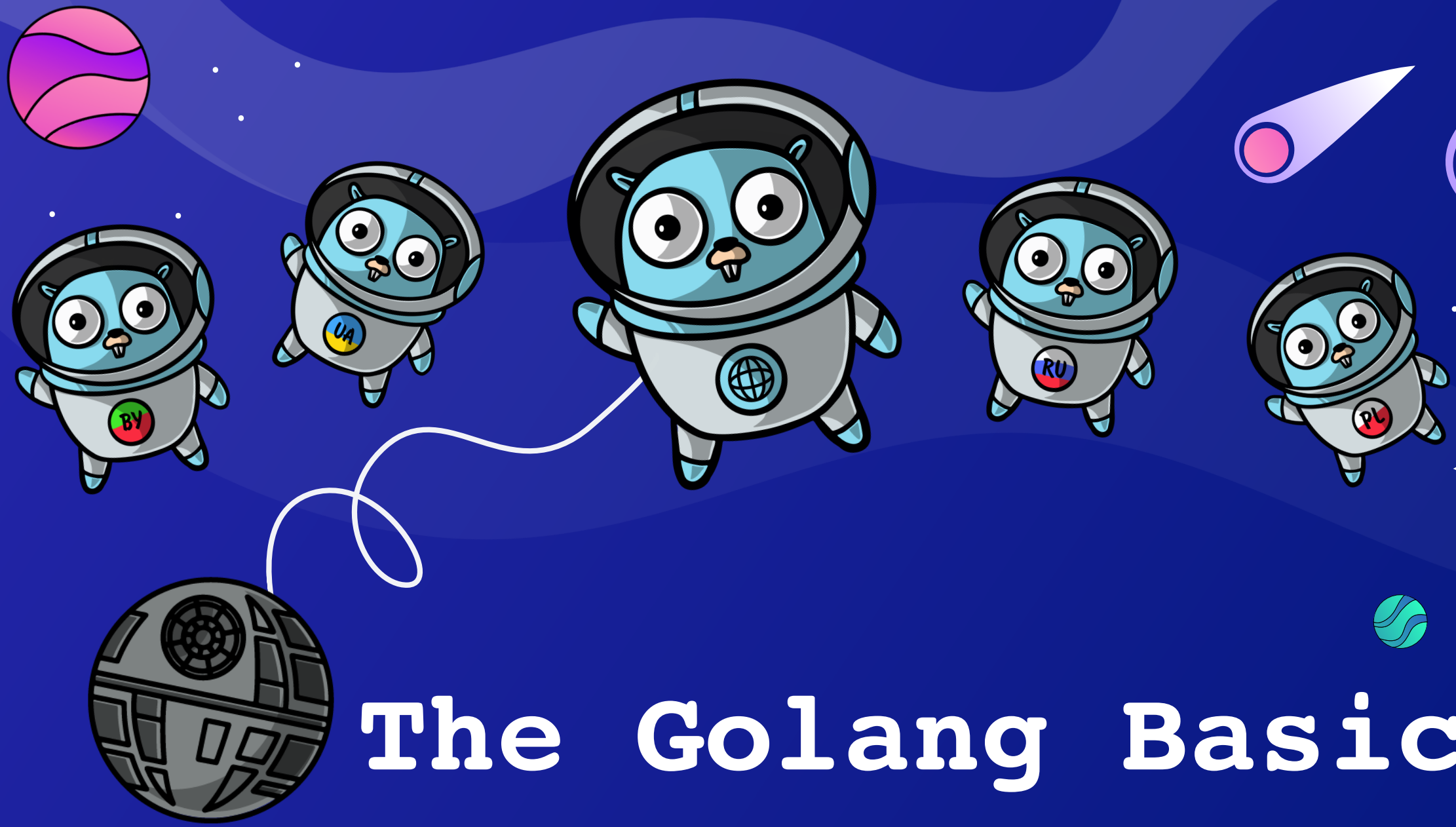



# The Golang Basics

# What we cover on this lecture?

- ***Variables***
- ***Basic types***
- ***Constants***
- ***functions***
- ***Flow-control***
- ***Loops***
- ***Packages***

