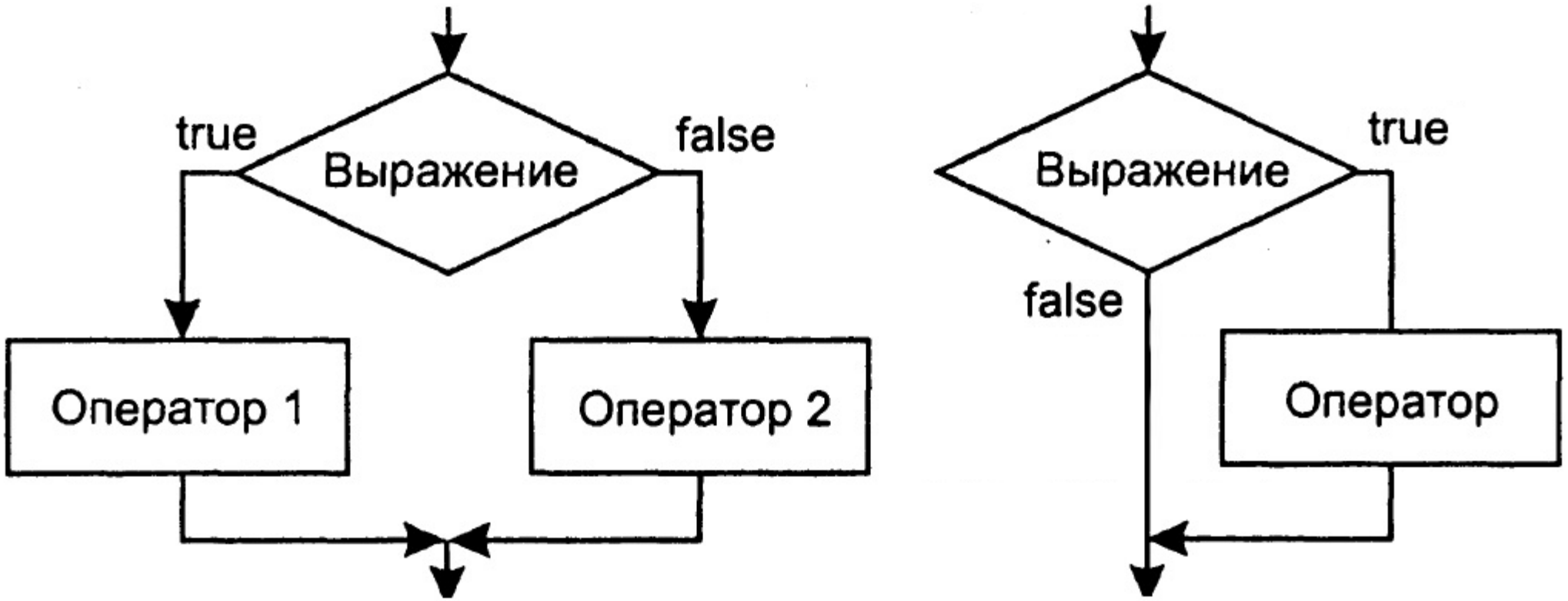
Алгоритмизация и программирование

Лекция 2 (Go)

Условный оператор



Структурная схема условного оператора

Тип bool

```
var x bool = true   // Истина
var y bool = false  // Ложь
```

* Маленькими буквами

Преобразование типа bool

В Go нет возможности преобразовать bool в другой тип или другой тип в bool стандартным способом. Автоматического преобразования тоже нет.

Преобразование типа bool руками

```
package main

import "fmt"

func main() {
    var a int = 10
    var b bool = a != 0
    fmt.Println(b) // true

    var c int
    if b {
        c = 1
    }else{
        c = 0
    }
    fmt.Println(c) // 1
}
```

Откуда берётся true и false

Операторы сравнения

Оператор	Проверяет на	Пример использования	Результат
==	Равенство	3 == 7	false
!=	Неравенство	3 != 7	true
<	Меньше	3 < 7	true
>	Больше	3 > 7	false
<=	Меньше или равно	7 <= 7	true
>=	Больше или равно	7 >= 7	true

Логические операторы

Название	Как выглядит	Как использовать	
И	&&	a && b;	a and b;
ИЛИ	||	a || b;	a or b;
НЕ	!	!a;	not a;

a	b	a and b	a or b	not a
false	false	false	false	true
true	false	false	true	false
false	true	false	true	true
true	true	true	true	false

Логические операторы применяются только к операндам типа `bool`.

Операторы И и ИЛИ вычисляются по сокращённым правилам, т.к. если результат можно получить вычислив первый аргумент, второй не вычисляется:

```
func plus() bool{
    fmt.Println("+")
    return true
}
```

```
fmt.Println(true && plus()) // +true
fmt.Println(false && plus()) // false
```

Логические операторы. Возможные ошибки

```
int a = 10;
-1 < a < 2   // ошибка
```

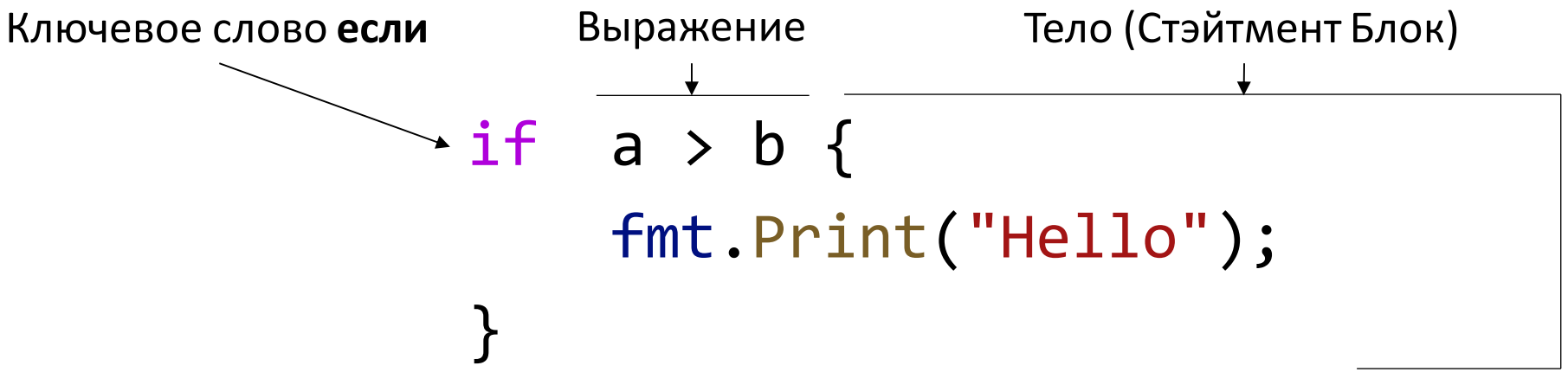
Последнее выражение вычисляется последовательно:

```
  -1 < a < 2
(-1 < a) < 2
    true < 2  // ошибка. Число не типа bool
```

