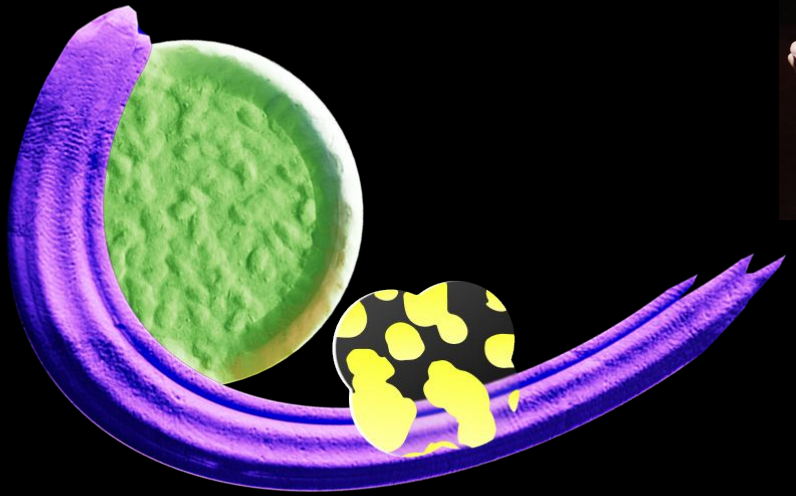
avito.tech Saint Petersburg | 2025

I. Go Fundamentals

language, env and all

Denis Zakharov
Engineer

Speaker



Denis Zakharov
Engineer

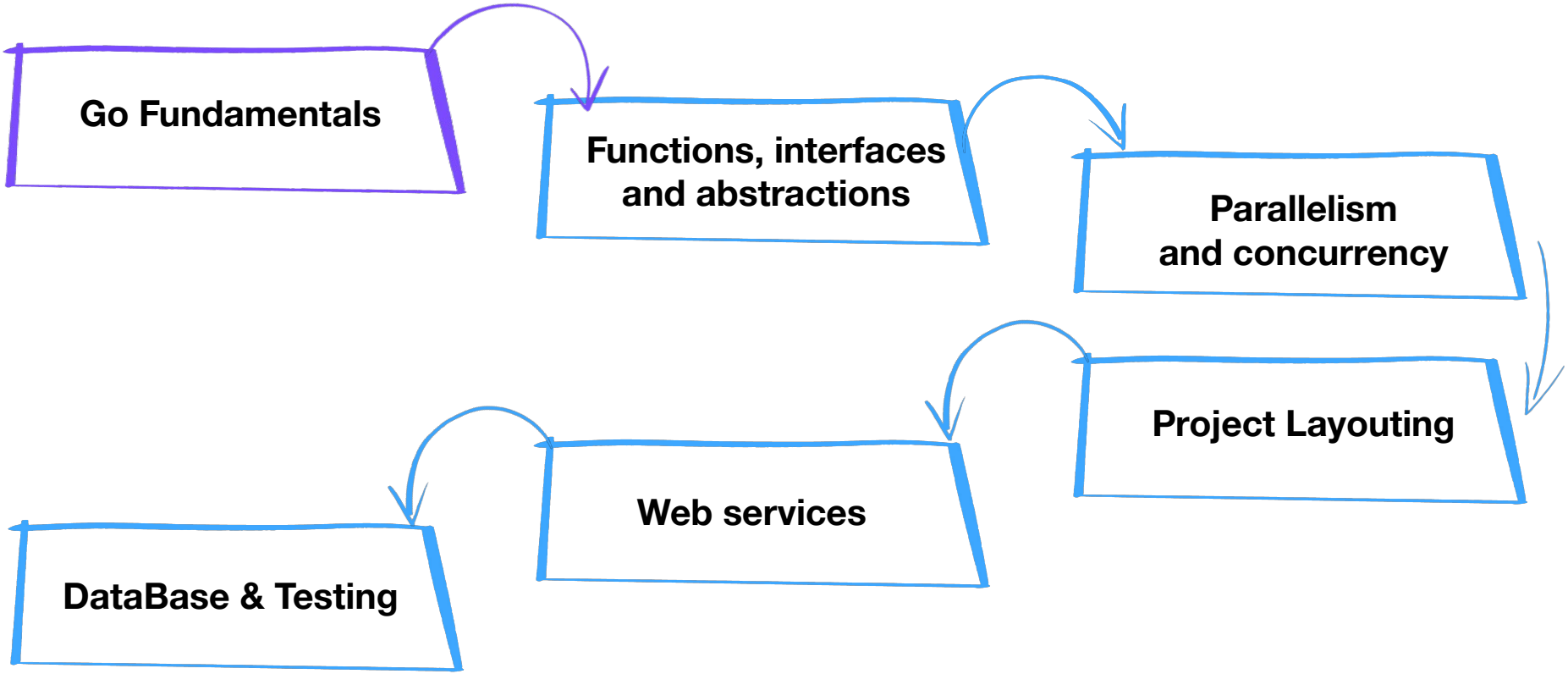
BCS & MCS n what; SE, SRE, WEB3 and Machine Cringe Learning Engineer

Using Golang since 2020

Условия успешного прохождения дисциплины

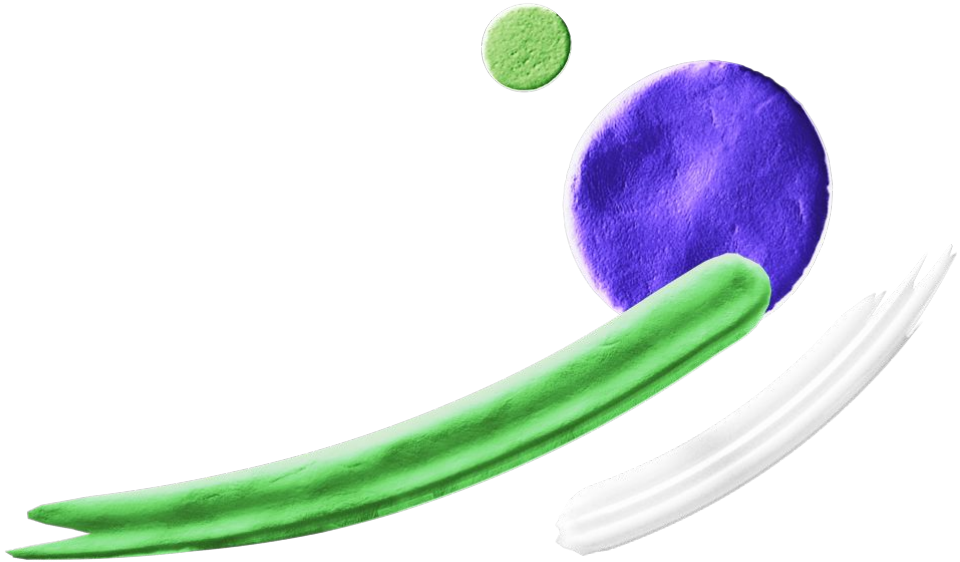
1. Посещение лекционных и лабораторных занятий — min 50% от всего количества.
2. Активное участие на занятиях — доп. баллы
3. Соблюдение дедлайнов сдачи лабораторных работ — можно сдать позже дедлайна на 3 дня и получить штраф -1 балл. Пересдать лабу можно в течение недели после проверки.
4. Самостоятельные работы выполняются самостоятельно