# Golang keywords

| | | | | |
|---|---|---|---|---|
| break | default<br>else in switch/case | func | list of methods<br>interface<br>for type (class) | 4 goroutine like<br>select<br>await ? |
| case | finally evaluation<br>defer | create goroutine<br>go | dict<br>map | class w/o methods<br>struct |
| multi cpu ?<br>chan<br>parallel run | else | jump to label in<br>goto<br>statement | package | switch |
| const | множественный fit<br>fallthrough<br>в switch/case | if | iterate over loop<br>range | new type<br>type |
| continue | for | import | return | var |

*You cannot use key words as variables, types, function names in your program*

# Predeclared names

| | |
|---|---|
| Constants | true, false, iota, nil<br>1 11<br>None<br>enumerate |
| Types | int, int8, int16, int32, int64,<br>uint, uint8, uint16, uint32, uint64,<br>uintptr,<br>float32, float64, complex128, complex64,<br>bool, byte, rune, string, error |
| Functions | make, len, cap, new, append, copy, close,<br>delete, — slices and etc.<br>complex, real, imag,<br>panic, recover — try / except |

*Because in general standart function, types and constants is not a language keywords, you can redeclare them, BUT NEVER DO THIS! :)*

# Variables

- Переменные инициализируются со значением, иначе с дефолтным значением
- У каждого типа свое
- Мельзя переопределять или сравнивать переменные разных типов

```
var x int
var x int = 5
var x = 5
```

— x = 0

— type will be computed

x := 5

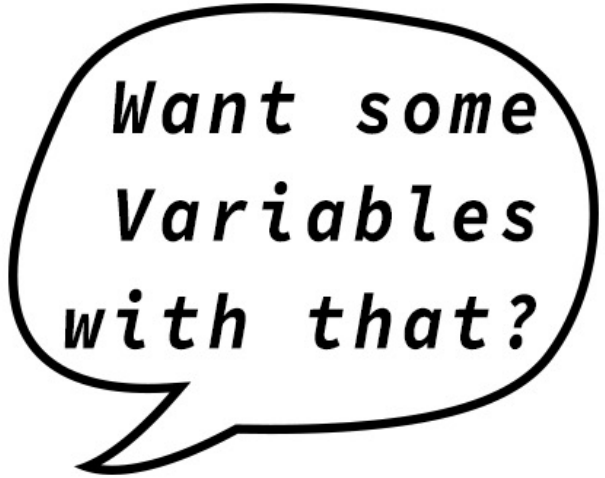
CamelCase

const ModuleScope

const packageScope

— У констант более слабая типизация, если тип computed

Short declaration:
Если переменная используется в маленьком скоупе — она должна быть короткой

# Variables initialization

```go
func main() {
    var (
        b1  bool
        s1  string
        i1  int
        ui1 uint
        by1 byte
        r1  rune
        f1  float32
        c1  complex64
    )

    fmt.Println("Boolean - ", b1)
    fmt.Printf("String - %q\n", s1)
    fmt.Println("Integer - ", i1)
    fmt.Println("Unsigned Integer - ", ui1)
    fmt.Println("Byte - ", by1)
    fmt.Println("Rune - ", r1)
    fmt.Println("Float number - ", f1)
    fmt.Println("Complex number - ", c1)
}
```

```
Boolean -  false
String - ""
Integer -  0
Unsigned Integer -  0
Byte -  0
Rune -  0
Float number -  0
Complex number -  (0+0i)
```

***https://goplay.tools/snippet/LUmySkremfm***

# Every literal has it's own default type

```go
package main

import (
    "fmt"
)

func main() {
    s := ""
    c := 'b'
    i := 0
    f := 0.0

    fmt.Printf("String literal - %T\n", s)
    fmt.Printf("Integer literal - %T\n", i)
    fmt.Printf("Character literal - %T\n", c)
    fmt.Printf("Floating number literal - %T\n", f)
}
```

**String literal — string**

**Integer literal — int**

**Character literal - int32**

**Floating number literal - float64**

https://goplay.tools/snippet/xnqb_jHuyNr

byte - uint8 alias, can hold ASCII

rune - int32 alias, can hold UTF-8

Инкремент и декремент:

i++ 1. считывает переменную из памяти
i-- 2. добавляет / вычитает 1
3. мутирует переменную

# Basic types

# Logical operators

| A | B | !A | A\|\|B | A&&B |
|---|---|---|---|---|
| false | false | true | false | false |
| false | true | true | true | false |
| true | false | false | true | false |
| true | true | false | true | true |