Если хотите получить результат по математическим правилам пишите:

```
(-1 < a) && (a < 2)   // false
```

if


Ключевое слово **если**
Выражение
Тело (Стэйтмент Блок)
if a > b {
fmt.Print("Hello");
}

if выражение

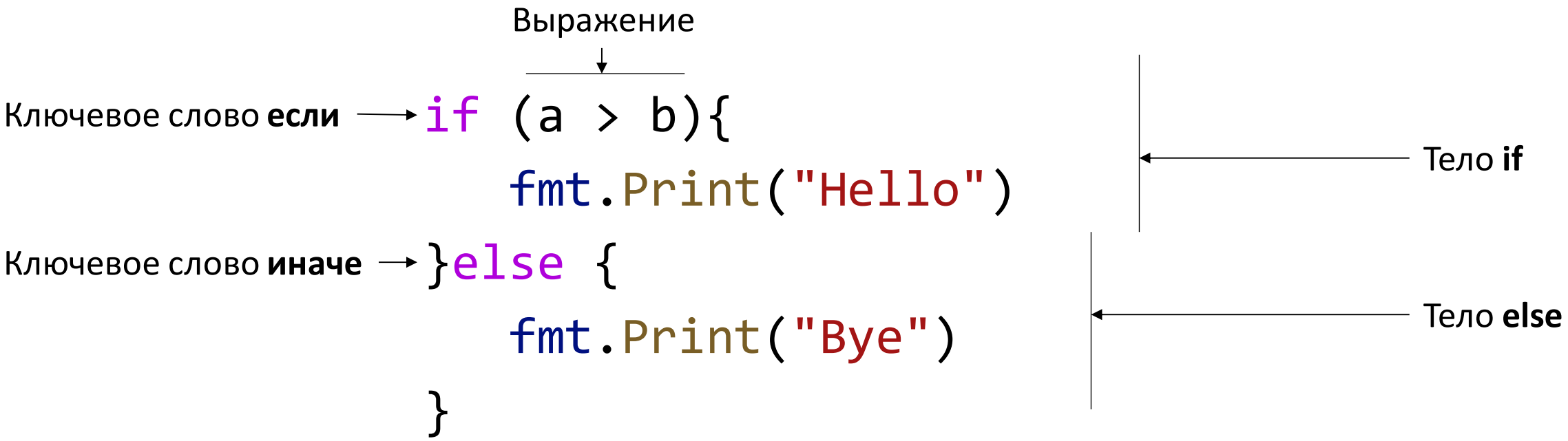
Ожидается, что выражение в скобочках типа `bool`.

if(инициализация; проверка)

SimpleStmt

```
if x := f(); x < y {
    return x
} else if x > z {
    return z
} else {
    return y
}
```

if-else


Выражение
Ключевое слово **если**
if (a > b){
fmt.Print("Hello")
Ключевое слово **иначе**
}else {
fmt.Print("Bye")
}
Тело **if**
Тело **else**

if

```
if /* условие */ {
    /* true */
}
```

If-else

```
if /* условие */ {
    /* true */
}else{
    /* false */
}
```

If-else if

```
if /* условие 1 */ {
    /* true */
} else if /* условие 2 */ {
    /* true */
} else {
    /* false */
}
```

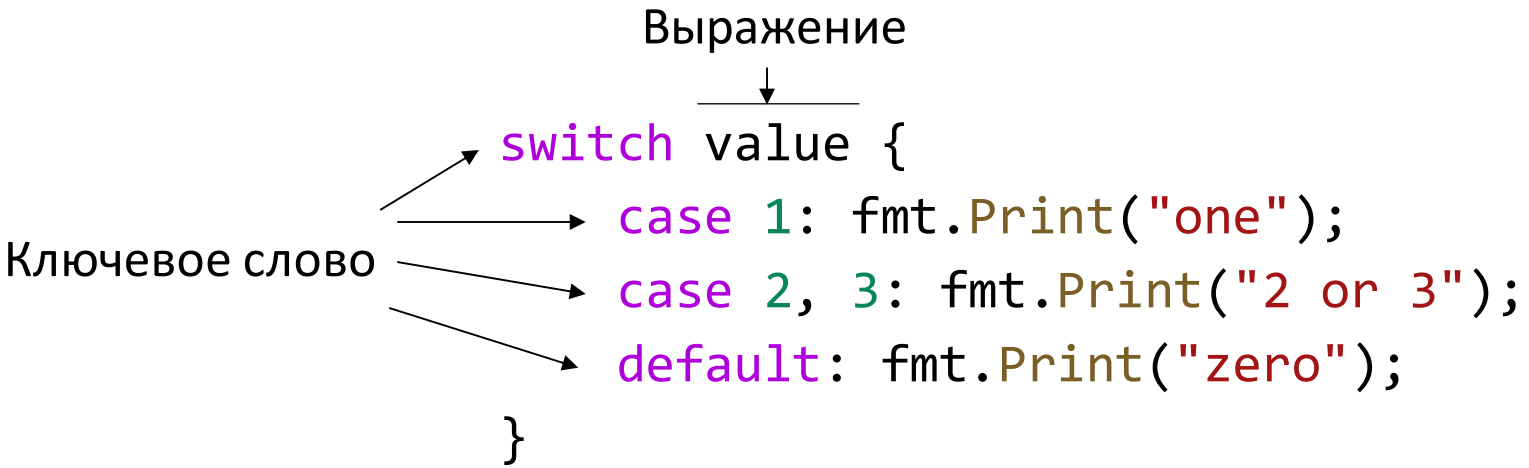
Вложенный if

```
if /* условие */ {
    if /* условие */ {
        /* true */
    }else{
        /* false */
    }
}else{
    if /* условие */ {
        /* true */
    }else{
        /* false */
    }
}
```

Тернарный оператор

Тернарного оператора или аналогов в Go нет.

switch [ExprSwitchStmt]



switch выражение

Выражение может быть любого типа, но обязательно такого же как и в case - ах.