Course Structure



Today we will learn how2

1. Install Go
2. Create a Go application
3. Declare basic data types
4. Use functions



Agenda

1 Go Intro

2 Program Structure

3 Variables

4 Consts & iota

5 Control flow

6 Functions

7 Go set up

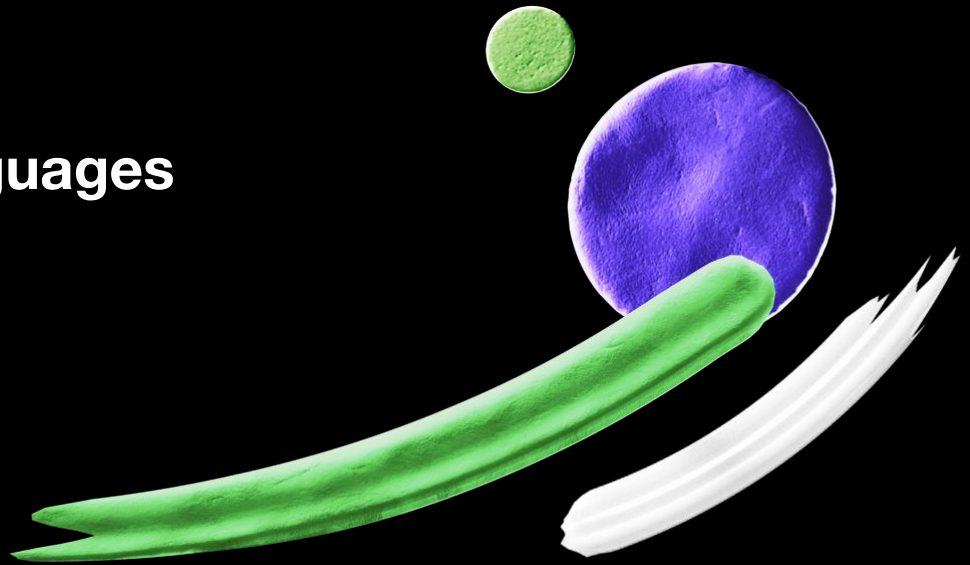
Memes Pre Hook

I. Intro

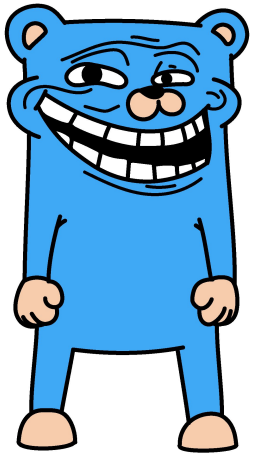
01 History

02 Language goals

03 Differences from other languages



Google language



It was designed at Google in 2007 by Robert Griesemer, Rob Pike, and Ken Thompson, and publicly announced in November of 2009.

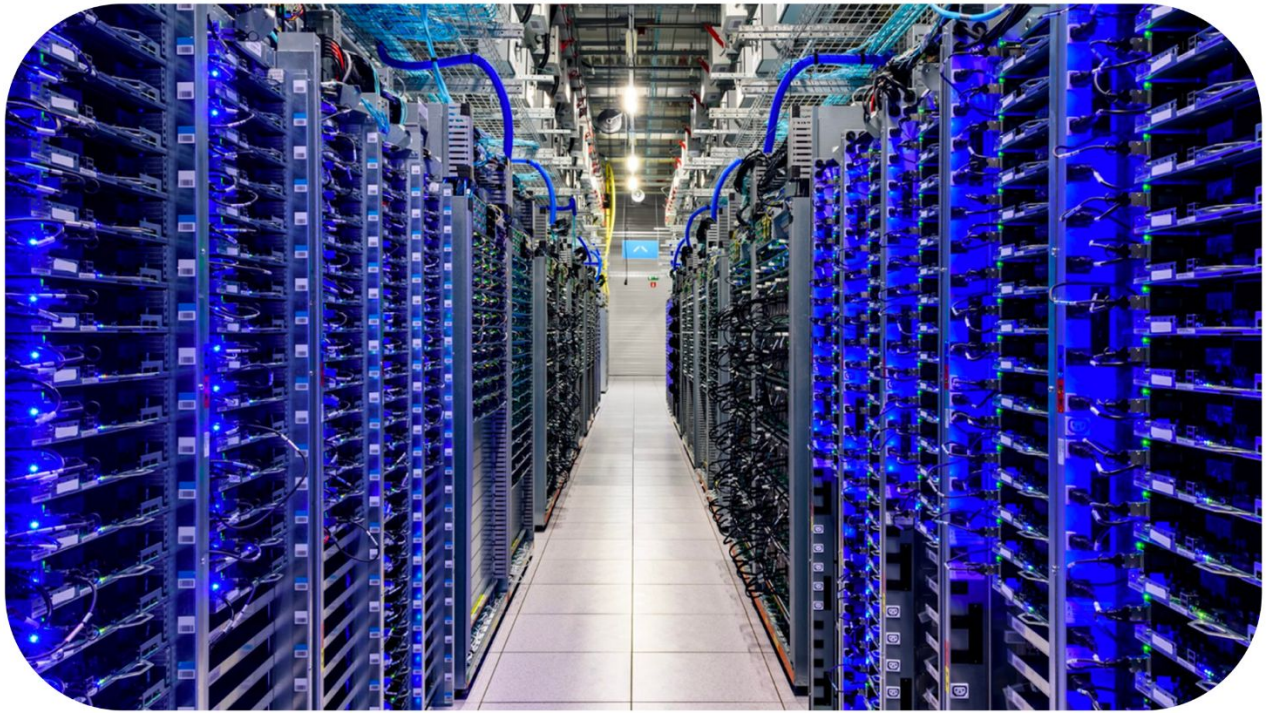
Google language

Rob Pike

Ken Thompson

Rob Pike

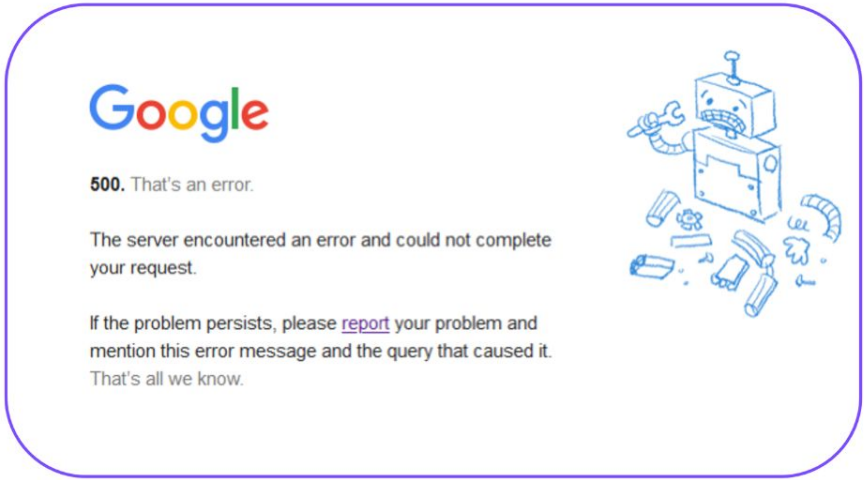
It’s all started



8 Layer of Security @ Google Cloud DCs

Google language

- slow builds
- uncontrolled dependencies
- each programmer using a different subset of the language
- poor program understanding
- duplication of effort
- cost of updates
- version skew
- difficulty of writing automatic tools
- cross-language builds



Or It's Just another Java server

Constraints

- It must work at scale
- The need to get programmers productive quickly
- It must be modern

Go Design

- Dependencies — forces to think earlier about a larger-scale issue
- Communicating Sequential Processes
- Garbage collection — tools to limit allocation by controlling the layout
- Composition not inheritance — satisfaction is statically checked at compile time
- Errors are just values