# Integer binary operators

побитовые операции

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | * | / | % | << | >> | & | &^ |
| 2 | + | - | \| | ^ | | | |
| 3 | == | != | < | <= | > | >= | |
| 4 | && | | | | | | |
| 5 | \|\| | | | | | | |

# String type

| | | |
|---|---|---|
| *byte* | *'A'* | *ASCI character, 1 byte size* |
| *rune* | *'ó'* | *UTF-8 character, up to 4 byte size* |
| *string* | *"Kraków"* | *String, a bunch of UTF-8 encoding characes.* |

# *Converting the string*

```
package main

import (
    "fmt"
)

func main() {
    msg := "Dzień dobry"
    //msg is a string with 11 characters

    fmt.Println(len(msg)) //12
    // Why we have 12 here ?
}
```

*Dzień dobry*

*https://goplay.tools/snippet/7yGc4jBf7uQ*

*Why len(msg) returns 12 when we have 11 characters? There answer is quite simple, len returns value in bytes, and character "ń" cost us 2 bytes in UTF-8 encoding (non-ASCI character).*

# Constants

```go
const name string = "Go"
const (
    e  = 2.7182
    pi = 3.1415
)
const (
    a = 1.1
    b
    c = 2
    d
)

func main() {
    fmt.Println(name)
    name := "Java"
    fmt.Println(name)

    fmt.Println(a, b, c, d)
    fmt.Printf("%T %T\n", b, d
)
}
```

*Go*
*Java*
*1.1 1.1 2 2*
*float64 int*

*https://goplay.tools/snippet/O-po3ANlo6u*

*Constants in Golang are variables that has in place initialization and that value cannot be changed after first initialization. (One exception, you can in smaller scope define variable with the same name)*

*Constants have a little bit different type system, strict type of the constant computed when it used, it means that you can compare constant e with float32 and float64 type, as well as constant b can be used without type casting with any numbered type. The same applicable with type aliases(we will talk about it later.)*

Enumerate

# Iota and constants

```
type Weekday int

const (
    Monday    Weekday = iota
    Tuesday           = iota
    Wednesday         = iota
)

const (
    a = iota * 2
    _
    b
    c
)

func main() {
    fmt.Println(Monday, Tuesday, Wednesday)
    fmt.Println(a, b, c)
}
```

*0 1 2*

*0 4 6*

https://goplay.tools/snippet/6DqDZZH1SCi

*Iota is a constant with special behavior, it is kind of generator that return unique sequential number for constant block.*

- *Value of iota not shared across different constants blocks, but*
- *You can change value of iota using standard operations as +, *, -.*
- *In case you want to skip value you can use blank identifier _*

# Type system variables vs constants

## VARIABLES

- *Variables has strict type system, int8 and int32 for example is different type and can't be used interchangeably.*

## CONSTANTS

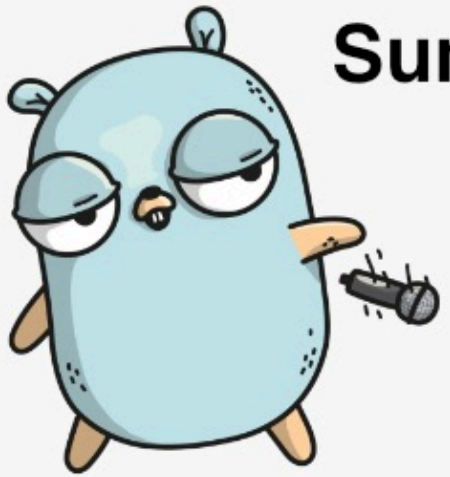
- *Constant has weak type system, for example string constant has type untyped string constant and numbers untyped integer, final type computed when constant is used, this allows to use constants with any castable type.*

```
func name (params) (results) {
    // body
}
```

```
func sum(a, b int) (res int){
    res = a+b
    return
}
```

← так же можно возвращать nil, если в теле ф-ции не опр-но возвращаемая переменная