Если выражение отсутствует, то это тоже самое, что и **true.**

switch - fallthrough

```
nextstop := "B"
fmt.Println("Stops ahead of us:")

switch nextstop {

case "A":
    fmt.Println("A")
    fallthrough

case "B":
    fmt.Println("B")
    fallthrough

case "C":
    fmt.Println("C")
    fallthrough

case "D":
    fmt.Println("D")
    fallthrough

case "E":
    fmt.Println("E")
}
```

switch инициализация; выражение

SimpleStmt

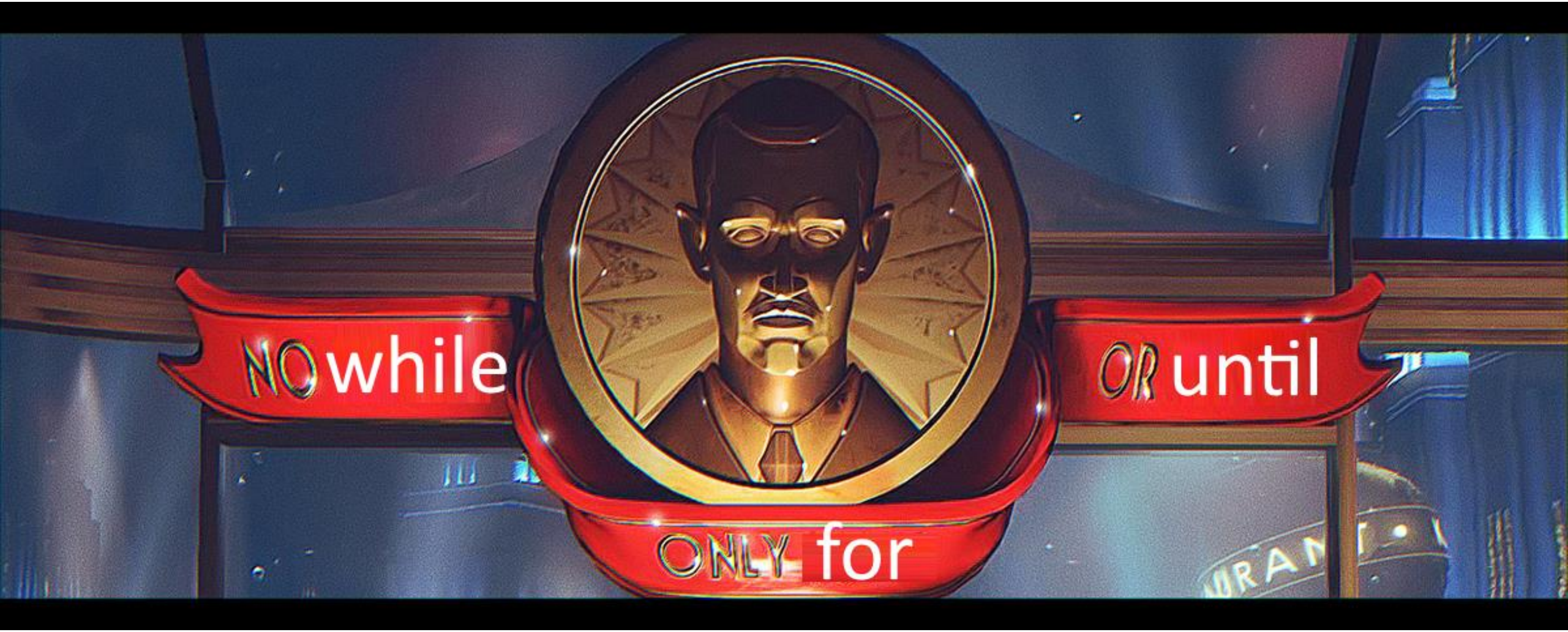
```
switch res := a > b; res {
  case true: fmt.Print(":)")
  default: fmt.Print(":(")
}
```