JFYI

Go is released every six months. Each major Go release is supported until there are two newer major releases. Critical problems are fixed by issuing minor revisions.

golang/**go**

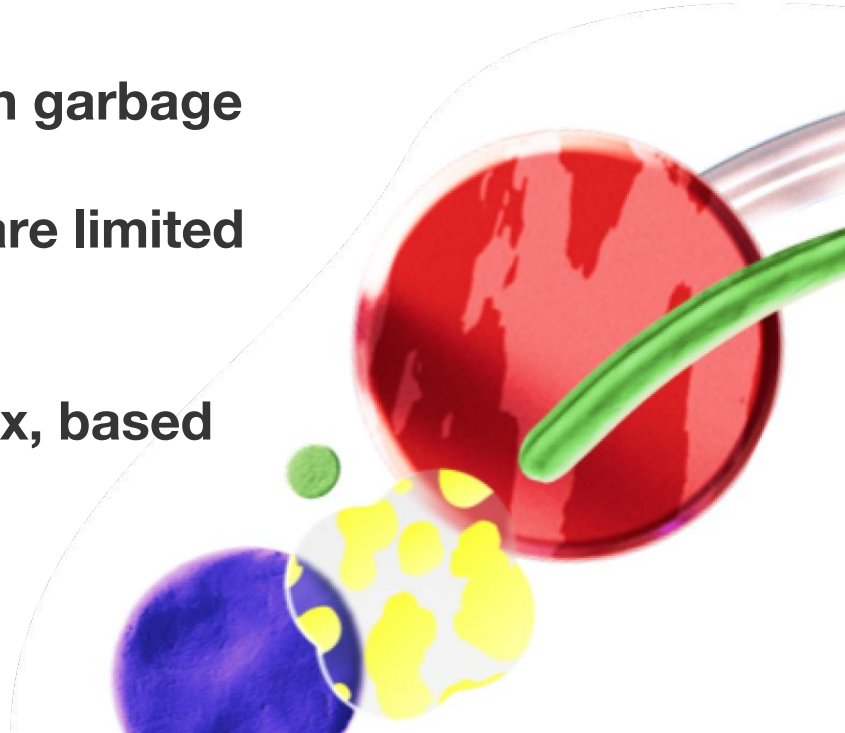
The Go programming language

2k Contributors | 78 Used by | 29 Discussions | 129k Stars | 18k Forks

0,1% of lang (syscalls) written in Perl

Features

- Functional programming tools
- Automatic memory management with garbage collector.
- Object-oriented programming tools are limited to interfaces.
- Means of parallel programming.
- Sufficiently concise and simple syntax, based on C.



Interaction

How do you think what are pros/cons of Go & why?

Go vs Rust vs Nim vs Zig

	Go	Rust	Nim	Zig
Performance	Slowest	Best	>>Best	>>Best
Concurrency	Best	Powerful	Good	Basic
Memory	GC	Ownership	GC (opt)	Basic
Compilation	Fastest	Slowest	Fast	Moderate
Libraries	Best	Growing	Small	Smallest
Complexity	Simplest	Hardest	Moderate	Moderate

Which companies use Golang?

¿Quieres?

Where do we write code?

GoLand

Zed

VS Code

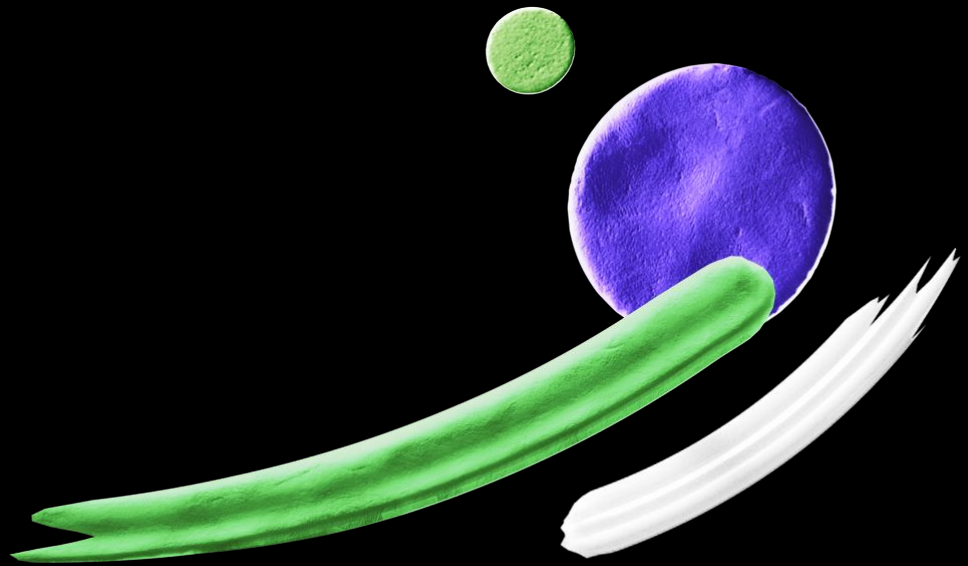
II. Program structure

01 Who is package main

02 How 2 import

03 Func main()

04 Go build, go run



program demo

Go program structure

- % go mod init hello — command initializes a new Go module named "hello" in the current directory.
- Creates go.mod file:
  This command creates a go.mod file in the directory where it is executed. This file is central to Go module management.
- Defines module path:
  The go.mod file will contain a line like module hello, indicating that the current directory and its contents belong to the "hello" module.

Go program structure

```
package main
```

signifies that a package is an executable program

```
import "fmt"

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

```
% go run .
```

Go program structure

```go
package main

import "fmt"

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

import keyword is used to include external packages in your source code

```
% go run .
```

