreased CPU overheads by 2–3%.
2025.04.01	go1.**24**.2: security & bug fixes.

Спецификация языка и стандартой библиотеки обратно своместимы с версиями Go 1.x.

Поэтому многие крупные компании, выждав паузу, убедились в долговременной поддержке языка и стали применять его в своих важных проектах.

Распространение

Go **применяется** в:

Alibaba, Amazon, AmericanExpress, Armut (*C# → Go*), BBC, bitly, Canonical, Capital One, CERN, Cloudflare, CockroachDB, Docker, DropBox *(Python → Go)*, GitHub, Google, gov.uk, IBM, InfluxDB, Intel, Kubernetes, Meta, Microsoft, Monzo, Mozilla (*Rust & Go*), Netflix (*Java → Go*), New York Times, Oracle, PayPal, Slack, Salesforce *(Python, C → Go)*, SendGrid, Stream *(Python → Go)*, SoundCloud, Terraform, The Economist, Twitch, Twitter, Uber, Walmart, YouTube, *многих других организациях и проектах open-source.*

В **России**:

Яндекс, ЦУМ, УГМК-Телеком, Точка, Т-Банк, Совкомбанк Технологии, СберТех, Ростелеком, Онлайн-кинотеатр Иви, МТС, Магнит.Tech, Лаборатория Касперского, ИТ-Холдинг Т1, ИНГОССТРАХ Банк, Домклик, Группа Астра, ГНИВЦ, АйТи Инновация, YADRO, X5 Digital, VK, Viasat Tech, Tutu, Tele2, Selectel, S8.Capital, Ozon, Okko, Mail.ru, Lamoda Tech, iSpring, IBS, Cloud.ru, Boxberry: IT, Beeline, Avito, 2GIS *и многие другие...*

Популярность

TIOBE index ▲ (since 2009):
- Now: **#7** (Apr 2025) ← #13 (Nov 2023)
- Highest Position (before): #7 (Apr 2024)
- Lowest Position: #122 (May 2015)
- Language of the Year: 2009, 2016

GitHut 2.0 ▲ stars: **#3** pulls: #3, pushes: #7 (Q1 2024)
IEEE Spectrum ▲ Top Programming Languages: **#8** (2024)
Crossover ▲ Top 10 In-Demand Programming Languages for 2025: **#10**
ZDnet ▲ The most popular programming languages in 2025: **#10**
StackOverflow ▲ **#11** Most popular techs: language (professionals) (2024)
PYPL ▲: **#12** (A[r 2025)
RedMonk ▲ Programming Language Rankings: **#12** (Jun 2024)
Statista ▲ Most used programming languages among developers (2024): **#12**
GeeksForGeeks ▲ 20 Best Programming Languages to Learn in 2025: **#13**

** я не учитывал в индексах позицию HTML/CSS*

Установка

Реализации:

1. Официальный компилятор (Google) для операционных систем AIX, Android, *BSD, iOS, Linux, macOS, Plan 9, Solaris, Windows (на разных аппаратных архитектурах) и для WebAssembly (WASM).
2. gofrontend + libgo для GCC и других компиляторов.
3. **TinyGo** для embedded systems и WebAssembly.
4. **GopherJS** – кросс-компилятор из Go в JavaScript.

Поддерживаются **архитектуры** i386, amd64, ARM, RISC-V, MIPS, ppc64, …

```
go tool dist list
```

Лёгкая кросс-компиляция!