goto

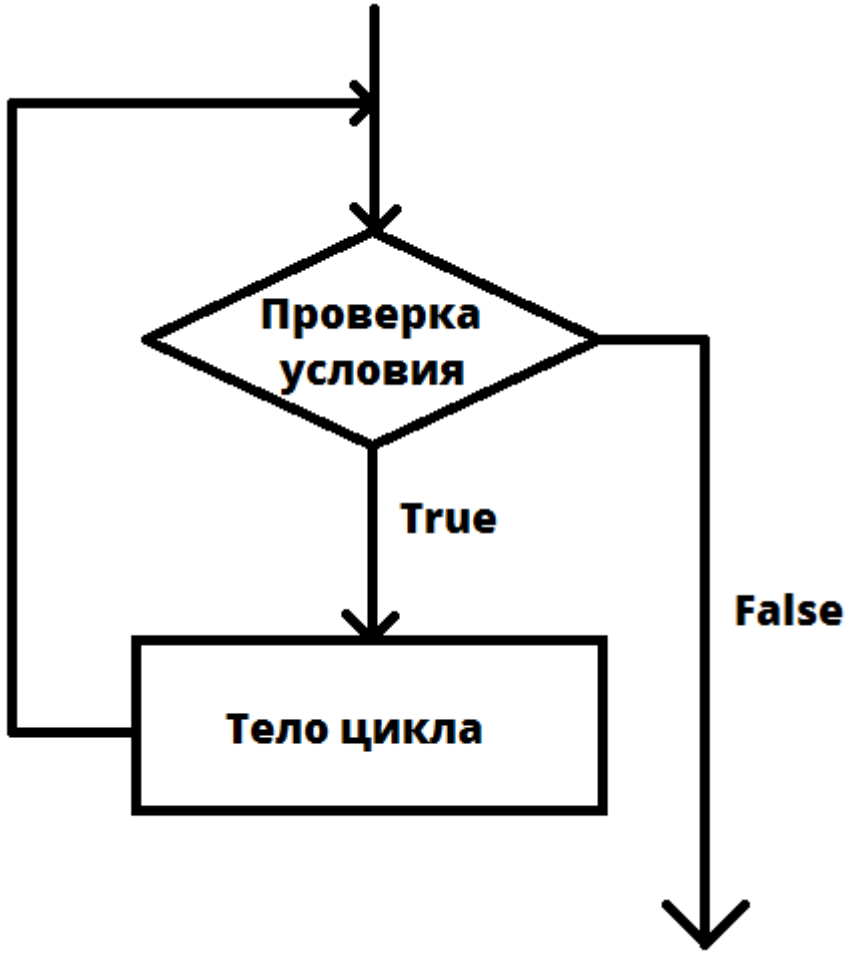
```
label:
    /* код */
goto label
```

* label – обычный идентификатор

Оператор цикла

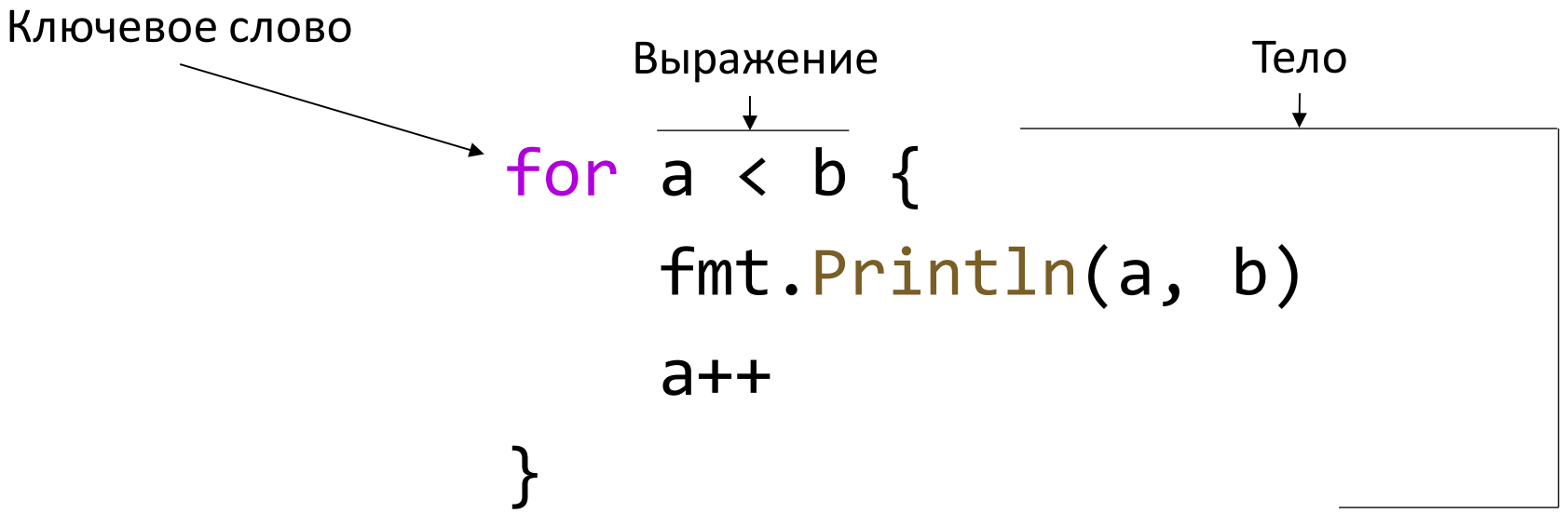


Оператор цикла



for

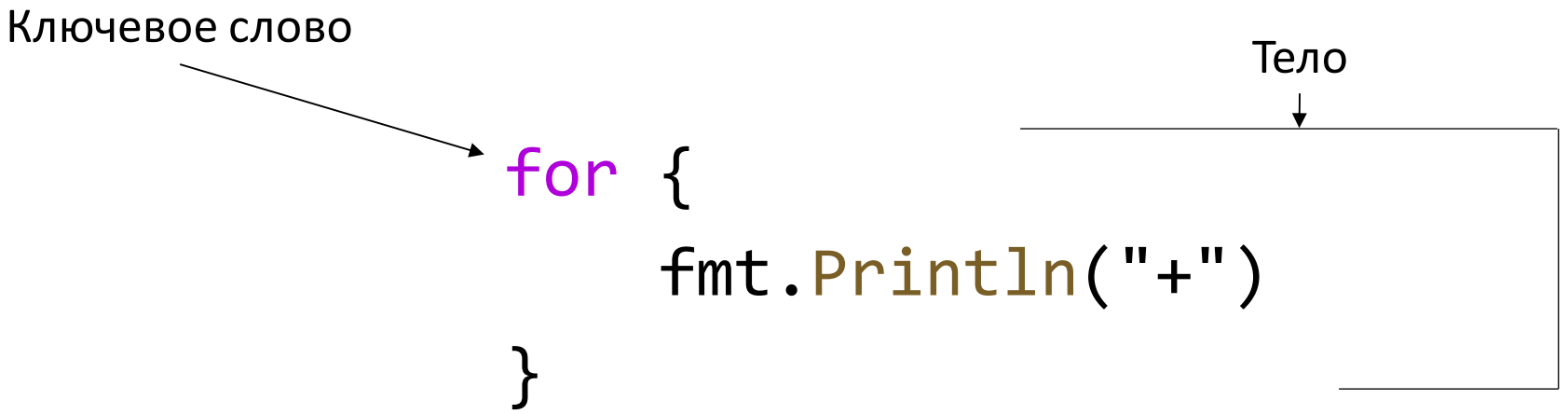
for



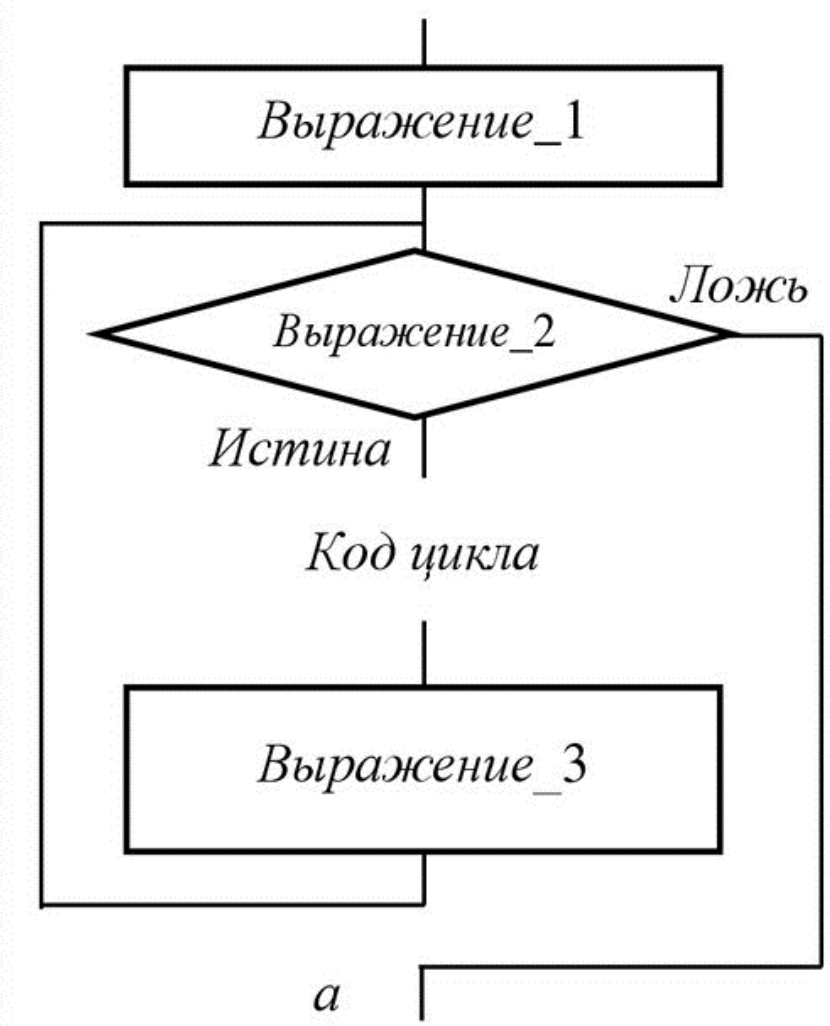
for выражение

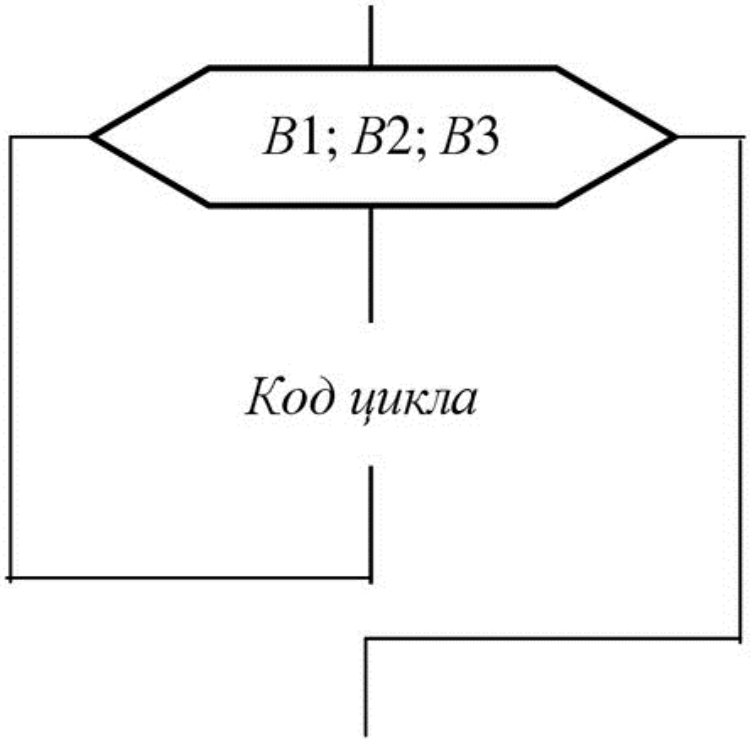
Ожидается, что выражение в скобочках типа `bool`.

Бесконечный цикл