# Functions



Sum

12

6

map[int64]int64

[]int32{}

[]interface{1,2,[]int64{}}

# Function types

```
package main

import "fmt"

func add(x int, y int) int
{ return x + y }
func sub(x, y int) (z int)
{ z = x - y; return }
func first(x int, _ int) int
{ return x }
func zero(int, int) int
{ return 0 }

func main() {
    fmt.Printf("%T\n", add)
    fmt.Printf("%T\n", sub)
    fmt.Printf("%T\n", first)
    fmt.Printf("%T\n", zero)
}
```

```
func(int, int) int
func(int, int) int
func(int, int) int
func(int, int) int
```

https://goplay.tools/snippet/cpbj8m-40rw

*Function in Golang has their own type.*

*Function name must be unique in the scope, you cannot override function.*

# Recursion

```go
package main

import "fmt"

func factorial(i uint) uint {
    if i == 0 {
        return 1
    }
    return i * factorial(i-1)
}

func main() {
    fmt.Println(factorial(5))
}
```

```
120
```

https://goplay.tools/snippet/Re_kjE0PHAl

# Multiple return values

```
package main

import (
    "fmt"
)

func main() {
    fmt.Println(devide(10, 3))
}

func devide(a, b int) (int, int) {
    return a / b, a % b
}
```

tuple

```
a, _ = devide(100, 15)
```

```
3 1
```

https://goplay.tools/snippet/fTlb7HpuDyi

*Functions in Golang can have multiple return parameters, it gives an ability to have pretty elegant solutions, but be conservative with this feature, large number or return statement usually signal that you break single responsibility principle.*

# Variadic parameter

```go
func main() {
    digits := []int{1, 2, 3, 4}

    //You cannot just pass an array, you have to expand it
    //fmt.Println(sum(digits)) // Will not work
    fmt.Println(sum(digits...))
}

//variadic parameter must be last parameter of the function
func sum(digits ...int) int {
    var sum int
    for _, d := range digits {
        sum += d
    }

    return sum
}
```

... eval of * in py

```
10
```

https://goplay.tools/snippet/Wobv8_U_y9d

You need to keep in mind a few things:

- Variadic parameter is a slice under the hood (slice is a dynamic array in Golan, about this type later)
- Variadic parameter must be last in a function
- You need to expand the slice to pass it as a variadic parameter with "..."

# Change underlined slice in variadic parameter

```
func main() {
    strs := []string{
        "Hello,", "My", "name",
        "is", "Alex",
    }
    mutate(strs...)

    fmt.Println(strings.Join(strs, " "))
}

func mutate(x ...string) {
    x[len(x)-1] = "Michał"
}
```

Hello, My name is Michal

https://goplay.tools/snippet/6imaJjAAB0A

You need to keep in mind a few things:

- ”...” doesn’t copy slice due to this you can change slice inside a variadic function

# High-order functions

```
package main

import "fmt"

func newWriter() func(string) {
    return func(s string) {
        fmt.Println(s)
    }
}
func main() {
    var writer func(string) = newWriter()
    writer("Hello Poland!")
    writer("Cześć!")
}
```

we can create decorators in Go? :)

```
Hello Poland!
Cześć!
```

https://goplay.tools/snippet/4y54T8g5qst

*What we just did?*

- *Assign a function to a variable*
- *Return a function from the function*
- *Define anonymous function (function without name)*

# Deferred functions = finally

```go
package main

import (
    "fmt"
)

func main() {
    var message string = "I will be executed after exit from the function"

    defer func() {
        fmt.Println(message)
    }()

    fmt.Println("Exit from function")
}
```

```
Exit from function
I will be executed after exit from the function
```

https://goplay.tools/snippet/AM29trKSYZK

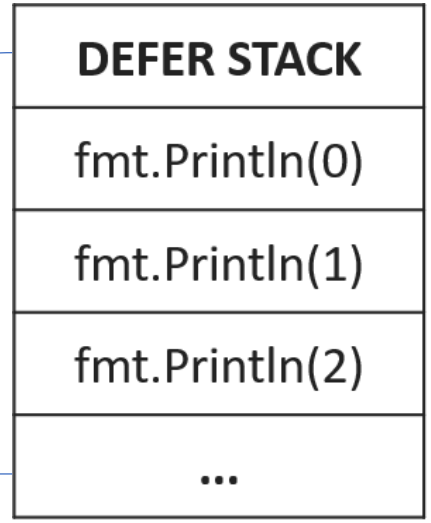
# Deferred functions execution order

```go
package main

import "fmt"

func main() {
    for i := 0; i < 5; i++ {
        defer fmt.Println(i)
    }

    // deferred funcs run here
}
```