Go program structure

```
package main

import "fmt"

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

serves as the entry point for an executable program

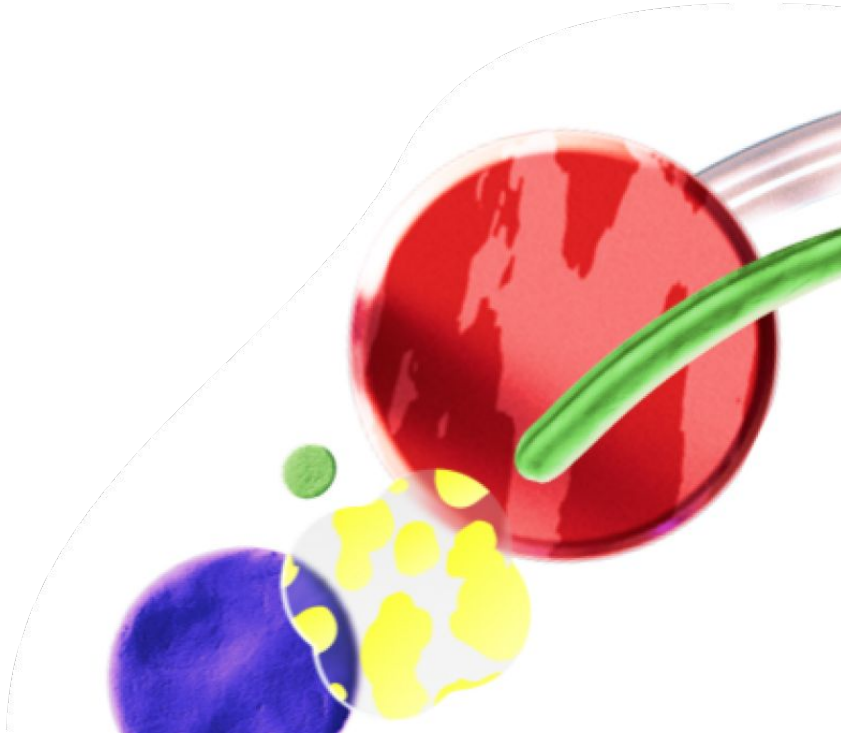
```
% go run .
```

What actually happens when go build

- Lexical Analysis – splits code into tokens.
- Parsing – builds an AST (Abstract Syntax Tree).
- Semantic Analysis – checks types and context.
- Intermediate Code (SSA) – optimizes the code.
- Machine Code Generation – produces an executable.

Does everyone have an editor?

Sandbox — https://go.dev/play/

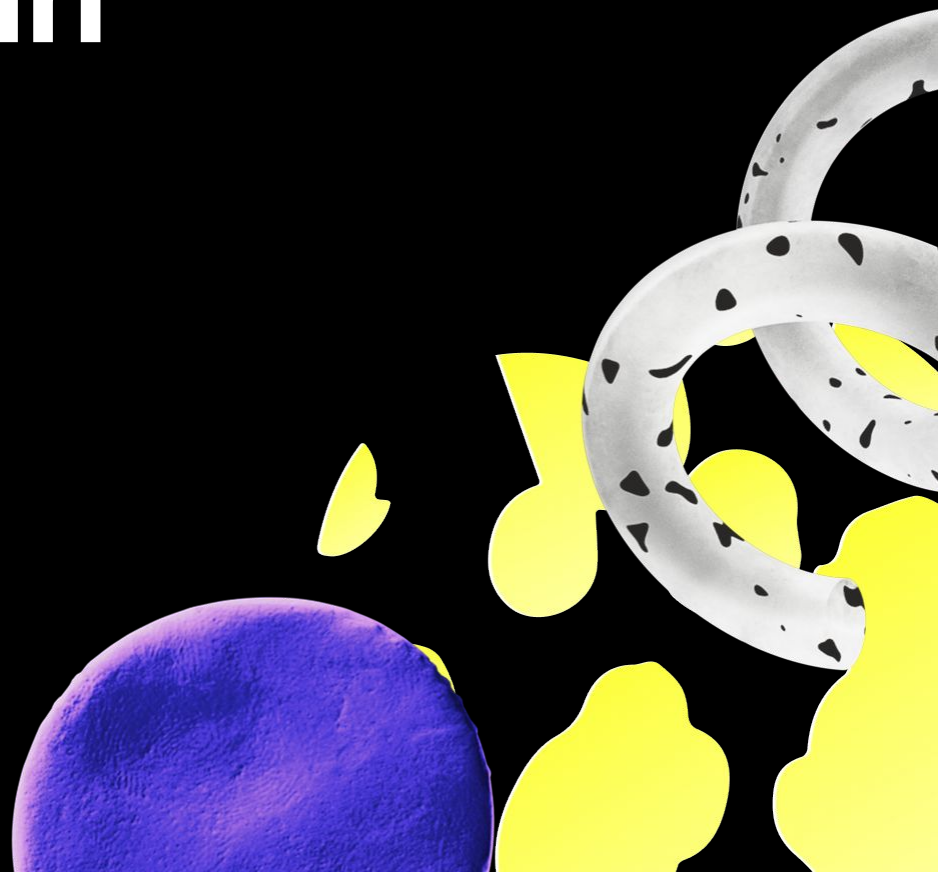


Hands Up 5 min

Write your own Hello World

Solve in:

- Sandbox — https://go.dev/play/
- Code Editor of your choice



Solution Leak

```go
package main

import "fmt"

func main() {
   fmt.Println("hello world")
}
```

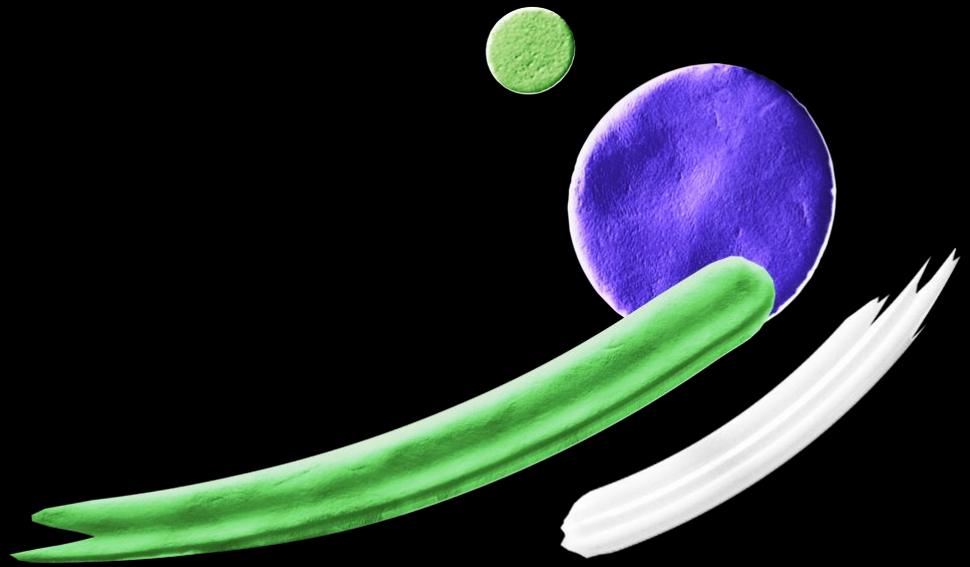
- Write code
- go build hello-world.go
- ./hello-world.go

III. Variables

01 Var, :=

02 Data types

03 Zero values



What is a variable?

??? basic types demo

Basic Types

Data Type	Size (bytes)	Description
`bool`	1	(true/false)
`int8`, `uint8`	1	8-битные целые (byte - алиас uint8)
`int16`, `uint16`	2	16-битные целые
`int32`, `uint32`	4	32-битные целые (rune - алиас int32)
`int64`, `uint64`	8	64-битные целые
`int`, `uint`	4 или 8	Зависит от архитектуры (32/64 бит)
`uintptr`	4 или 8	Для хранения указателей
`float32`	4	32-битное число с плавающей точкой
`float64`	8	64-битное число с плавающей точкой
`complex64`	8	Комплексное (2 float32)
`complex128`	16	Комплексное (2 float64)
`string`	16	Строка (8 байт указатель + 8 байт len)

Variables

```
var p int = 1000
var a int
a = 5
var (
   k int
   f int
)
b := 50
c, _, e := 12, 123, 900
```

full variable declaration syntax

Variables

```
var p int = 1000
var a int
a = 5
var (
    k int
    f int
)
b := 50
c, _, e := 12, 123, 900
```

declare a variable
and fill it with value

Variables