Оператор цикла for

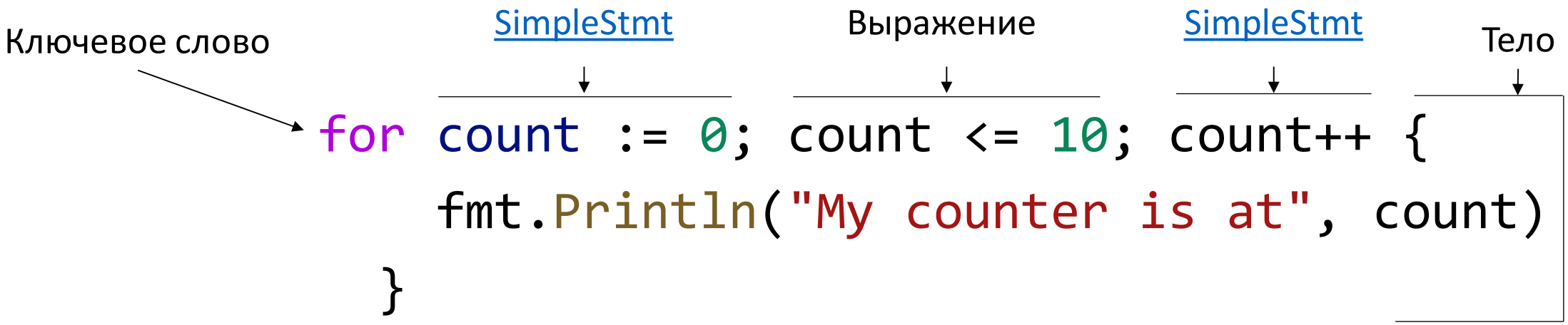




B1, B2, B3 - Выражения 1,2,3

б

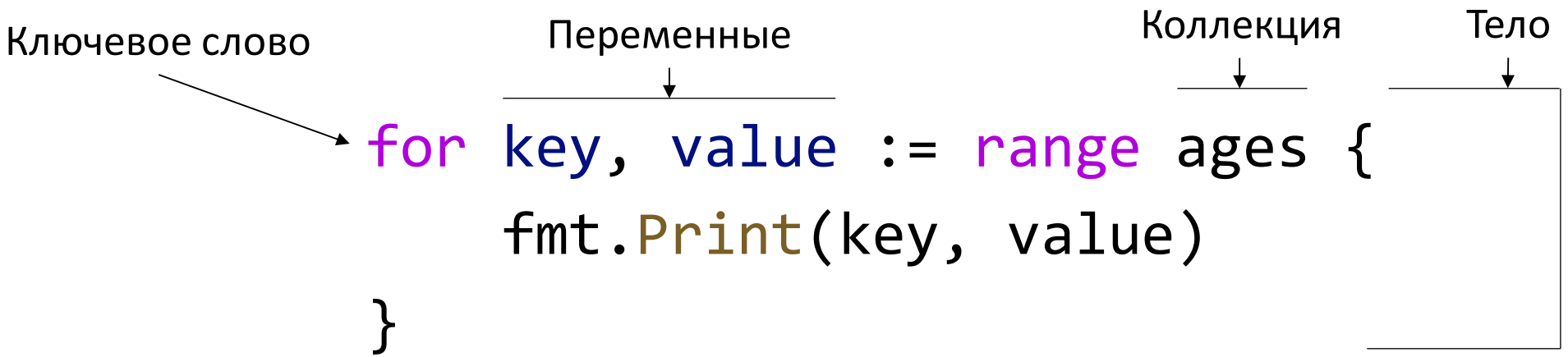
for


Ключевое слово
SimpleStmt
Выражение
SimpleStmt
Тело
for count := 0; count <= 10; count++ {
fmt.Println("My counter is at", count)
}

for SimpleStmt; Выражение; SimpleStmt

Каждое из выражение не обязательное (можно не писать), но точки с запятой писать нужно.