4
3
2
1
0

https://goplay.tools/snippet/ePn-bHg2S-X

# Deferred functions

```
package main

import (
    "fmt"
)

func main() {
    for i := 0; i < 10; i++ {
        defer func() {
            fmt.Print(i, " ")
        }()
    }
}
```

10 10 10 10 10 10 10 10 10 10

https://goplay.tools/snippet/URnuA-CzSE_V

# Deferred functions use cases

```
row, err := db.Query(`SELECT ...`)
if err != nil {
        // handle error or path it to the
caller
}
defer row.Close()
```

```
defer func() {
    if err := recover(); err != nil {
        ...
    }
}()

panic("oops!")
```

***https://goplay.tools/snippet/dEShiFiGuCO***

# Function in Golang key points

- *Function is just a bunch of operationgs grouped in a logical way.*
- *You can assign function to variable, pass it as a parameter to the function or return from the function.*
- *Functions in Golang has type as well, type of the function depending on its parameters and return statements.*

# Flow control

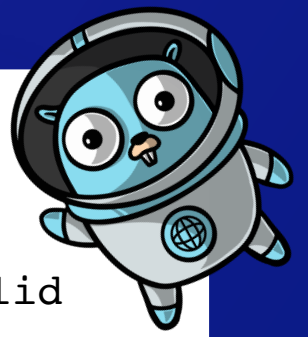
# If - else

```
func main() {
    for i := 0; i <= 7; i++ {
        if weekday, err := isWeekDay(i); err != nil {
            fmt.Println("unexpected error - ", err)
        } else {
            fmt.Println(i, "-", weekday)
        }
    }
}

func isWeekDay(d int) (bool, error) {
    if d <= 0 || d > 7 {
        return false, fmt.Errorf("invalid value, valid range is [1-7]")
    } else if d > 5 {
        return false, nil
    } else {
        return true, nil
    }
}
```

```
unexpected error - invalid value, valid range is [1-7]
1 - true
2 - true
3 - true
4 - true
5 - true
6 - false
7 - false
```

https://goplay.tools/snippet/V27mHXgTT42



# Switch statement

```go
func main() {
    fmt.Println(numberToWeekDay(1))
    fmt.Println(numberToWeekDay(3))
}
func numberToWeekDay(i int) string {
    switch i {
    case 1:
        return "Monday"
    case 2:
        return "Tuesday"
    case 3:
        return "Wednesday"
    case 4:
        return "Thursday"
    case 5:
        return "Friday"
    case 6:
        return "Saturday"
    case 7:
        return "Sunday"
    default:
        return "unknown"
    }
}
```

*Monday*

*Wednesday*

*https://goplay.tools/snippet/YMHG82I9nFN*

*Switch statement in Golang has pretty common syntax, one important difference is that instead of falltrough behavior, Golang breaks after each case.*

# Switch statement

```go
func main() {
    fmt.Println(isWeekDay(1))
    fmt.Println(isWeekDay(6))
    fmt.Println(isWeekDay(10))
}

func isWeekDay(i int) (bool, error) {
    switch i {
    case 1, 2, 3, 4, 5:
        return true, nil
    case 6, 7:
        return false, nil
    default:
        return false, fmt.Errorf("invalid value, valid range [1-7]")
    }
}
```

```
true <nil>
false <nil>
false invalid value, valid range [1-7]
```

https://goplay.tools/snippet/SMVOi9WLJag

In case you have the same behaviour for multiple values, you can specify multiple values in one case.

# Switch statement

```
func main() {
    fmt.Println(isWeekend(6))
    fmt.Println(isWeekend(5))
    fmt.Println(isWeekend(10))
}
func isWeekend(i int) (bool, error) {
    switch {
    case i >= 6 && i <= 7:
        return true, nil
    case i >= 1 && i <= 5:
        return false, nil
    default:
        return false, fmt.Errorf("invalid value, valid range [1-7]")
    }
}
```

```
true <nil>
false <nil>
false invalid value, valid range [1-7]
```

https://goplay.tools/snippet/xENCYSAAXeu

You can implement all use-cases of if-else statement using switch statement, in case of absence of variable in switch you can just specify expression with boolean result.