Установка (описание `https://golang.org/doc/install`):

```
sudo apt-get install golang
```

The Go **Playground** ~ интерактивное выполнение программ в браузере:

`https://go.dev/play/`

Пример *с приветом*

```
package main                          // все программы принадлежат к своему пакету
import (                              // подключить...
  "fmt"                               // ... пакет форматированного вывода
  "os"                                // ... и взаимодействия с OS
)
const world = "世界"                   // UTF-8 для исходного кода и литералов

func main() {                         // с main() начинается выполнение программы
  var s string = world                // var переменная тип = значение
                                      // len() - встроенная функция определения размера
  if len(os.Args) > 1 {               // в os.Args[0] – имя программы
    s = os.Args[1]                    // имена с заглавной буквы экспортируются
  }
  fmt.Printf("Привет, %s!\n", s) // вызов функции из импортированного пакета
}
```

```
$ go run helloWorld.go
Привет, 世界!
$ go build helloWorld.go
$ ./helloWorld мир
Привет, мир!
$ GOOS=windows GOARCH=amd64 go build helloWorld.go
$ ls helloWorld*
helloWorld helloWorld.exe helloWorld.go
```

Пример: *web-server*

```
package main                                // на основе примера, который привёл Rob Pike

import (                                    // импортировать пакеты:
    "fmt"                                   // форматированная печать
    "log"                                   // протоколирование
    "net/http"                              // сетевая магия – в этом пакете
)

                          // во всех идентификаторах можно использовать символы UTF-8
func こんにちは Мир(w http.ResponseWriter, req *http.Request) {
    fmt.Fprintf(w, "Привет, мир!!!\n")            // вывод ответа прямо в сеть 8-)
}                                                  // точнее: в любой Writer

func main() {
    http.HandleFunc("/", こんにちは Мир)           // подключить функцию-обработчик
    // слушать порт на хосте
    log.Fatal(http.ListenAndServe("localhost:12345", nil))
    // обработать ошибку и// записать в протокол
}                                                        // бесконечный цикл
// аргумент nil означает, что надо использовать HandleFunc
```

Пример: *кириллица*

```
package main
import (
    "fmt"
    "time"
)
const Вопрос = "Главный вопрос жизни, вселенной и вообще"
const Ответ = 42

func Мыслитель(вопрос string) int {
    return Ответ
}

func main() {
    var время_работы = 7_500_000
    окончание := time.Now()
    начало  := окончание.AddDate(-время_работы, 0, 0) // + (годы, месяцы, дни)
    ответ := Мыслитель(Вопрос)
    fmt.Printf(
        "Начав в %v и проработав %v лет, Мыслитель в %v дал ответ: '%v'.\n",
        начало, время_работы, окончание, ответ)
}
```

Особенности

Простой синтаксис. Минимум синтаксических конструкций.
Каждое утверждение (statement) начинается с ключевого слова.

В языке нет исключений (exceptions). Ошибки – это тип данных error.
Нет классов, но в struct можно описывать поля, и для всех типов данных можно определять методы.
Есть интерфейсы (абстрактные наборы действий), и типы могут неявно удовлетворять этим интерфейсам.
`defer` ~ отложенное исполнение функции: появилось в Go.

`;` служит переводом строки (line feed).
`,` запятая иногда обязательна в конце строки!
_ “пустая переменная“ (blank identifier) для игнорирования значения.
`:=` простое объявление (with inferred type) и инициализация переменной.

Имена с заглавной буквы (Capitalized) экспортируются (видны вне пакета).
Все имена с маленькой буквы видны во всех файлах внутри пакета.

Функции: multiple return values, named return values, bare return, variadic functions (с переменным списком параметров).
`func init() { }` // инициализирующие функции в файлах пакета.

Безтиповые константы (untyped constants).
`rune` // тип данных для ”символа” (code point) в кодировке UTF-8
`iota` // перечисление (enumerator) именованных целых значений

variable declared / package imported but not used – не скомпилируется!

Пунктуация

`()` список: импортов, констант, параметров, возвращаемых значений, ...	`import ( "fmt" )`
	`const ( answer = 42 )`
	`func f(x float64) float64 { return 0.0 }`

`[]` размер массива, показатель среза элемент массива, среза, словаря	`array [4]int;  slice []string` `array[i];  slice[i];  map[key]`

`{}` блок определения
`{}` блок начальных значений
`{}` блок кода
	`type struct Point { x, y int32 }`
	`ipAddress = [4]int{127, 0, 0, 1}`
	`func answer() int { return 42 }`