for RangeClause



for RangeClause пример - строка

```go
package main

import "fmt"

func main() {
    // Пример итерации по
    str := "Hello, World!"
    for index, char := range str {
        fmt.Printf("Индекс: %d, Значение: %c\n", index, char)
    }
}
```

for RangeClause пример - срез

```go
package main

import "fmt"

func main() {
    numbers := []int{1, 2, 3, 4, 5}

    // Пример итерации по срезу
    for index, value := range numbers {
        fmt.Printf("Индекс: %d, Значение: %d\n", index, value)
    }
}
```

for RangeClause пример - словарь

```go
package main

import "fmt"

func main() {
    // Пример итерации по словарю
    ages := map[string]int{
        "John":  25,
        "Alice": 30,
        "Bob":   35,
    }
    for name, age := range ages {
        fmt.Printf("Имя: %s, Возраст: %d\n", name, age)
    }
}
```

Строки

String literals (escape characters)

Expression	Result	Note
`""`		Default zero value for type `string`
`"Japan 日本"`	Japan 日本	Go code is Unicode text encoded in UTF-8
`"\xe6\x97\xa5"`	日	`\xNN` specifies a byte
`"\u65E5"`	日	`\uNNNN` specifies a Unicode value
`"\\"`	\	Backslash
`"\""`	"	Double quote
`"\n"`		Newline
`"\t"`		Tab
`` `\xe6` ``	\xe6	Raw string literal*
`html.EscapeString("<>")`	<>	HTML escape for <, >, &, ' and "
`url.PathEscape("A B")`	A%20B	URL percent-encoding `net/url`

* In `` ` `` string literals, text is interpreted literally and backslashes have no special meaning. See Escapes and multiline strings for more on raw strings, escape characters and string encodings.

Строки

Concatenate

Expression	Result	Note
`"Ja" + "pan"`	Japan	Concatenation

> **Performance tips**
> See 3 tips for efficient string concatenation for how to best use a string builder to concatenate strings without redundant copying.

Equal and compare (ignore case)

Expression	Result	Note
`"Japan" == "Japan"`	true	Equality
`strings.EqualFold("Japan", "JAPAN")`	true	Unicode case folding
`"Japan" < "japan"`	true	Lexicographic order

Строки

Length in bytes or runes

Expression	Result	Note
`len("日")`	3	Length in bytes
`utf8.RuneCountInString("日")`	1	in runes `unicode/utf8`
`utf8.ValidString("日")`	true	UTF-8? `unicode/utf8`

Строки

Length in bytes or runes

Expression	Result	Note
`len("日")`	3	Length in bytes
`utf8.RuneCountInString("日")`	1	in runes `unicode/utf8`
`utf8.ValidString("日")`	true	UTF-8? `unicode/utf8`

Строки

Index, substring, iterate

Expression	Result	Note
`"Japan"[2]`	'p'	Byte at position 2
`"Japan"[1:3]`	ap	Byte indexing
`"Japan"[:2]`	Ja	
`"Japan"[2:]`	pan	

A Go range loop iterates over UTF-8 encoded characters (runes):

```
for i, ch := range "Japan 日本" {
    fmt.Printf("%d:%q ", i, ch)
}
// Output: 0:'J' 1:'a' 2:'p' 3:'a' 4:'n' 5:' ' 6:'日' 9:'本'
```

Iterating over bytes produces nonsense characters for non-ASCII text:

```
s := "Japan 日本"
for i := 0; i < len(s); i++ {
    fmt.Printf("%q ", s[i])
}
// Output: 'J' 'a' 'p' 'a' 'n' ' ' 'æ' '\u0097' '¥' 'æ' '\u009c' '¬'
```

Строки

Search (contains, prefix/suffix, index)

Expression	Result	Note
`strings.Contains("Japan", "abc")`	false	Is abc in Japan?
`strings.ContainsAny("Japan", "abc")`	true	Is a, b or c in Japan?
`strings.Count("Banana", "ana")`	1	Non-overlapping instances of ana
`strings.HasPrefix("Japan", "Ja")`	true	Does Japan start with Ja?
`strings.HasSuffix("Japan", "pan")`	true	Does Japan end with pan?
`strings.Index("Japan", "abc")`	-1	Index of first abc
`strings.IndexAny("Japan", "abc")`	1	a, b or c
`strings.LastIndex("Japan", "abc")`	-1	Index of last abc
`strings.LastIndexAny("Japan", "abc")`	3	a, b or c

Строки

Replace (uppercase/lowercase, trim)

Expression	Result	Note
`strings.Replace("foo", "o", ".", 2)`	f..	Replace first two “o” with “.” Use -1 to replace all

`f := func(r rune) rune {`
`  return r + 1`
`}`
`strings.Map(f, "ab")`
	bc	Apply function to each character

`strings.ToUpper("Japan")`	JAPAN	Uppercase
`strings.ToLower("Japan")`	japan	Lowercase
`strings.Title("ja pan")`	Ja Pan	Initial letters to uppercase
`strings.TrimSpace(" foo\n")`	foo	Strip leading and trailing white space
`strings.Trim("foo", "fo")`		Strip *leading and trailing* f:s and o:s
`strings.TrimLeft("foo", "f")`	oo	*only leading*
`strings.TrimRight("foo", "o")`	f	*only trailing*
`strings.TrimPrefix("foo", "fo")`	o	
`strings.TrimSuffix("foo", "o")`	fo	

Строки

Split by space or comma

Expression	Result	Note
`strings.Fields(" a\t b\n")`	`["a" "b"]`	Remove white space
`strings.Split("a,b", ",")`	`["a" "b"]`	Remove separator
`strings.SplitAfter("a,b", ",")`	`["a," "b"]`	Keep separator

Join strings with separator

Expression	Result	Note
`strings.Join([]string{"a", "b"}, ":")`	a:b	Add separator
`strings.Repeat("da", 2)`	dada	2 copies of “da”

Строки

Format and convert

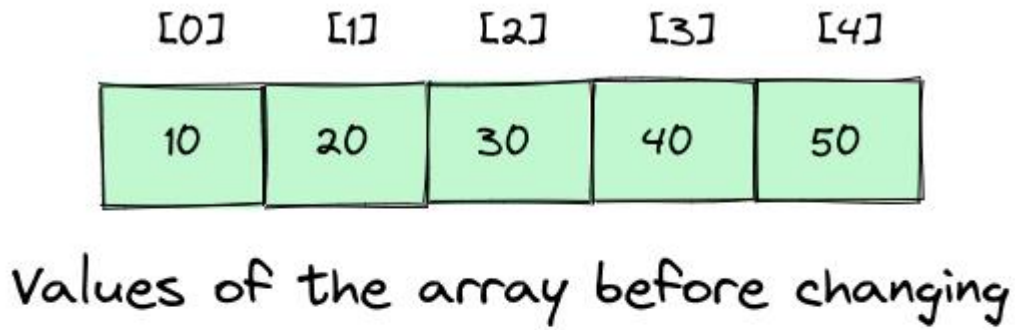
Expression	Result	Note
`strconv.Itoa(-42)`	`"-42"`	Int to string
`strconv.FormatInt(255, 16)`	`"ff"`	Base 16