```
var p int = 1000
var a int
a = 5
var (
    k int
    f int
)
b := 50
c, _, e := 12, 123, 900
```

declare a block with variables

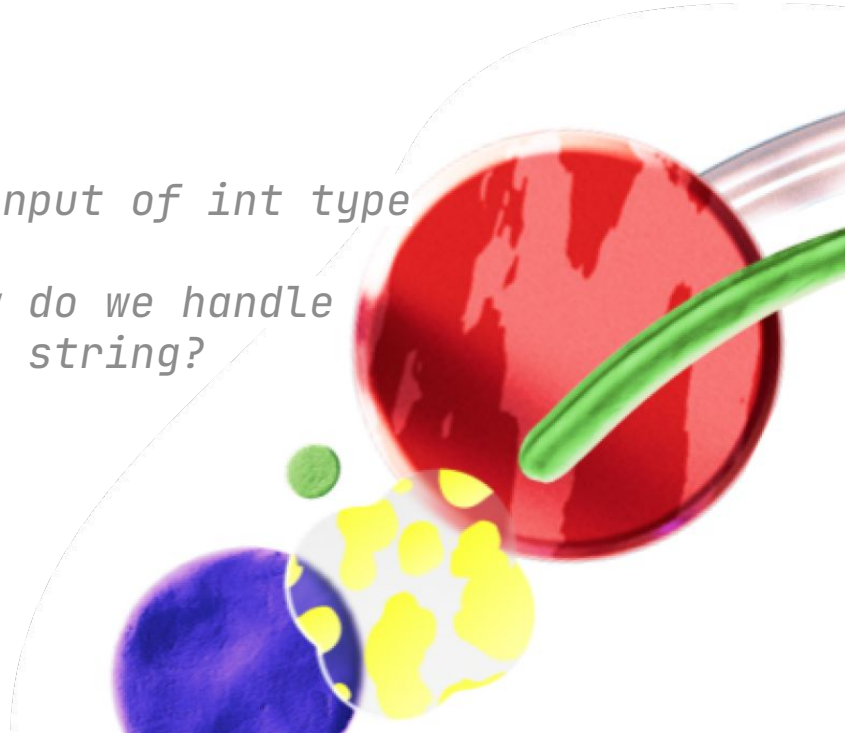
Variables

```
var p int = 1000
var a int
a = 5
var (
   k int
   f int
)
b := 50
c, _, e := 12, 123, 900
```

short declaration of several variables

Simple User Input

```go
func main() {
    fmt.Println("Input: ")
    var x int
    _, err := fmt.Scan(&x) // scan user input of int type
    if err ≠ nil {
        fmt.Println("Error: ", err) // how do we handle
    }                               // ... string?

    fmt.Println("You've typed", x)
}
```



Hands Up 5 min

Declare Variables of Different Types and Print Their Values

Solve in:

- **Sandbox — https://go.dev/play/**
- **Code Editor of your choice**

Possible Solution Leak

```go
func main() {
    var age int = 25
    fmt.Println("Age:", age)

    var isStudent bool = true
    fmt.Println("Is student:", isStudent)

    // Short variable declaration (alternative)
    country := "Russia"
    fmt.Println("Country:", country)
}
```

Write code

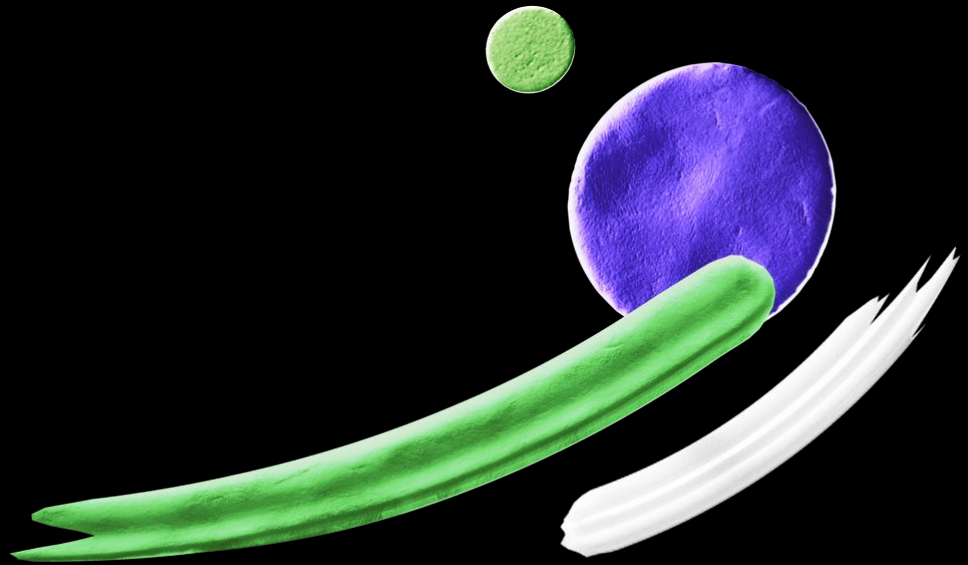
go build hello-world.go

./hello-world.go

IV. consts & iota

01 consts

02 iota’s

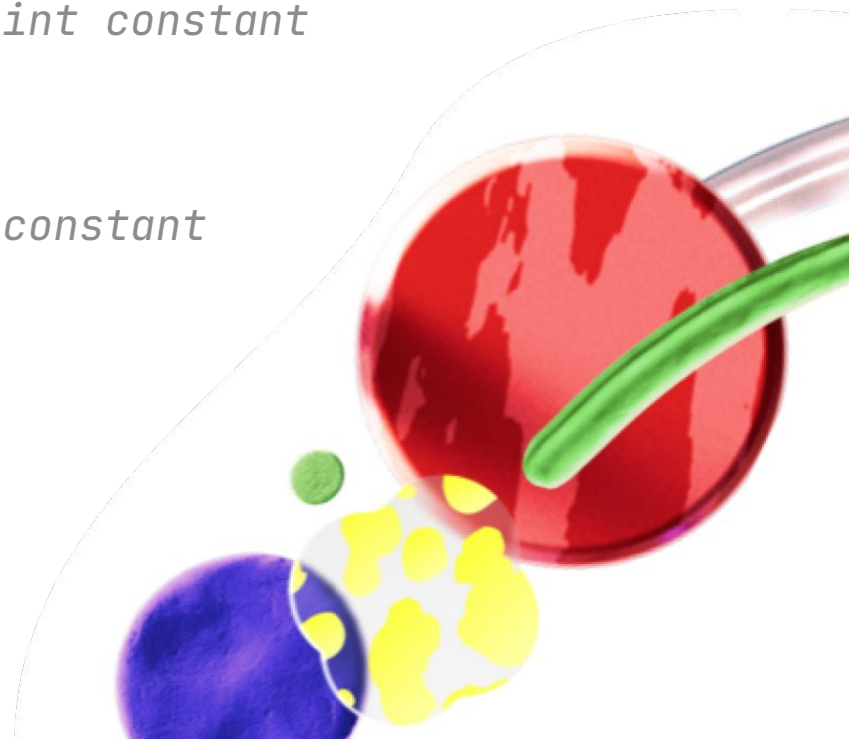


What is a constant and why is it used?

A constant is a named value in programming that does not change during the execution of a program. Unlike variables, which can be modified, constants retain their initial value once defined.

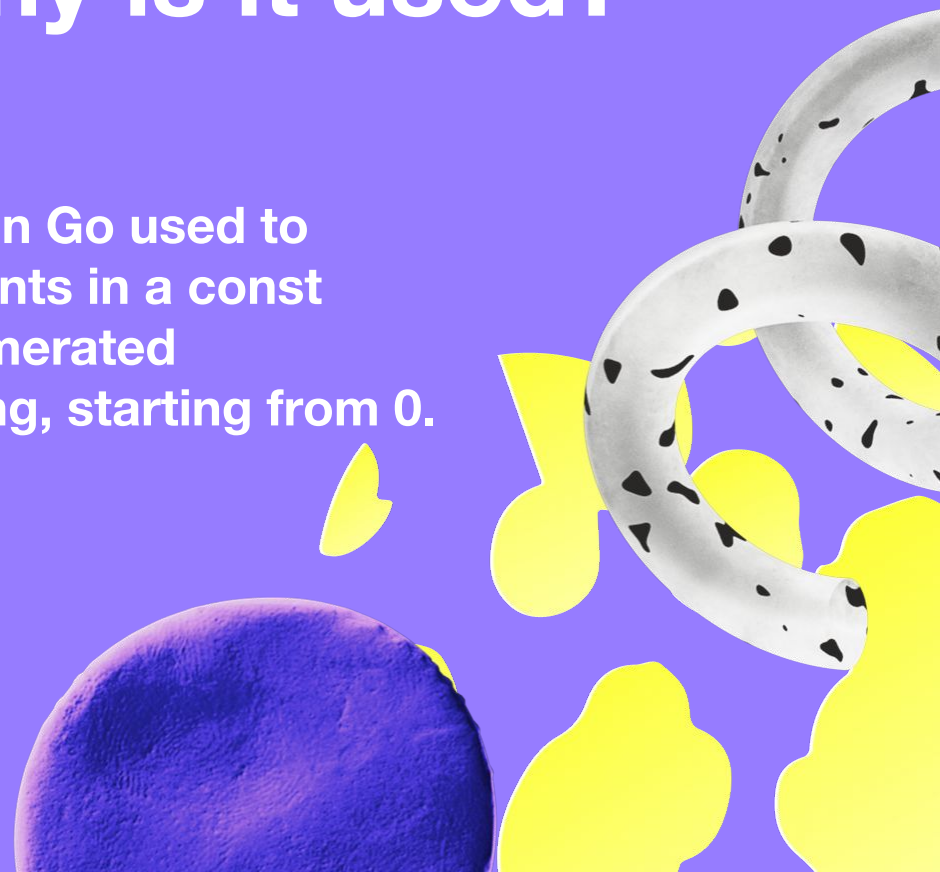
consts