`:` отделяет индекс или ключ от значения	`gender := [2]string{ 0:"Female", 1:"Male" }` `language := map[string]int { "Go": 2007 }`
`:` ставится после метки	`label:`

`...` размер массива вычисляется по значениям
`...` список параметров переменной длины
`...` список аргументов переменной длины
	`shadrinsk := [...] float32 { 56.05, 63.38 }`
	`func sum(numbers ...int) (sum int) { /* range numbers */ }`
	`integers := []int{1,2,3,4,5}; sum(integers...)`

`;` перевод строки (разделитель утверждений)	`version := 1.0; released := 2012`
`:=` объявление и присваивание	`site := "https://go.dev/"`
`,` разделитель в списке	`place = findLocation(latitude, longitude)`
`.` разделитель объекта и метода	`result = object.method()`

Управление выполнением: ветвление

```
if условие {
    // код
} else {
    // код
}
```

```
if выражение; условие {
    // код
} else {
    // код
}
```

```
switch выражение {
    case значение-1, значение-2:
        // код
        fallthrough
    case значение-N:
        // код
        break
    default:
        // код
}
```

```
switch {                //
    case условие-1:
        // код
    case условие-2:
        // код
}
```

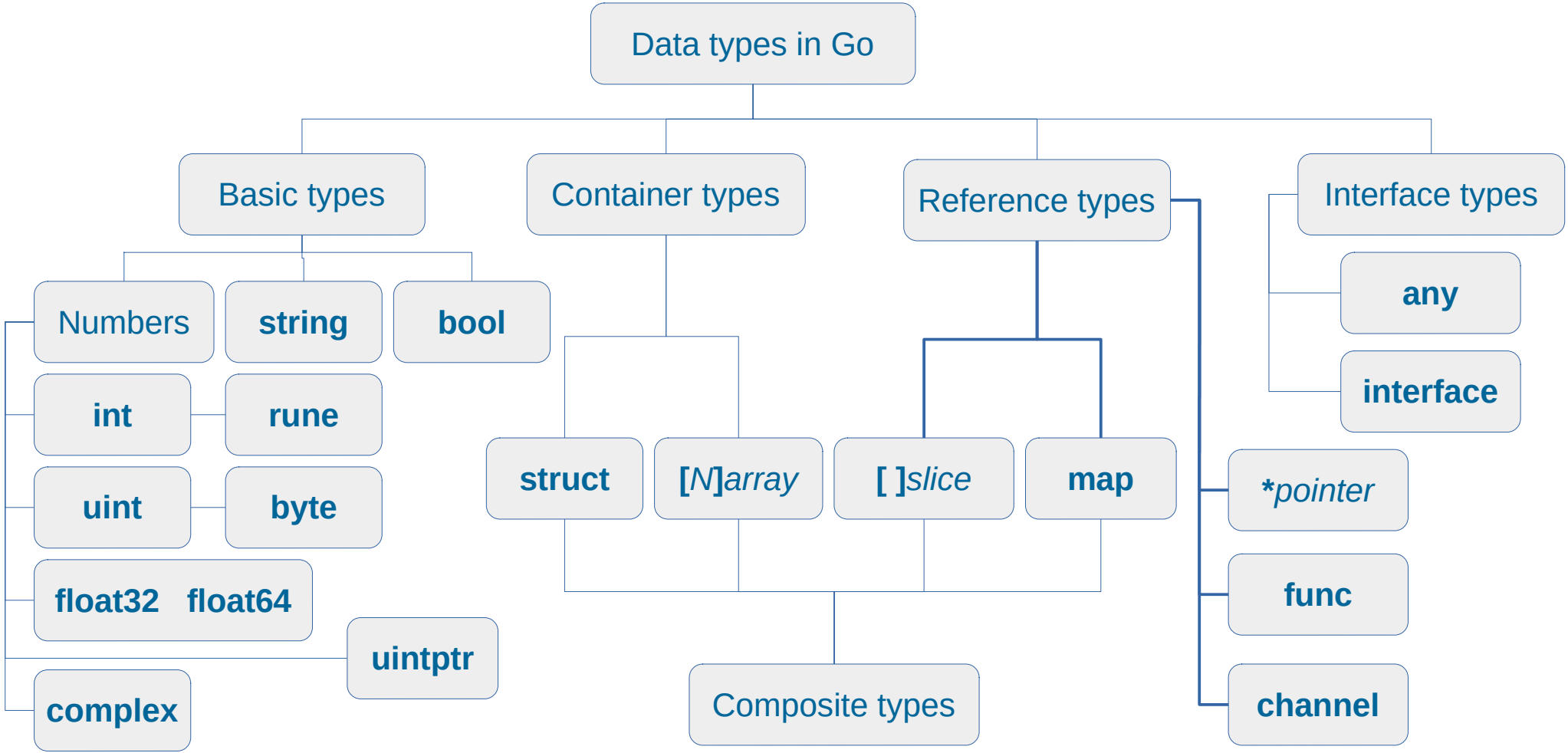
«Всё уже придумано до нас!»