Sprintf

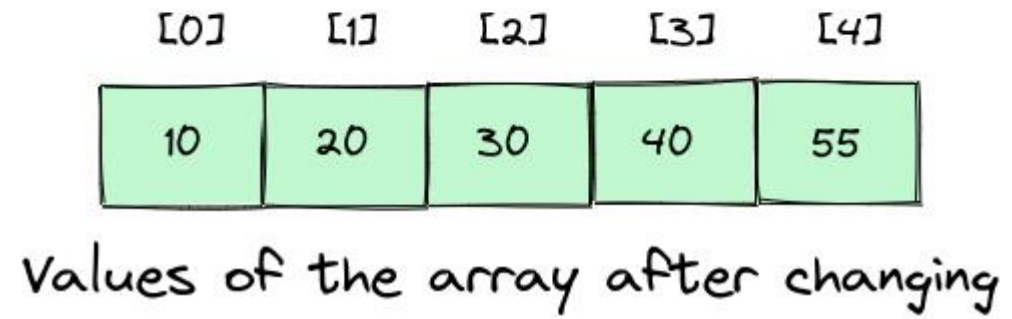
The `fmt.Sprintf` function is often your best friend when formatting data:

```
s := fmt.Sprintf("%.4f", math.Pi) // s == "3.1416"
```

Массивы



Values of the array before changing



Values of the array after changing

Массивы

```
// Массив чисел
// Длина фиксирована и = 8
// Все элементы типа int
// Все элементы инициализированы нулевым значением
var planets [8]int
fmt.Print(planets)
```

Массивы

```
// Вначале пустой
var planets [8]string
planets[0] = "Меркурий" // Присваивает планете индекс 0
planets[1] = "Венера"
planets[2] = "Земля"

// Ещё один массив
planets := [5]string{"Церера", "Плутон", "Хаумеа", "Макемаке", "Эрида"}

// Длина определяется по количеству элементов
planets := [...]string{ // Компилятор Go подсчитывает элементы
    "Меркурий",
    "Венера",
    "Земля",
    "Марс",
    "Юпитер",
    "Сатурн",
    "Уран",
    "Нептун", // Запятая в самом конце является обязательной
}
```

Массивы

```
planets := [...]string{
    "Меркурий",
    "Венера",
    "Земля",
    "Марс",
    "Юпитер",
    "Сатурн",
    "Уран",
    "Нептун",
}

planetsMarkII := planets // Копирует массив planets

planets[2] = "упс" // Прокладывает путь для межзвездного шунтирования

fmt.Println(planets)       // Выводит: [Меркурий Венера упс Марс Юпитер Сатурн Уран Нептун]
fmt.Println(planetsMarkII) // Выводит: [Меркурий Венера Земля Марс Юпитер Сатурн Уран Нептун]
```

Массивы

```
planets := [...]string{
    "Меркурий",
    "Венера",
    "Земля",
    "Марс",
    "Юпитер",
    "Сатурн",
    "Уран",
    "Нептун",
}

planetsMarkII := planets // Копирует массив planets

planets[2] = "упс" // Прокладывает путь для межзвездного шунтирования

fmt.Println(planets)       // Выводит: [Меркурий Венера упс Марс Юпитер Сатурн Уран Нептун]
fmt.Println(planetsMarkII) // Выводит: [Меркурий Венера Земля Марс Юпитер Сатурн Уран Нептун]

var planetsMarkIII [8]string = planets  // Тоже работает
var planetsMarkIV [10]string = planets  // Ошибка
```

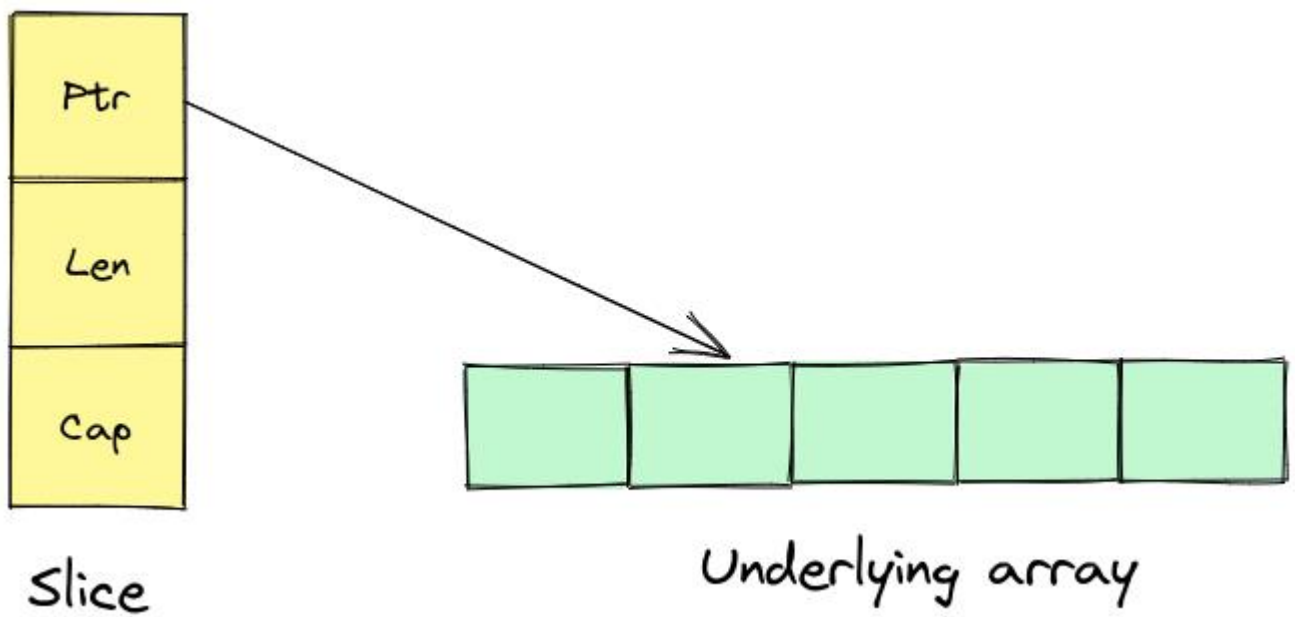
Массивы многомерные

```
var board [3][3]string // Массив из 3 массивов с 3 строками

 // Координаты [ряд][столбец]
board[0][0] = "1"
board[0][1] = "2"
board[0][2] = "3"
board[1][0] = "4"
board[1][1] = "5"
board[1][2] = "6"
board[2][0] = "7"
board[2][1] = "8"
board[2][2] = "9"

for i:=0; i<3; i++ {
    fmt.Println(board[i][0], board[i][1], board[i][2])
}
```