```
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"
```



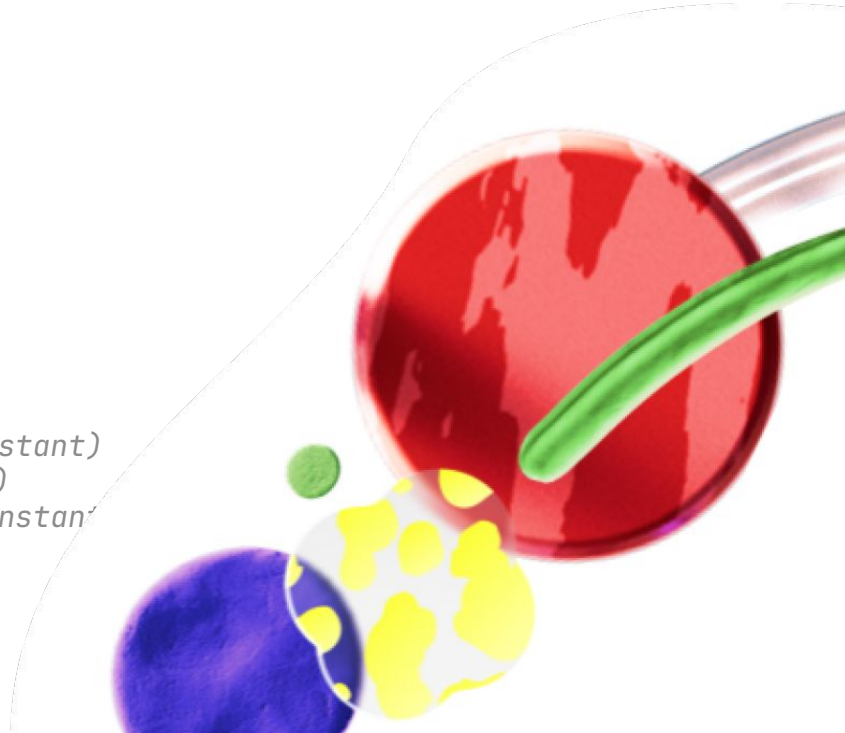
What is iota and why is it used?

iota is a special predeclared identifier in Go used to generate a sequence of related constants in a const block. It simplifies the creation of enumerated (enum-like) values by auto-incrementing, starting from 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 constan
)

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



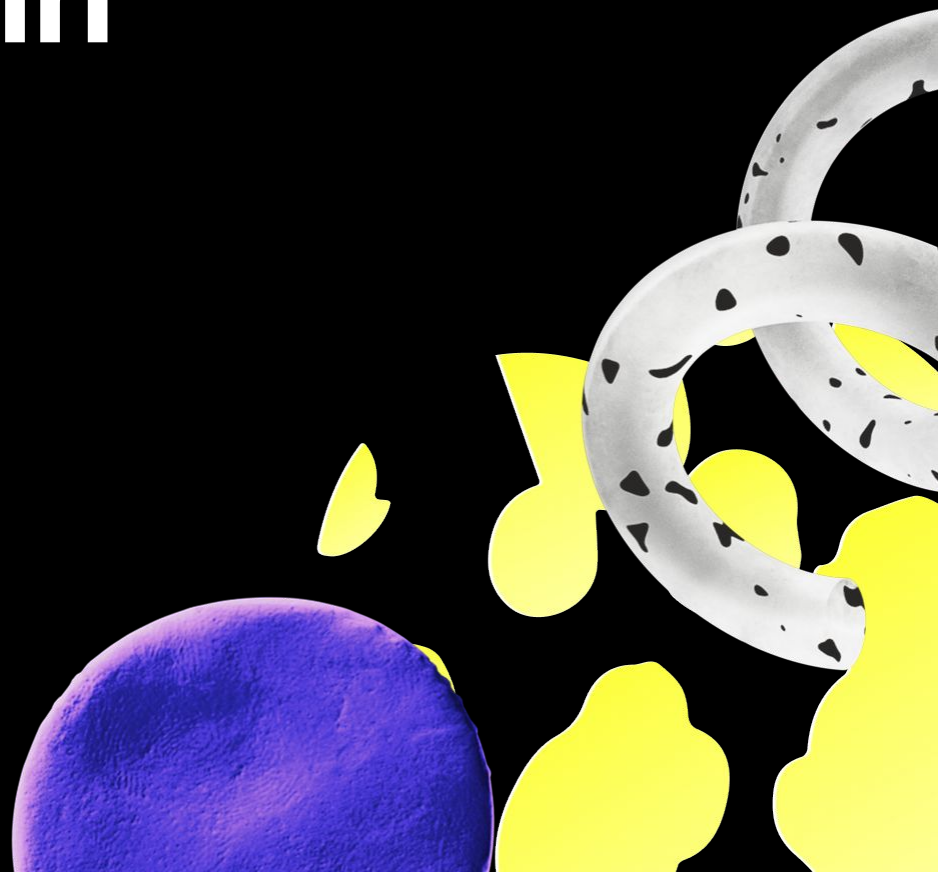
Hands Up 6 min

Create weekday list using const & iota

Solve in:

- Sandbox — https://go.dev/play/
- Code Editor of your choice

Save solution for later Hands Up



Solution Leak