Управление выполнением: циклы

```
метка:
for инициализация; условие; изменение {      // классический итерационный
  // break метка
  // continue
}

for условие {                                 // == while
    // код
}

for индекс := range число {                   // перебор целых чисел
    // код
}

for индекс, элемент := range коллекция {      // перебор array, slice или channel
    // код
}

for ключ, значение := range map {             // перебор хэша
    // код
}

for {                                         // бесконечный цикл
    // код
}
```

*«One **for** to rule them all.»*

Типы данных


Data types in Go
Basic types
Container types
Reference types
Interface types
Numbers
string
bool
int
rune
uint
byte
float32 float64
uintptr
complex
struct
[N]array
[ ]slice
map
Composite types
*pointer
func
channel
any
interface

Константы: const

Предопределённые константы (predefined constants):

```
    true false nil iota
```

Безтиповые константы (untyped constants:

```
    const constant = expression
    const constant1, constant2 = value1, value2
    const ( constant1 = value1, constant2 = value2 )

    const (
        e  = 2.71828182845904523536028747135266249775724709369995957496696763
        pi = 3.14159265358979323846264338327950288419716939937510582097494459
    )
```

Константы с заданным типом (typed constants):

```
    const constant Type = expression

    const b byte = 0Xf                      // байт
    const x complex128 = 2+5i               // комплексное число
    const Big float64 = 1 << 100            // binary 1 with 100 zeros
    const i int32 = -273                    // целое
    const Go rune = '碁'                    // символ
    const language string = "Go"            // строка
```

Операции: объявление и присваивание

```
// объявление новой переменной
var переменная Тип


// объявление новой переменной и присваивание значения
i := 0                  // тип переменной выводится из присваиваемого значения
t, f := true, false     // параллельное присваивание



// присваивание значения уже объявленной переменной
s = ""                  //
G, o = "G", "o"         // tuple assignment
i, j = j, i             // обмен значений i и j
```

Типы данных: struct

struct ~ структура: набор разнотипных полей, аналог класса

```
type Structure1 struct {                    // объявление типа
    field1 Type1
    field2, field3 Type2
}

s := Structure1 {                           // объявление переменной и
    field1: initial_value1_of_Type1,        // инициализация полей значениями
    field2: initial_value2_of_Type2,        // по именам (не всех) полей
}

var s2 Structure1                           // объявление переменной
s2 = s1

// объявление переменной и инициализация полей значениями по порядку
s3 := Structure1{s2.field1, s2.field2, s2.field3} // следования полей

var s4 Structure1 = Structure1{}            // инициализация пустой структурой
```

Ссылочные типы: reference types

```
pointer ~ указатель:
        p := *Type

slice ~ динамический массив:
        s := []Type

map ~ хэш | ассоциативный массив | словарь:
        m := map[KeyType]ValueType = { key: value }

function ~ функция:
        func f() { /* код */ }

channel ~ канал:
        chan ch
```