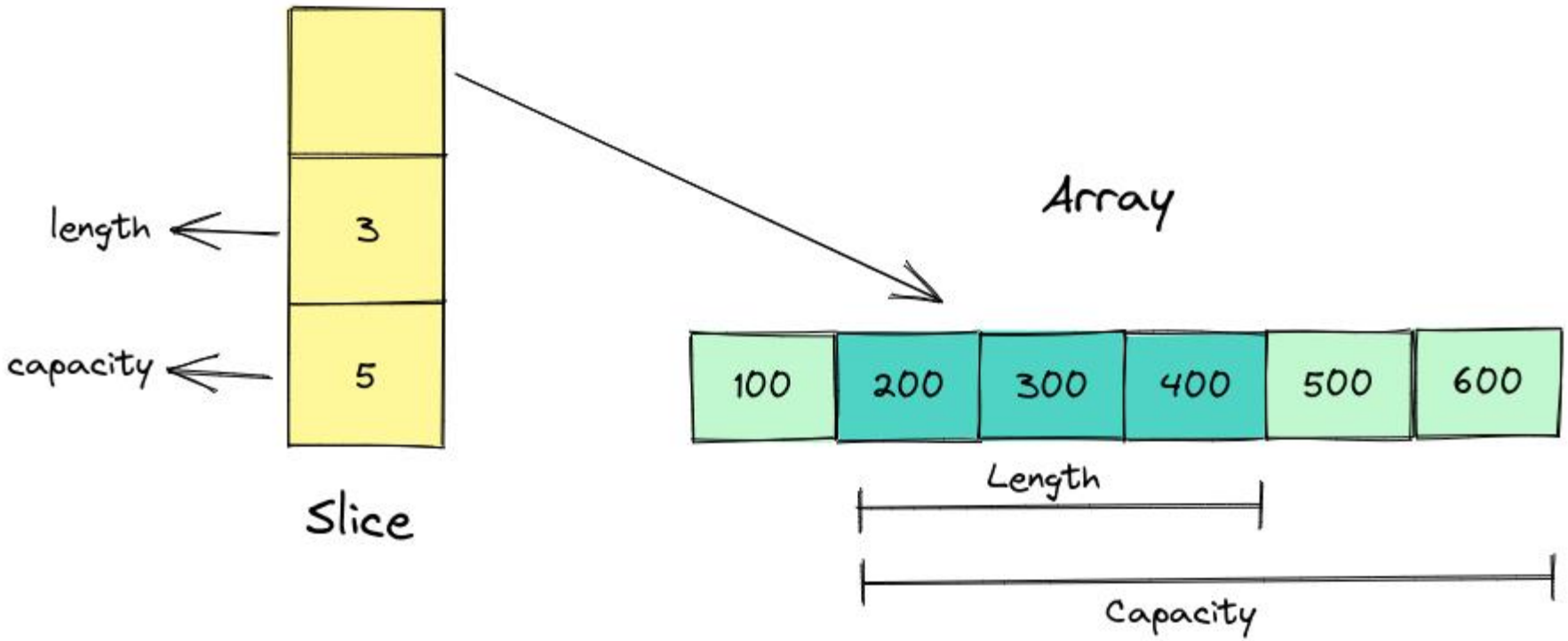
Массивы многомерные

```
var board =  [3][3]int{{1,2,3},
                       {4,5,6},
                       {7,8,9}}

for i:=0; i<3; i++ {
    fmt.Println(board[i][0], board[i][1], board[i][2])
}
```

Срез (слайс)



Срез (слайс)


length
3
capacity
5
Slice
Array
100
200
300
400
500
600
Length
Capacity

Слайс

```
// Срез чисел
// Длина может изменяться
// Все элементы типа int
// Сейчас никуда не ссылается
var planets []int
fmt.Print(planets == nil)
```

Слайс

```
// make создаст новый слайс для 10 элементов и 3 из них занулит
// Сейчас ссылается на массив
var planets []int = make([]int, 3, 10)
fmt.Print(planets, planets == nil)
```

Слайс

```
var planets []int = []int{1, 2, 3, 4, 5}
planetsII := []int{1, 2, 3, 4, 5}
planetsIII := planetsII

// planetsII и planetsIII ссыдаются на один и тот же массив
fmt.Print(planets, planetsII, planetsIII)
```

Слайс

```
planets  := [5]int{1, 2, 3, 4, 5} // Это НЕ слайс, это массив
other := planets[0 : 2]
fmt.Println(planets, other)

planets[0] = 10
fmt.Println(planets, other)

other[0] = -10
fmt.Println(planets, other)
```

Слайс

```
planets  := [5]int{1, 2, 3, 4, 5} // Это НЕ слайс, это массив
other := planets[0:2]    // [1, 2]
otherI := planets[:2]    // [1, 2]
otherII := planets[2:]   // [3, 4, 5]
otherIII := planets[:]   // [1, 2, 3, 4, 5]
```

Многомерный слайс

```
seaNames := [][]string{{"shark", "octopus", "squid", "mantis shrimp"},
                       {"Sammy", "Jesse", "Drew", "Jamie"}}

fmt.Print(seaNames)
```

Слайс – основные функции

```go
planets := []int{1, 2, 3, 4, 5}

// len - текущая длина слайса
// cap - ёмкость
fmt.Println(planets, len(planets), cap(planets)) // [1, 2, 3, 4, 5] 5 5

planets = append(planets, 9)
fmt.Println(planets, len(planets), cap(planets)) // [1, 2, 3, 4, 5, 9] 6 10

newPlanets := make([]int, 3, 20)
copy(newPlanets, planets)
fmt.Println(newPlanets, len(newPlanets), cap(newPlanets)) // [1, 2, 3] 3 20

newPlanets = append(newPlanets, planets...)
fmt.Println(newPlanets, len(newPlanets), cap(newPlanets)) // [1 2 3 1 2 3 4 5 9] 9 20

// Удалим 2 элемент из planets
planets = append(planets[:2], planets[3:]...)
fmt.Println(planets)  // [1, 2, 4, 5, 9]
```

https://pkg.go.dev/slices

Слайс – append особенности

```
array := [5]int{5, 5, 5, 5, 5}
sl := array[:3]
fmt.Println(sl) // [5, 5, 5]

sl = append(sl, 10)
fmt.Println(array, sl) // [5, 5, 5, 10, 5] [5, 5, 5, 10]

sl = append(sl, 11, 12) // Слайс переполнится и создаст другой массив для себя
fmt.Println(array, sl)  // [5, 5, 5, 10, 5] [5, 5, 5, 10, 11, 12]
```