# Loops

# Loops (Plain old for loop)

```go
package main

import (
    "fmt"
)

func main() {
    for i := 0; i < 10; i++ {
        fmt.Print(i, " ")
    }
}
```

```
0 1 2 3 4 5 6 7 8 9
```

https://goplay.tools/snippet/KEBAFKZvwvx

*Plain for loop syntax is common for most languages, the only difference is that you don't need parentheses around conditions, but the braces are required.*

# Loops (While loop)

```go
package main

import (
    "fmt"
)

func main() {
    var i int
    for i < 10 {
        fmt.Print(i, " ")
        i++
    }
}
```

```
0 1 2 3 4 5 6 7 8 9
```

https://goplay.tools/snippet/Fbp11kKVQw4

*In Golang while loop is special type of for loop.*

# Loops (Infinite loop)

```go
package main

import (
    "fmt"
)

func main() {

    for {
        fmt.Println("Hello from infinite loop")
    }
}
```

```
Hello from infinite loop
Hello from infinite loop
Hello from infinite loop
Hello from infinite loop
Hello from infinite loop
...
```

https://goplay.tools/snippet/AhPisV44upd

*Empty values works as infinite loop*

# Loops (For each)

```go
func main() {
    strs := []string{
        "Hello",
        "from",
        "for-each",
        "loop",
        "!",
     }

    for _, s := range strs {
        fmt.Print(s, " ")
    }
}
```

```
Hello from for-each loop !
```

https://goplay.tools/snippet/JvDY6ZHkrUY

*For range loop is helpful do go over collections, range key word used, first parameter returned by range is index of the element, second parameter is actual value.*

# Loops (continue, break)

```go
func main() {
    strs := []string{
        "Hello",
        "from",
        "for-each",
        "loop",
        "!",
    }

    for _, s := range strs {
        if s == "for-each" {
            continue
        } else if s == "!" {
            break
        }
        fmt.Print(s, " ")
    }
}
```

```
Hello from loop
```

https://goplay.tools/snippet/u6f1sHcxdbM

*You can use break and continue operators, to control loop behavior.*

*Continue stop current iteration and start next one.*

*Break completely stop loop.*

# What should you remember?

- In Golang there is only **for** loop, no **while** or **do-while.**
- Loop variable initialized only once; you need to copy it before using in closures or path to a function in case it reference type.
- For loop contains three parts, variable initialization, condition, post statement any of this part can be skipped or extended with different conditions.

- в директории должен быть (1) .go файл и называться также как директория, и kw package <name> тоже должно совпадать.
- переменные/функции и др. объекты:
  myConst – package scope - приватные
  MyConst – module scope - можно исп-ть в др. пакетах

# Packages

# Exported/unexported objects

```
~/go/src/github.com/burov/
greeting/
├── go.mod
├── main.go
└── language
    ├── polish
    │   └── polish.go
    ├── english
    │   └── english.go
    └── language.go
```

```go
package main

import (
    "fmt"

    "github.com/burov/greeting/language/polish"
)


func main() {
    fmt.Println(polish.Name)
    fmt.Println(polish.internalName)
}
```

```go
const Name = ”Polish"
const internalName = ”pl"
```


```
./main.go:10:14: cannot refer to unexported name polish.internalName
./main.go:10:14: undefined: polish.internalName
```

# Packages example

```go
package main

import (
    "fmt"
    "github.com/burov/greeting/language/english"
    "github.com/burov/greeting/language/polish"
)

func main() {
    fmt.Println(english.Name)
    fmt.Println(polish.Name)
    //fmt.Println(english.internalName)
    //cannot refer to unexported name english.internalName
    //fmt.Println(polish.internalName)
    //cannot refer to unexported name polish.internalName
}
```

English

Polish

https://goplay.tools/snippet/GSHnaLmADtH

*To import package, you need to use module name + package or if you use $GOPATH just a relative path from $GOPATH/src + package*

*Every variable/function/constant/structure named from upper-case letter is exported (you can use them from other packages), from lower-case letter is package private only.*

# Summary about packages

- *Package is just a folder with files, in one folder you can have only one package.*
- *Every file start from package definition, as a good practice name of the package is equals to folder name.*
- *Main package is a special package that is entry point of the code*
- *Every variable/function/constant/structure named from upper-case letter is exported and can be accessed outside of the project.*
- *Every variable/function/constant/structure named from lower-case letter is package private and cannot be accessed outside of the project.*