Контейнерные типы: container types

```
[] ~ массив (array):
        a := [размер]Type

slice ~ динамический массив:
        s := []Type

map ~ хэш | ассоциативный массив | словарь:
        m := map[KeyType]ValueType = { key: value }
```

Функции: func

```
// функция без возвращаемого значения = процедура
func f1(param Type) { /* ... */ } // no return value
    f1(arg1)

// функция с одним возвращаемым значением
func f2(p1, p2 Type1, p3 Type3) ReturnType { /* ... */ }
    result = f2(a1, a2, a3)

// функция с несколькими возвращаемыми значениями: возможно, именованными
func f3(p1 T1, p2 T2) (returnValue1 T1, rv2 RT2) { /* ... */ }
    (result1, result2) = f3(a1, a2)

// функция с переменным списком параметров
func f4(p ...T) ReturnType { /* ... */ } // variadic
    result = f4([]slice)
```

Методы: func (t Type)

К любому типу данных можно присоединить поведение с помощью методов.

```
type Temperature float32
func (t Temperature) Celsius() Temperature { return (t - 32.0) * (5.0 / 9.0) }
var f Temperature = 37.0
c := f.Celsius() // 2.777778

type Album struct { name, artist string; year, length int }
type TapeRecorder struct {
    //
}
func (r TapeRecorder) play(a Album) {
    fmt.Printf("Playing album '%s' by '%s' from tape for %d minutes...\n",
    a.name, a.artist, a.length)
}

recorder := TapeRecorder{/* начальные значения */}
recorder.play(album)
```

Интерфейсные типы: interface types

interface – это тип данных, у которого может быть набор методов

```
type interfaceName interface{}                    // пустой интерфейс == any

type Flyer interface { fly() string }             // 1-й интерфейс с методом
type Swimmer interface { swim() string }          // 2-й интерфейс с методом
// все типы, которые реализуют метод fly(), будут удовлетворять типу Flyer

type Bird struct { Name string }                  // пользовательский тип Bird
func (b *Bird)fly() string {                      // удовлетворяет интерфейсу Flyer
    return "flying..."
}
type Penguin struct { Name string }               // пользовательский тип Penguin
func (f *Penguin)swim() string { return "swimming..." } // удовлетворяет сразу
func (b *Penguin)fly() string { return "can't fly!" }   // двум интерфейсам

    var s = Bird{"Sparrow"}
    var p = Penguin{"Gentoo"}
    birds := []Flyer{&s, &p, &Bird{"Dove"}}// Flyer – это тип данных
    for _, b := range birds {                 // polymorphism
        fmt.Println(b, b.fly())
    }
```

МноGoзадачность

CSP (communicating sequential processes)

Goroutines
Channels

Каналы: channel

channel ~ канал:

```
        chan ch
```

Инструменты: go command

```
    go <command> [arguments]
The commands are:
    bug         start a bug report
    build       compile packages and dependencies
    clean       remove object files and cached files
    doc         show documentation for package or symbol
    env         print Go environment information
    fix         update packages to use new APIs
    fmt         gofmt (reformat) package sources
    generate    generate Go files by processing source
    get         add dependencies to current module and install them
    install     compile and install packages and dependencies
    list        list packages or modules
    mod         module maintenance
    work        workspace maintenance
    run         compile and run Go program
    telemetry   manage telemetry data and settings
    test        test packages
    tool        run specified go tool
    version     print Go version
    vet         report likely mistakes in packages
```

Инструменты: IDEs

▲ IDEs And Text Editor Plugins @ go.dev:

- **Visual Studio Code** + plug-in (Microsoft)
- **GoLand** (JetBrains)
- **LiteIDE** (open source and cross-platform Go IDE)
- **jEdit** (open-source, cross-platform text editor: Java)
- **Komodo IDE** (cross-platform IDE with built-in Go support)
- **Komodo Edit** + plug-in (cross-platform text editor)
- **Geany** (cross-platform programmer's text editor)
- **Notepad++** (text & source code editor: Windows)
- **Kate** (cross-platform text editor with Go support out-of-the-box: KDE)
- **Sublime Text** (commercial text editor: macOS, Windows, Linux)
- **TextMate** (commercial text editor: macOS)