```
const (
    Monday = iota + 1   // 1
    Tuesday             // 2
    Wednesday           // 3
    Thursday            // 4
    Friday              // 5
    Saturday            // 6
    Sunday              // 7
)
```

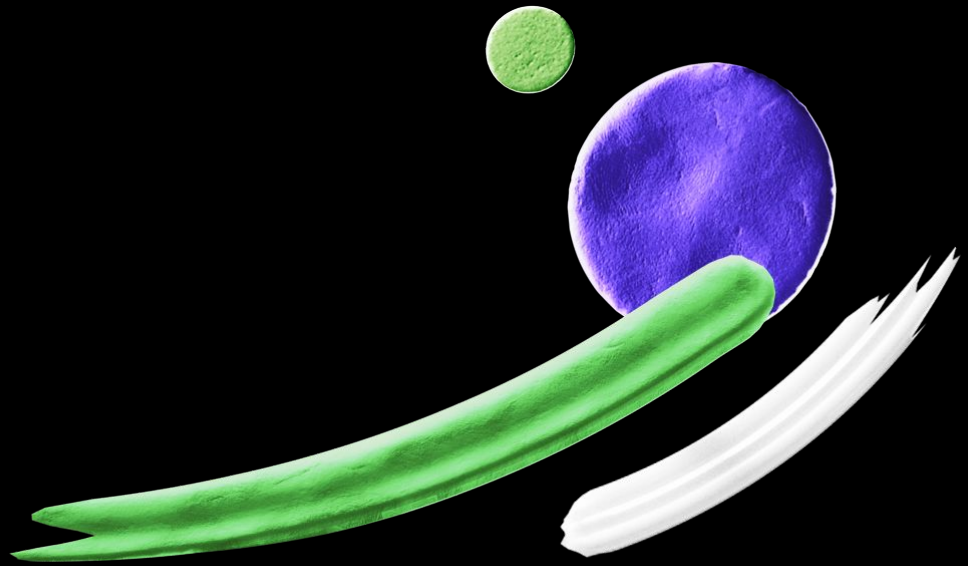
- We declare a struct
- Initialize it
- print all data

V. control flow

01 if - else

02 switch

03 for, break, continue

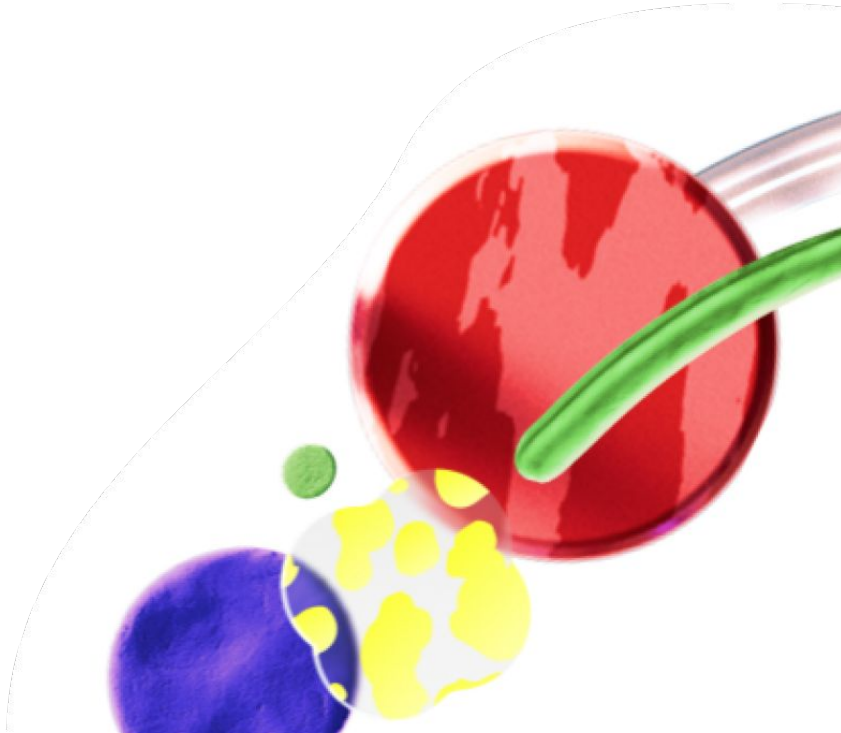


Control flow: if

```
flag := 1 == 23
counter := 0

if flag {
    counter += 1
} else {
    counter -= 1
}
fmt.Println(flag, counter)
```

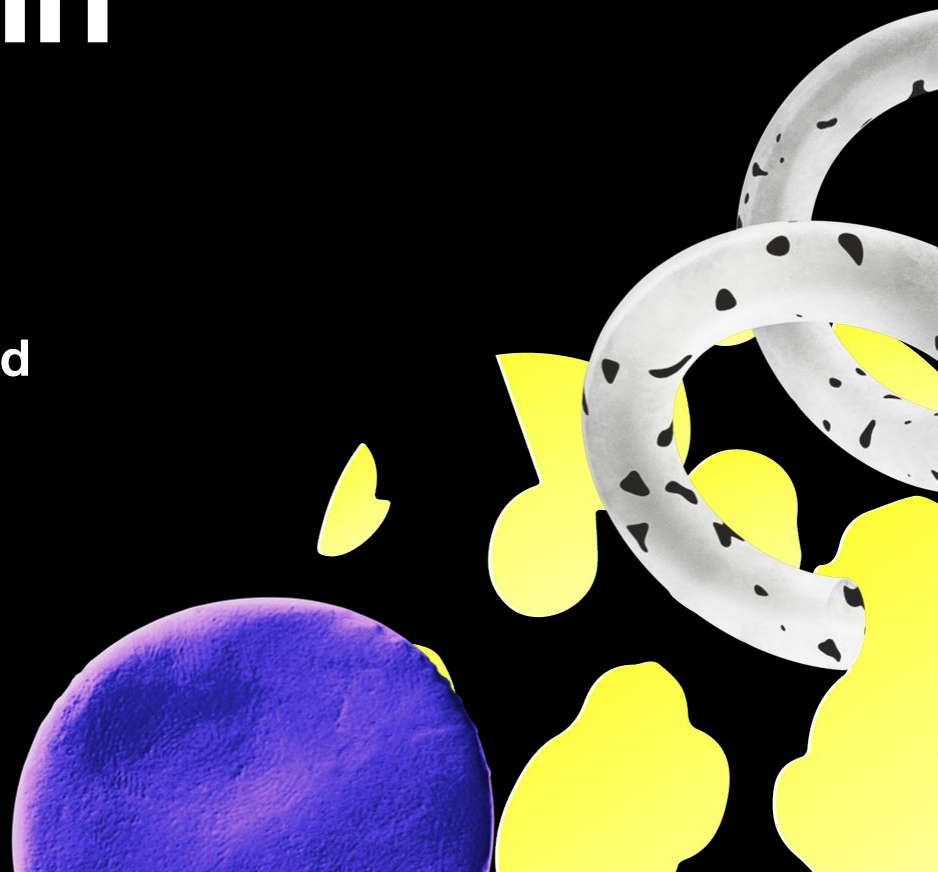
Control flow: switch

```
var value int
switch value {
case 1:
    fmt.Println("1")
    fallthrough
case 0:
    fmt.Println("0")
case 2:
    fmt.Println("2")
}
```



Hands Up 5 min

Open weekday list hands up:
Write a program that checks prompted weekday and
prints it number



Solution Leak

```
var day string
fmt.Print("Enter the day of the week (e.g., Monday): ")
fmt.Scan(&day)

switch day {
case "Monday":
   fmt.Println("Day number: ", Monday)
   …
case "Sunday":
   fmt.Println("Day number: ", Sunday)
default:
   fmt.Println("Error: Invalid day name!")
}
```

- We declare a struct
- Initialize it
- print all data

```
Enter the day of the week (e.g., Monday): Friday
Day number: 5
```

Control flow: for