Применение

Области применения ▲ Go:

- DevOps & SRE ▲ (Development Operations & Site Reliability Engineering)
- Cloud & Distributed Network Services ▲
- Web Development ▲ (frameworks, toolkits, engines, servers)
- System Automation & CLIs ▲, Utilities & Stand-Alone Tools

Software на Go:

Allegro *(eCommerce)*, AmneziaWG, **AKS** [Azure Container Service] @ Microsoft, AresDB @ *Uber*, Buffalo *(web framework)*, Caddy *(web server)*, CockroachDB, Digger *(IaC)*, **Docker**, Drone *(CD)*, **DropBox** *(backend)*, ent @ *Meta*, Flamingo *(web framework)*, Gin *(web framework),* Gitlab Runner, **Google Cloud**, Gorilla *(web toolkit)*, **Grafana**, Hugo *(website engine)*, InfluxDB, JuiceFS, **Kubernetes**, LXD @ *Canonical*, Mattermost *(messaging platform),* Monzo *(banking app)*, **Ollama** (89%), **PayPal**, **Prometheus** *(monitoring & alerting toolkit)*, Rend *(large scale data caching @ Netflix)*, **SoundCloud**, Soundscape *(music streaming server)*, Terraform *(IaC)*, Timesheets *(project management),* Twitch *(live-streaming)*, VITESS @ *YouTube*, **Zabbix** agent2, …

Top 60+ Open-source Apps Written with Golang in 2024 ▲

TinyGo

Мои впечатления от Go

Синтаксис простой, но с некоторыми непривычными конструкциями.
Логично спроектирован, предсказуем. Исходники хорошо понимаются.
Непривычно после Ruby: все объявления и преобразования надо делать явно.

Низкоуровневый, как Си – вспомнил молодость!
Очень быстро компилируется.
Легко скомпилировать исполняемую программу для другой платформы и ОС.
Действительно очень быстро выполняется.

Очень строгий: переменная не используется – код компилироваться не будет!
Strong typing и другие строгости важны для надёжности больших приложений.
Явная работа с ошибками дисциплинирует программиста.

Много *стандартных* библиотек – на все случаи жизни.
Хорошая документация на библиотеки (с исполняемыми примерами).
Легко подключать сторонние модули.
Много сайтов с примерами – изучать легко.

Наверное, это последний язык, разработанный «классиками», которые создали Unix.

Книг много *(лучше читать на английском)*

Ссылки ▲

- **go.dev/** // Официальный сайт языка
- **go.dev/play/** // Go Playground ~ выполнение в браузере
- **go.dev/ref/spec** // **Спецификация языка**
- **go.dev/doc/** // Документация
- **go.dev/doc/code** // How to Write Go Code
- **pkg.go.dev/std** // стандартная библиотека
- **gobyexample.com** // Go в примерах
- **go.dev/doc/modules/layout** // Структура каталогов
- **github.com/golang-standards/project-layout** // Стандартный макет [большого] Go проекта

- **tour.golang.org** // Экскурсия по возможностям Go
- **golangdocs.com** // примеры конструкций
- **appliedgo.net/why-go/** // 15 Reasons I Love Go
- **awesome-go** // libraries for everything...

- **go.dev/doc/effective_go** // "Effecive Go" бесплатная web-книга
- **gopl.io** // "The Go Programming Language" by A.A.A.Donovan & B.W.Kernighan

- **w3schools.com/go/** @ w3schools // Справочник
- **Самоучитель по Go для начинающих** @ proglib.io // Самоучитель
- **Дорожная карта Go-разработчика** @ proglib.io // План изучения
- **lyceum.yandex.ru/go** // Яндекс-лицей: Программирование на Go
- **Книги по Go** @codelibs.ru //

- **tinygo.org** // TinyGo

Готов ответить на вопросы

Ссылка на презентацию

???

Словарик

CSP = communicating sequential processes
gopher ~ программист на Go