```
for i := 0; i < 10; i++ {
    /*  */
}

/*  later in series */

sl := []int{1, 2, 3}
for i, v := range sl {
    fmt.Println(i, v)
}

for range 3 {
    fmt.Println("Wake Up")
}
```

Control flow: for

```
i := 0
for {
    i++
    if i%2 == 0 {
        continue
    }

    if i > 5 {
        break
    }
}
```

Solution Leak

```
for {
    // ...
    switch day {
    case "Monday":
        fmt.Println("Day number: %d", Monday)
    // ...
    case "exit":
        fmt.Println("Bye!")
        return
    default:
        fmt.Println("Error: Invalid day name!")
        return
    }
}
```

- We declare a struct
- Initialize it
- print all data

Enter the day of the week (e.g., Monday): Friday
Day number: 5

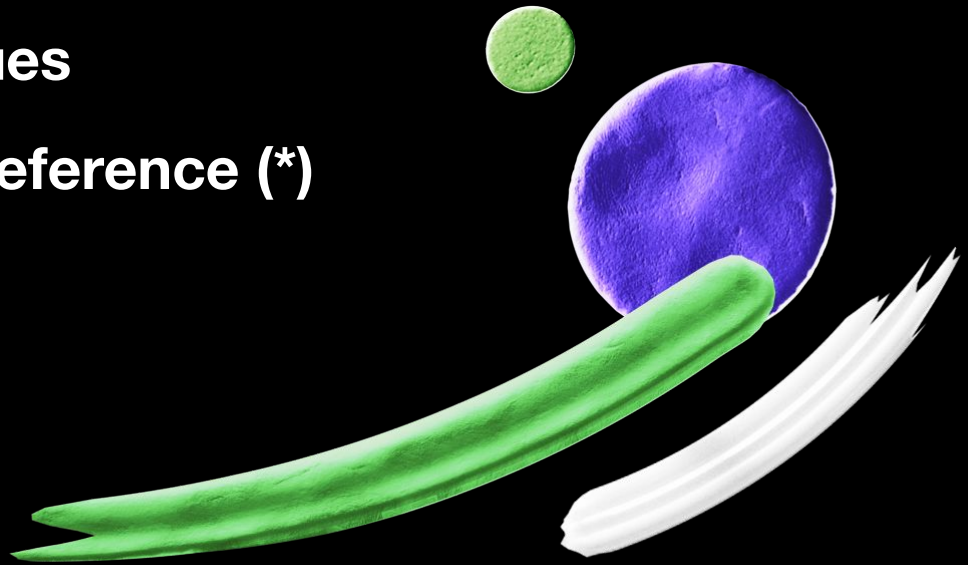
VI. funcs

01 Function Declaration

02 Arguments and Return Values

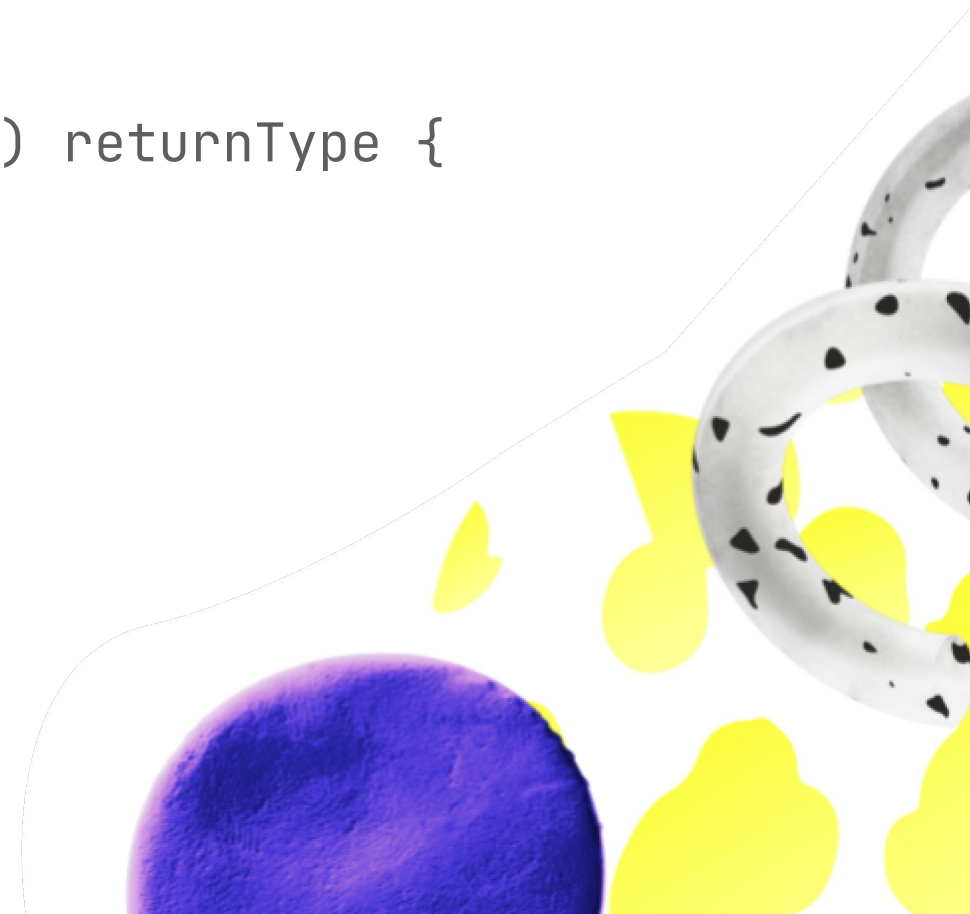
03 Pass by Value vs Pass by Reference (*)

04 defer



Functions

```
func functionName(parameters) returnType {
    // function body
}
```



Functions

```go
package main

import "fmt"

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

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

Functions

```go
package main

import "fmt"

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

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

Pass By Value vs Reference

```go
package main

import "fmt"

// Неправильно (по значению)
func swapBad(a, b int) {
    a, b = b, a // Изменятся только копии
}

// Правильно (по ссылке)
func swapGood(a, b *int) {
    *a, *b = *b, *a
}

func main() {
    x, y := 10, 20
    swapBad(x, y)
    fmt.Println(x, y) // 10 20 (не сработало)

    swapGood(&x, &y)
    fmt.Println(x, y) // 20 10 (теперь работает)
}
```

defer

```go
package main

import "fmt"

func main() {
    defer fmt.Println("Это выполнится третьим")
    defer fmt.Println("Это выполнится вторым")
    fmt.Println("Это выполнится первым")
}
```

avito.tech Saint Petersburg | 2025

Hands Up 7 min

Modify previous HandsUp by decomposing logic into functions

Solution Leak

```
func decider() bool {
   for {
      switch parseInput() {
      case "Monday":
         fmt.Println("Day number: ", Monday)
      // ...
      case "exit":
         fmt.Println("Bye!")
         return false
      default:
         fmt.Println("Error: Invalid day name!")
         return false
      }
   }
}
```

- We declare a struct
- Initialize it
- print all data

Solution Leak

```
func parseInput() (cmd string) {
    fmt.Print("Enter the day of the week (e.g., Monday): ")
    fmt.Scan(&cmd)
    return
}


func main() {
    for decider() { /* … */}
}
```

- We declare a struct
- Initialize it
- print all data

Class Summary

Today we've

- Learn how to use functions
- Understad iotas and consts
- Get familiar with variables control flow and basic data types
- And briefly checked history of language creation

What We'll Do Now – Installation Guide

- Install the language
- Set up the environment/editor (if needed)
- Go to github.com/avito-edu/goms-2025
- Complete the task
- Self-check the task
- Submit the task + get a grade

Get in touch

Заполните короткую форму обратной связи — нам очень важно быть с вами в диалоге